ACCELERATOR™
DEVELOPMENT SOLUTIONS

# MAGIC User Conference Lab

## .NET and the IBM i for Modern Desktop and Mobile Applications


Version 1.0
9/27/2017

## Table of Contents

# 1. Prepare Development Environment

For this Lab, several prerequisites are required:

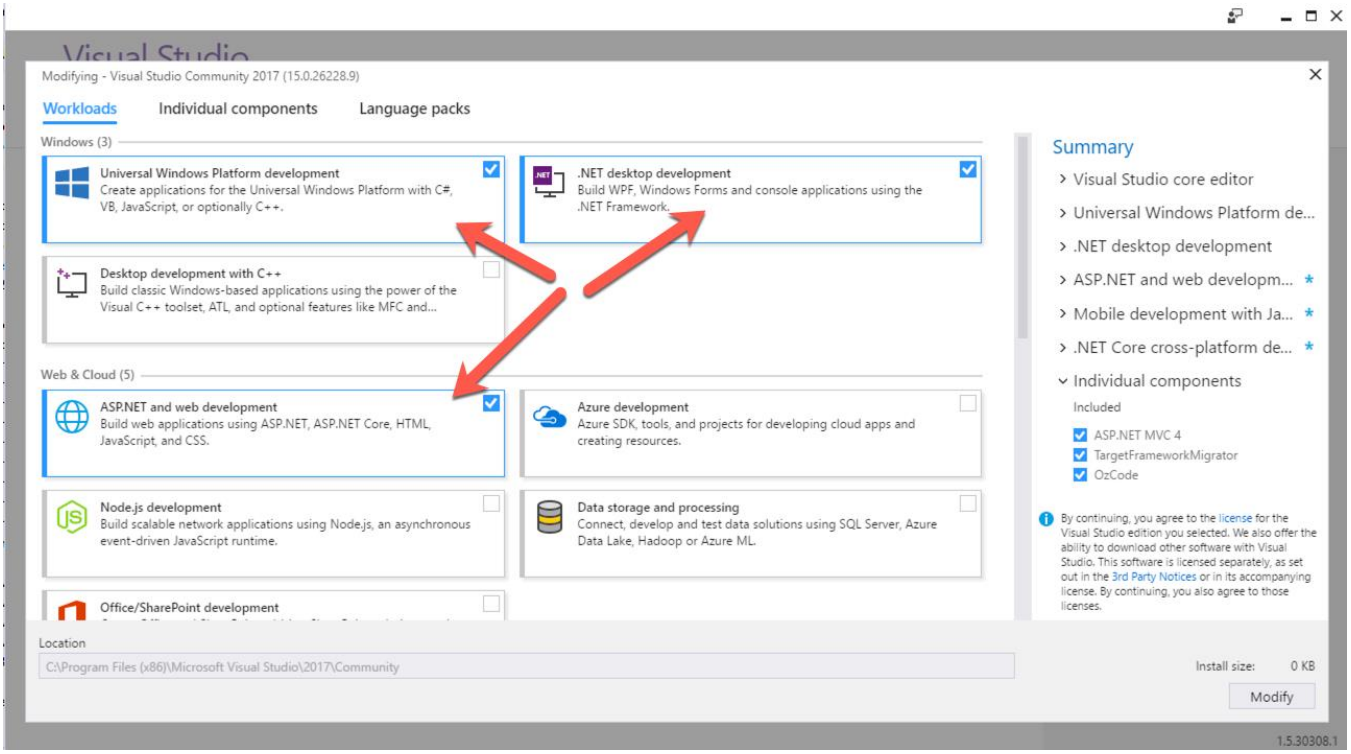
- Visual Studio 2013 or greater
- the IBM i .NET Data Provider (for communication between .NET and DB2)
- The Accelerator by Surround Technologies
- Accelerator Generation Standards (includes Open Generation Technology plugin for Cloud Services)
- A Provisioned Environment (Database and Framework Web Services for Surround Architecture)

If you are attending a pre-arranged Lab with a Surround Technologies Instructor, these prerequisites will have been provided to you, and the remainder of this section is only for your reference.

## 1.1. Visual Studio 2017 Community

Visual Studio 2013, 2015, and 2017 are supported for use with Accelerator. If you do not already have Visual Studio, you may download Visual Studio 2017 Community for free: https://www.visualstudio.com/vs/community/

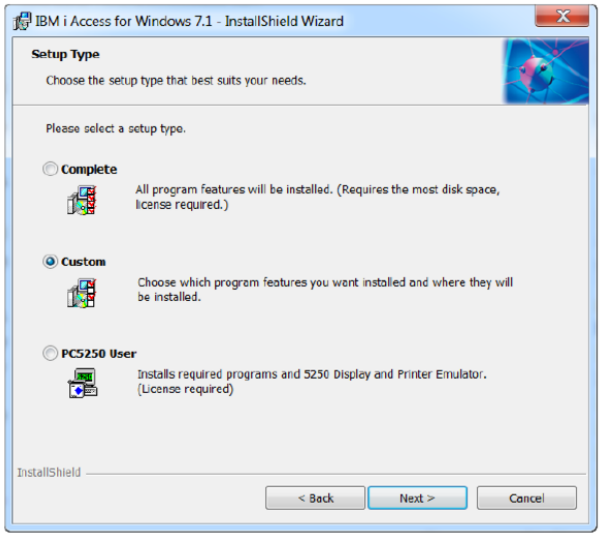
When installing Visual Studio, three Workloads are required: Universal Windows Platform development, .NET desktop development, and ASP.NET and web development.

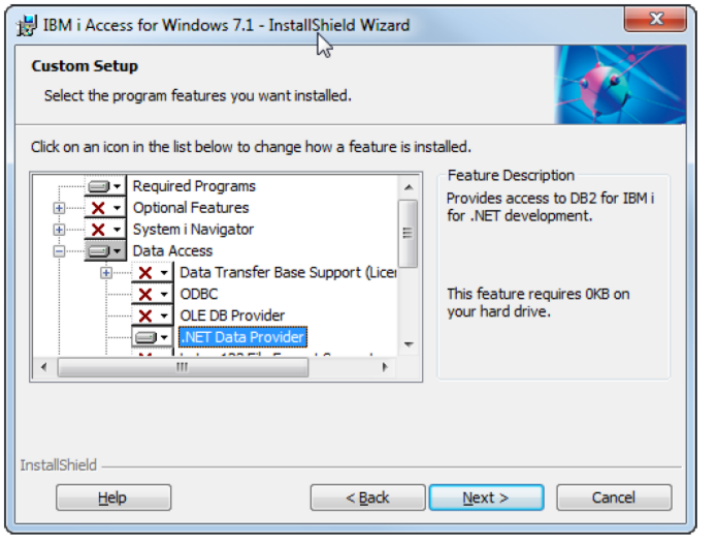


## 1.2. IBM i .NET Data Provider

You can download the IBM i .NET data provider directly from the IBM website. Here is the URL for your convenience: https://www-03.ibm.com/systems/power/software/i/access/windows/dotnet.html

The provider can be installed as part of the IBM i Access for Windows. Follow the on-screen instructions, and be sure to select Custom Install on the Setup Type screen. Make sure the .NET Data Provider in the Data Access section is selected and click the **Next** button. The Accelerator only requires that **Required Programs** and **.NET Data Provider** be installed. You may choose not to install the other items. Continue to finalize the installation.



## 1.3. Accelerator, Generation Standards, Provisioned Environment

If you have signed up for an Accelerator Trial at http://surroundtech.com/accelerator-trial, you will have received an email like the one pictured at right. This email includes download links for Accelerator, the Generation Standards, a Quick Start Guide (not needed for this Lab), and a URL and credentials for a Provisioned Environment. There are also links to some video tutorials, which can also be found on the Surround Technologies website.

If you are attending a pre-arranged Lab with a Surround Technologies Instructor, your workstation will already have Accelerator and the Generation Standards installed, and your instructor will provide you with the URL and credentials for your environment.

WOOHOO! YOU'RE ALL SET!
Your Surround Technologies Accelerator Environment is ready to go!

SURROUND TECHNOLOGIES™
*Creating software people love!*




Douglas Webb,
Developing smarter starts now!
Your personal Accelerator environment has been created!
Follow the steps below to get started.

**Step 1: Download The Accelerator**
The Accelerator is available in both 32-bit and 64-bit versions. Please download the version which best suits your systems configuration. If you are unsure on your system configuration right-click on Your Computer and Select Properties.

Download 32-bit Version for x86 systems | or | Download 64-bit Version for x64 systems

**Step 2: Grab the Generation Standards**
We've gone ahead and setup some pre-configured standards to get you generating faster. Start with these, and save new standards as you need. Download the Generation Standards ZIP file below and follow the instructions in the Quick Start guide to configure the Accelerator for your environment.

Download Generation Standards
ZIP File Archive

**Step 3: Last Step. Get Going with Quick Start Guide and your Credentials**
Our Quick Start Guide is as simple as this email. It'll guide you through everything you'll need to start developing with the Accelerator using Microsoft SQL Server or DB2 on your IBM i. Feel free to use your own tables, or use the Quick Start guide to download our free demo database called EasyBuy and you'll be generating Windows, Web and Mobile systems in minutes.

Download Quick Start Guide
PDF File

**Accelerator Services Connection Credentials**
You will need the following credentials during the configuration and setup of the Accelerator. Use the following to access your individually provisioned Accelerator Services Environment.

***Username:*** A4DNUser
***Password:*** demo
***Accelerator Address:*** http://services.surroundtech.com/Provisions/Surround_Technologies_Webb_v7_2_0_0_24719/

**Video Tutorials give you a running start**
Developer see, Developer do. Our video tutorials make working with the Accelerator simple and efficient. Learn about everything from installation through generation. We're so confident in our methodology, we have our ***UX designer*** guiding ***Developers*** through generating systems!



Installing the Accelerator



Configuring the Accelerator.



Building and Generating a new System in minutes.

For additional help getting started with or generating systems with the Accelerator, visit our videos page for guided tours of the Accelerator and its features and functionality!

You can find more detailed information about developing for the Accelerator, as well as tutorials to help you get started, on our documentation page located at www.AcceleratorTours.com (registration is required).

If you have any questions about your environment, or would like to speak to someone at Surround Technologies about the Accelerator, you can contact us at support@surroundtech.com.

*From all of us at Surround Technologies, thank you for choosing the Accelerator!*

Follow on Twitter | Friend on Facebook


Generated by on 4/9/2017 10:55:35 AM

# 2. Overview of the Database and Desired Application

We will be building an Enterprise application for managing data in the Cloud Services 24x7 database.

This Legacy system consists of 4 physical files, with 45 fields, and 3 file relationships.


The new application has the following requirements:

1. This application must prompt for a Customer number. If the customer number is valid, the application will present a screen that will allow the user to change an order or enter a new one.
2. The application must allow the user to add, change, and delete orders.
3. Orders must allow for multiple line items.
4. The order items should include an image of the part.
5. An order should be exportable to a spreadsheet directly from the screen.

# 3. Building your First Application (without Accelerator)

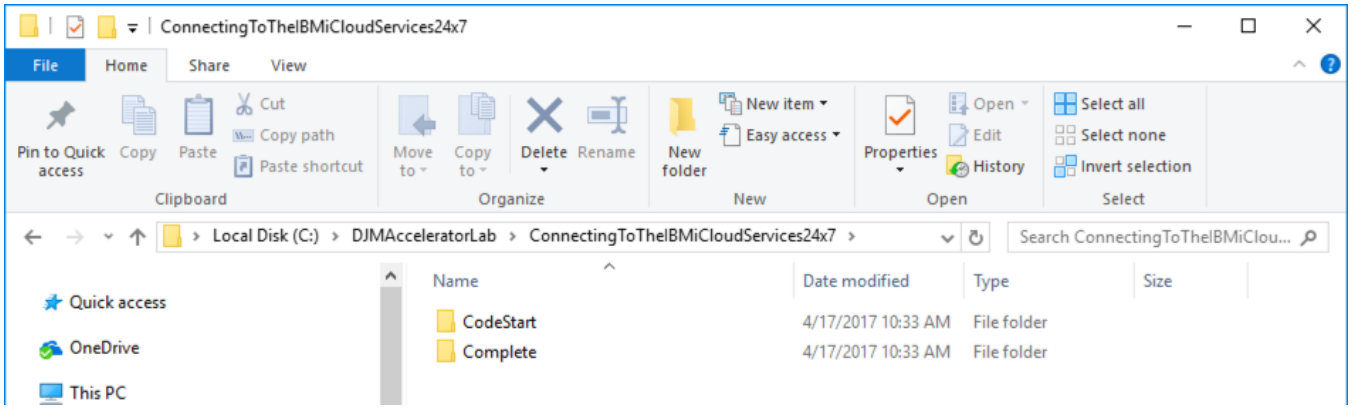
This section will walk you through how to write .NET code to access data on the IBM i. We will be using the Cloud Services 24x7 database in this example. We have provided a code start example that you can download to help save time and allow us to focus on the connectivity code. This example will provide the basics needed to get started with talking to the IBM i from .NET.

## 3.1. Download Code Start

1. Create a new folder on the C Drive using your initials followed by "AcceleratorLab" ex: C:**<YourInitials>**AcceleratorLab
2. Download the code start from this URL and save the zip file to the folder you created in step 1 above:

http://services.surroundtech.com/downloads/Products/a4dn/ConnectingToTheIBMiCloudServices24x7.zip

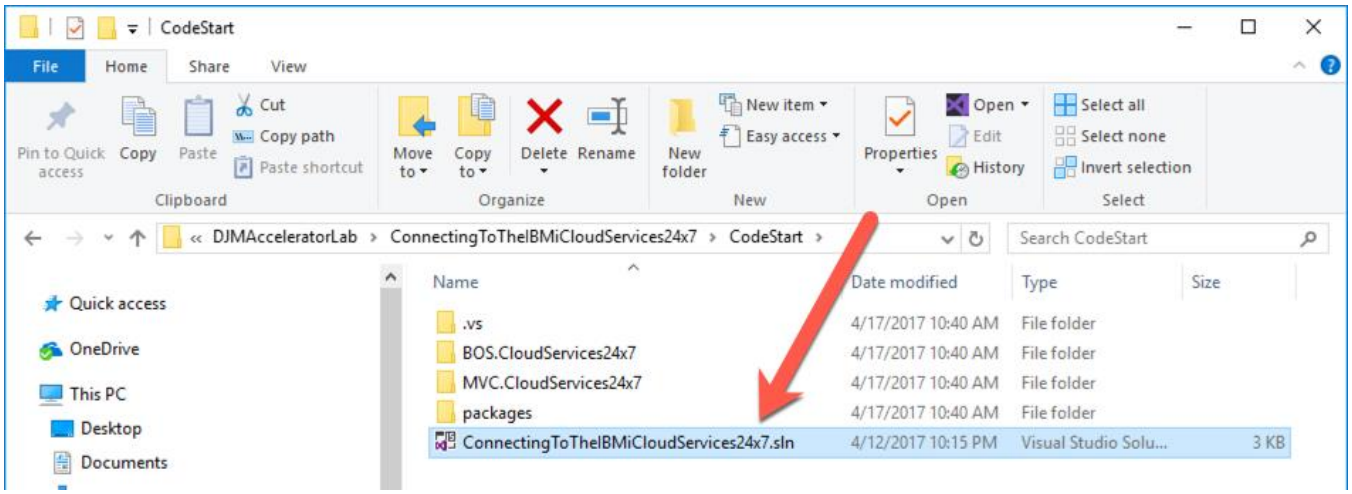
3. Unzip the IBMiCloudServices24x7.zip file to C:**<YourInitials>**AcceleratorLab. You should now have a folder "ConnectingToTheIBMiCloudServices24x7" that contains 2 folders: "CodeStart" and "Complete".



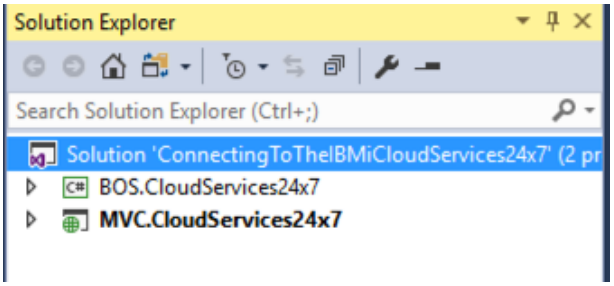

The "CodeStart" folder contains a starting point for this lab. It contains the Entity (Model) and Data Access Layer (DAL) classes for 2 of the files in the Cloud Services 24x7 database: Customer (CSCSTP) and Orders (CSORDP). It also contains a generated ASP.NET MVC CRUD maintenance for the files. The Data Access Layer classes are partially written and this lab will guide you through the steps to complete the programming logic. The Complete folder contains all the code in a runnable completed state for your reference.

## 3.2. What is in the CodeStart?

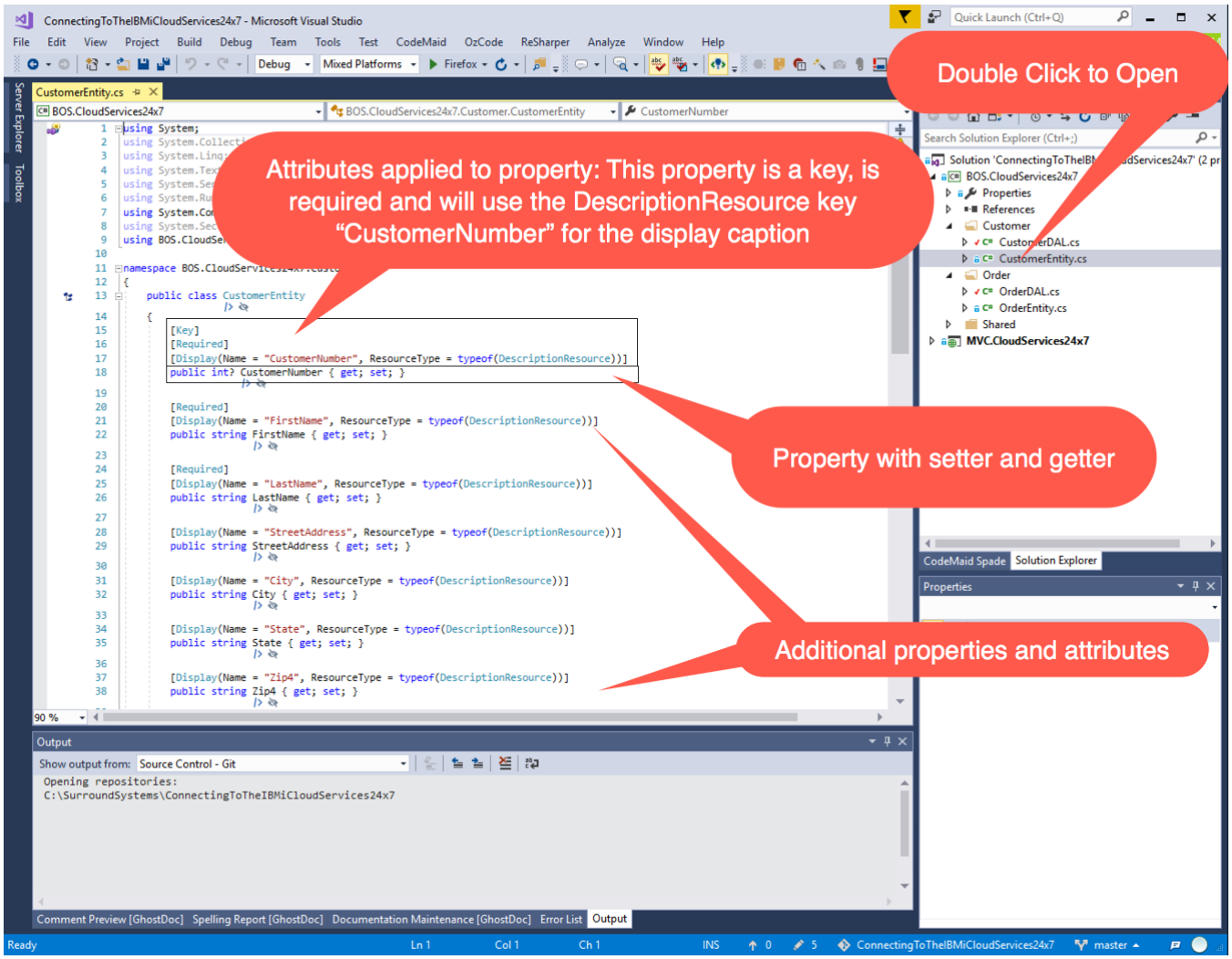
1. In the CodeStart folder, open the solution in Visual Studio by double clicking on the solution file: ConnectingToTheIBMiCloudServices24x7.sln




2. There are 2 projects in this solution:
   - BOS.CloudServices24x7 – This contains the Entity and Data Access Layer for both the Customers and Orders. This is separated out so that it is centralized and can be accessed by multiple UI. We will be doing some coding in the Data Access Layer.
   - MVC.CloudServices24x7 – This is the ASP.NET MVC UI that calls the Data Access Layer through the controller and outputs a view. In the MVC pattern, the Model is the entity defined in the BOS.CloudServices24x7 project. The View and the Controller were generated based on the model. We will be reviewing some code in the controller that calls the methods in the Data Access Layer.

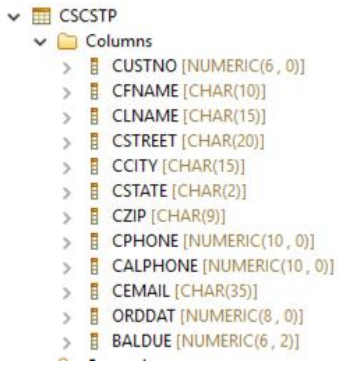


3. Expand the BOS.CloudServices24x7 project and then expand the Customer project folder. Double click on the CustomerEntity.cs to open the class. This class is composed of public properties that have a specific data type where the value can be set and get. The keywords in brackets are called Data Annotation Attributes and are added before the property. ASP.NET MVC data controls will use this metadata. You can read more about it here: https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations(v=vs.110).aspx

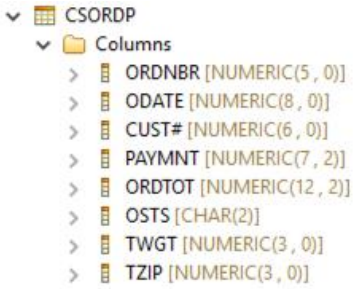



4. The strongly typed .NET data types in the entity are based on the DB2 data type for each column. Here is the convention we used for the Cloud Services files. The *decimal* .NET data type was used for all *numeric* DB2 data types except for keys and dates. The Customer Number uses the *int* data type and the Last Order Date uses the *DateTime* data type. All *character* DB2 data types use the *string* .NET data type.

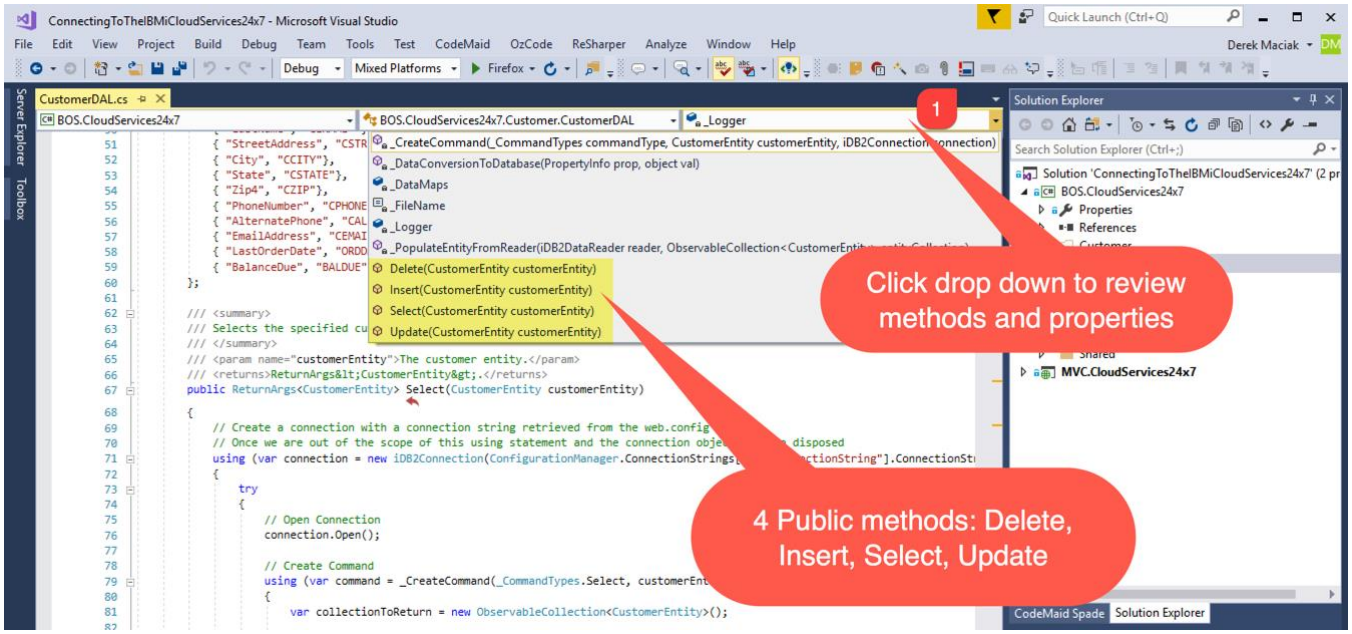
Here is the customer file schema.




5. Here is the Order file schema. Open the OrderEntity to review the corresponding .NET data types.

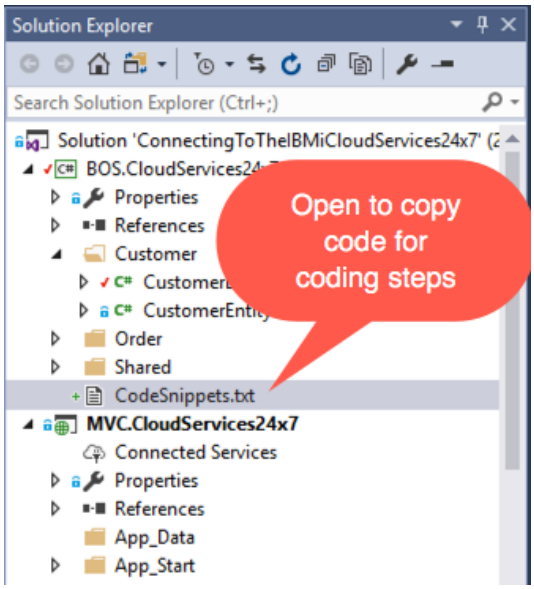



6. Open the CustomerDAL.cs class. Click on the drop down to review the methods and properties in the class. The customer data access layer contains the code to select, insert, update and delete customers.

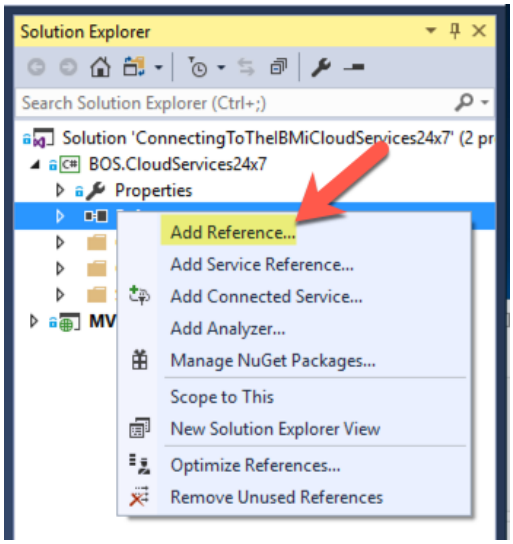


## 3.3. Coding the Data Access Layer

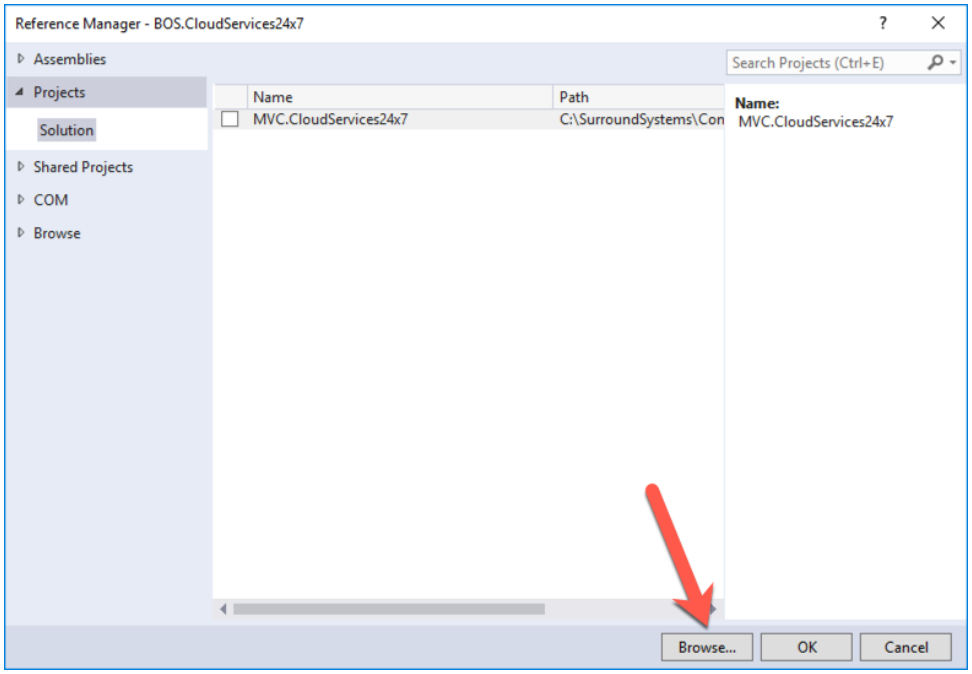
1. There is a text file called CodeSnippets.txt that you can open in Visual Studio. This will allow you to copy the code for each step instead of having to type it.



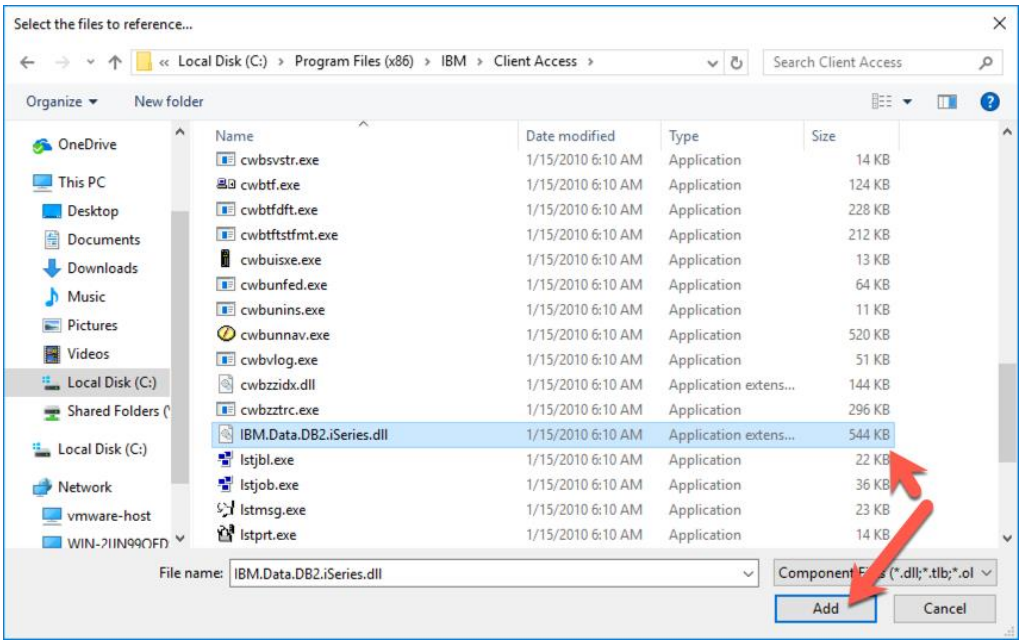

2. The first thing we need to do is to add a reference to the IBM.Data.DB2.iSeries.dll that was installed with the IBM i .NET Data Provider. This dll contains the ADO.NET interface and managed code to access the DB2 databases on the IBM i. To add a reference, right click on the References folder under the BOS.CloudServices24x7 project and click the “Add Reference…”.



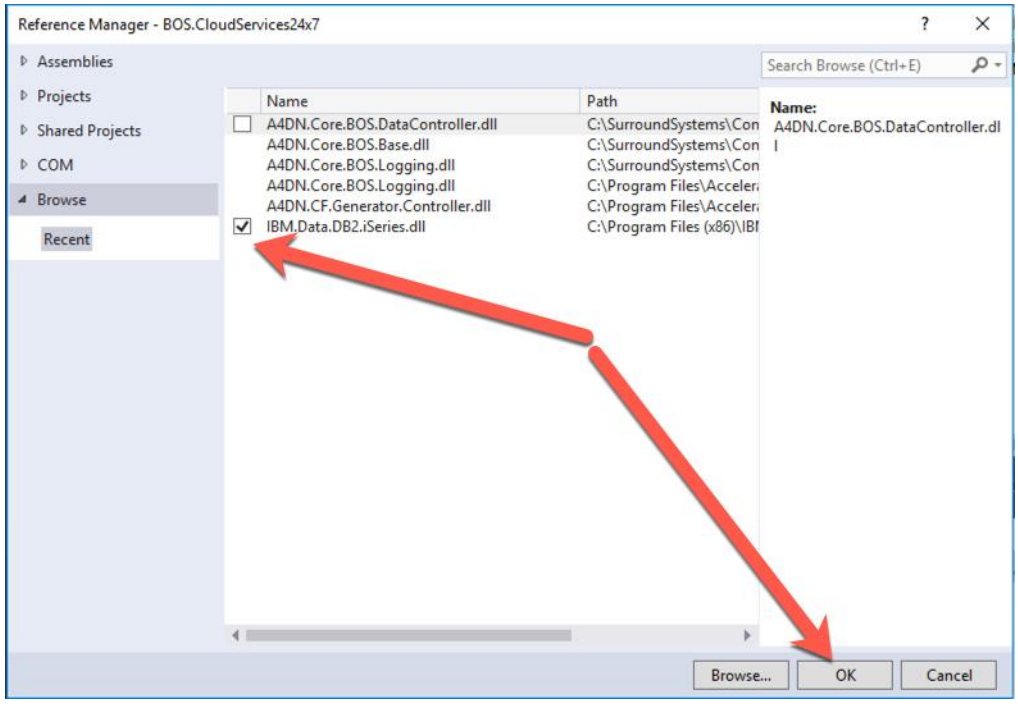
A dialog box will show. Click the browse button.




Navigate to "C:\Program Files (x86)\IBM\Client Access" folder and locate the IBM.Data.DB2.iSeries.dll. Select and click Add.



The DLL should automatically be checked. Click OK.




3. Open the CustomerDAL.cs code in Visual Studio if you don't already have it open.
4. Find the _FileName declaration around line 41, you need to specify the DB2 file name we will be accessing. In this case, we will be accessing the customer file CSCSTP. Add the highlighted code.

```
38      /// <summary>
39      /// The file name
40      /// </summary>
41      private const string _FileName = "CSCSTP";
42
```

5. Around Line 46, you need to specify how to map database column names from/to the .NET property names. This can be done with a dictionary collection where the key is the .NET property name and the value is the file column name. Add the highlighted mappings to the _DataMaps Dictionary:

```
43      /// <summary>
44      /// The data maps - this provides the maps from the database field name to the entity property name
45      /// </summary>
46      private readonly Dictionary<string, string> _DataMaps = new Dictionary<string, string>()
47      {
48          { "CustomerNumber","CUSTNO" },
49          { "FirstName","CFNAME" },
50          { "LastName", "CLNAME" },
51          { "StreetAddress", "CSTREET"},
52          { "City", "CCITY"},
53          { "State", "CSTATE"},
54          { "Zip4", "CZIP"},
55          { "PhoneNumber", "CPHONE"},
56          { "AlternatePhone", "CALPHONE"},
57          { "EmailAddress", "CEMAIL"},
58          { "LastOrderDate", "ORDDAT"},
59          { "BalanceDue", "BALDUE"}
60
61      };
```

6. The IBM i .NET Data Provider uses the ADO.NET Interface. The ADO.NET connection object allows you to specify the connection information to connect to the database. Soft coding the connection information allows you to easily configure the connection properties without having to build the project. The connection object for the IBM i .NET Data Provider is **iDB2Connection.** Here are the public properties and Methods of the connection object:

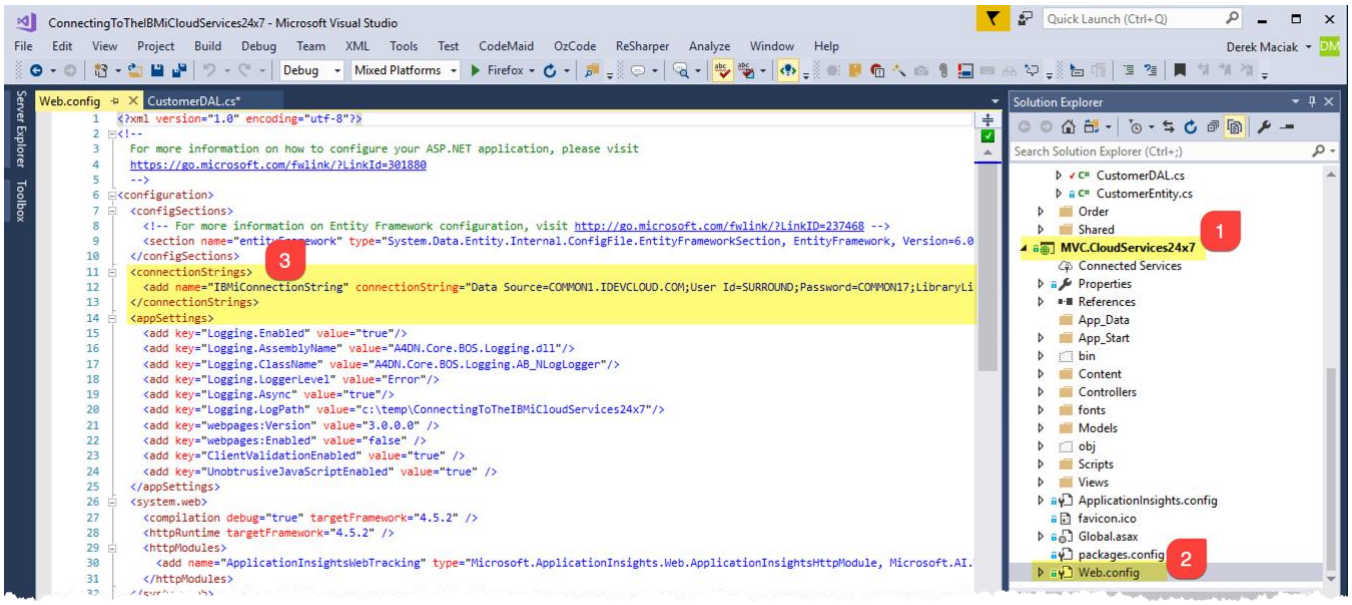
Public Properties

- **ConnectionString** - Use Provider specific connection string properties
- **ConnectionTimeout** – Gets the time to wait while establishing a connection before terminating and generating an error

Public Methods

- **Open()** – Opens a connection to the data source using the settings specified in the ConnectionString
- **Close()** – Closes a connection to the data source
- **CreateCommand()** – Creates a new iDB2Command object for use with this connection
- **BeginTransaction()** – Begins a database transaction for this connection using isolation level
- **Dispose()** - Releases the resources used by this connection.

The connection string information is in the web.config located in the MVC.CloudServices24x7 project. See if you can locate it by expanding the MVC project and double clicking on the web.config.




You need to add the "IBMiConnectionString" connection string name as the parameter to the iDB2Connection object class initializer. This will create an instance of the connection object using the parameters defined in the web.config. Add the following highlighted code.

```
63      /// <summary>
64      /// Selects the specified customer entity.
65      /// </summary>
66      /// <param name="customerEntity">The customer entity.</param>
67      /// <returns>ReturnArgs&lt;CustomerEntity&gt;.</returns>
68      public ReturnArgs<CustomerEntity> Select(CustomerEntity customerEntity)
69      {
70          // Create a connection with a connection string retrieved from the web.config
71          // Once we are out of the scope of this using statement and the connection object will be disposed
72          using (var connection = new iDB2Connection(ConfigurationManager.ConnectionStrings["IBMiConnectionString"].ConnectionString))
73          {
74              try
75              {
76
77                  return null;
78
79              }
80              catch (Exception ex)
81              {
82                  _Logger.am_WriteLog(GetType().Name, AB_LogLevel.Error, System.Reflection.MethodBase.GetCurrentMethod().Name, ex.Message);
83                  return new ReturnArgs<CustomerEntity>("ER", ex.Message);
84              }
85              finally
86              {
87              }
88          }
89      }
```

You will notice that the connection object instance is created within a using statement. It is a best practice

to declare and instantiate an IDisposable object in a using statement for the following reasons:

- Provides a convenient syntax that ensures the correct use of IDisposable objects
- Causes the object itself to go out of scope as soon as Dispose() is called
- Ensures that Dispose is called even if an exception occurs while you are calling methods in the object

7. Once we have the connection object instance, we need to call the methods to open and close the connection. The open connection should be placed in a try/catch statement so that any errors can be trapped and logged. The close connection should be placed in the finally statement so that it will always be executed whether or not an exception is thrown. The connection only needs to be closed if it is currently open. Add the following highlighted code.

```
63      /// <summary>
64      /// Selects the specified customer entity.
65      /// </summary>
66      /// <param name="customerEntity">The customer entity.</param>
67      /// <returns>ReturnArgs&lt;CustomerEntity&gt;.</returns>
68      public ReturnArgs<CustomerEntity> Select(CustomerEntity customerEntity)
69      {
70          // Create a connection with a connection string retrieved from the web.config
71          // Once we are out of the scope of this using statement and the connection object will be disposed
72          using (var connection = new iDB2Connection(ConfigurationManager.ConnectionStrings["IBMiConnectionString"].ConnectionString))
73          {
74              try
75              {
76                  // Open Connection
77                  connection.Open();
78
79                  return null;
80
81              }
82              catch (Exception ex)
83              {
84                  _Logger.am_WriteLog(GetType().Name, AB_LogLevel.Error, System.Reflection.MethodBase.GetCurrentMethod().Name, ex.Message);
85                  return new ReturnArgs<CustomerEntity>("ER", ex.Message);
86              }
87              finally
88              {
89                  // Close the connection
90                  if (connection.State != System.Data.ConnectionState.Closed)
91                  {
92                      connection.Close();
93                  }
94              }
95          }
96      }
```

8. After the connection is open, you need to create a command object instance. In ADO.net, the Command Object is used to execute SQL statements or Stored Procedures against a data source. The command object for the IBM i .NET Data Provider is **iDB2Command.** Here are the public properties and methods:

### Public Properties

- **CommandText** – Contains the SQL statement or stored procedure to run against a data source
- **CommandType** – Values of *StoredProcedure*, *TableDirect* or *Text* specify how the CommandText Property is interpreted
- **ConnectionTimeout** – maximum number of seconds to wait for the command to execute before terminating and generating an error. 0 = wait indefinitely.

- **Parameters** – Collection of iDB2Parameters. Used with Parameterized Queries.

Public Methods

- **Cancel()** – Attempts to cancel execution of a command
- **Dispose()** – Releases the resources used by this command
- **CreateParameter()** – Creates a new instance of an iDB2Paramter Object. The return value parameter could be input-only, output-only, bidirectional or a stored procedure. The default is Input.
- **ExecuteNonQuery()** – Executes the command and ignores any result set
- **ExecuteReader()** – Executes the command and returns an iDB2DataReader object
- **ExecuteScalar()** – Executes the command and returns the first column of the first row in the result set

Creating the command instance and calling the ExecuteReader() method on the command is done with a using statement so that the Dispose() method is called as soon as the object falls out of scope. Add the following highlighted code:

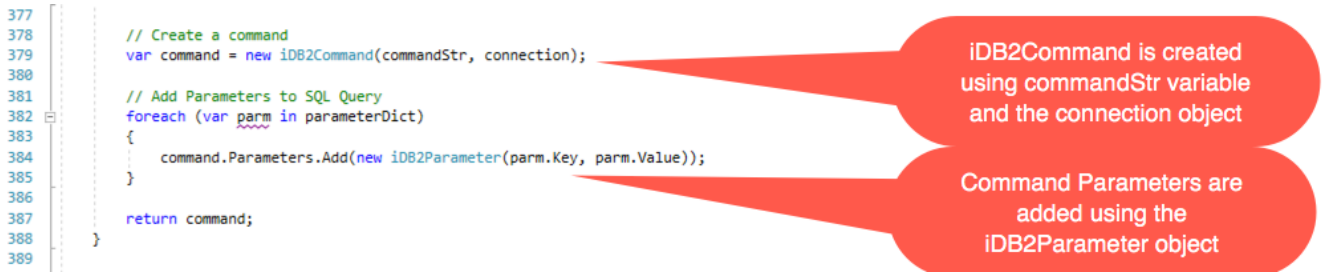
```
68      public ReturnArgs<CustomerEntity> Select(CustomerEntity customerEntity)
69      {
70          // Create a connection with a connection string retrieved from the web.config
71          // Once we are out of the scope of this using statement and the connection object will be disposed
72          using (var connection = new iDB2Connection(ConfigurationManager.ConnectionStrings["IBMiConnectionString"].ConnectionString))
73          {
74              try
75              {
76                  // Open Connection
77                  connection.Open();
78
79                  // Create Command
80                  using (var command = _CreateCommand(_CommandTypes.Select, customerEntity, connection))
81                  {
82                      var collectionToReturn = new ObservableCollection<CustomerEntity>();
83
84                      // Execute Command using ExecuteReader to return rows
85                      using (var reader = command.ExecuteReader())
86                      {
87                          if (!reader.HasRows)
88                          {
89                              return new ReturnArgs<CustomerEntity>("OK", collectionToReturn);
90                          }
91
92                          // Populate Entity with data from Reader
93                          _PopulateEntityFromReader(reader, collectionToReturn);
94                      }
95
96                      return new ReturnArgs<CustomerEntity>("OK", collectionToReturn); ;
97                  }
98
99              }
100             catch (Exception ex)
101             {
```

This code calls 2 private methods: _CreateCommand() and _PopulateEntityFromReader(). The _CreateCommand() method returns an **iDB2Command** object instance that is created using the command string, which is formatted based on the _CommandTypes enum (Values: Select, Insert, Update and Delete), and the connection instance. Take a look at the _CreateCommand() method. Here is some code that creates the commadStr variable for the Select statement. Here are a couple things we want to point out.


```
switch (commandType)
{
    case _CommandTypes.Select:

        // Loop through datamaps and get value from entity
        foreach (var map in _DataMaps)
        {
            // Get Property from Entity
            var prop = customerEntity.GetType().GetProperties().FirstOrDefault(p => p.Name == map.Key);
            if (prop == null) continue;

            var val = prop.GetValue(customerEntity, null);
            if (val == null) continue;

            if (prop.PropertyType == typeof(string))
            {
                // For Strings, use SQL LIKE
                fieldList.Add(string.Format("UPPER({0}) like {1}", _DataMaps[prop.Name], "@" + prop.Name));
                parameterDict.Add("@" + prop.Name, "%" + val.ToString().ToUpper() + "%");
            }
            else
            {
                // Otherwise use SQL Equal
                fieldList.Add(string.Format("{0} = {1}", _DataMaps[prop.Name], "@" + prop.Name));
                parameterDict.Add("@" + prop.Name, val.ToString().ToUpper());
            }
        }

        commandStr = string.Format("Select {0} FROM {1} ", string.Join(", ", _DataMaps.Values.ToList()), _FileName);

        // Add WHERE command if Search parameters exist
        if (fieldList.Any())
        {
            // Example: Select CUSTNO, CFNAME, CLNAME, CSTREET, CCITY, CSTATE, CZIP, CPHONE, CALPHONE, CEMAIL, ORDDAT, BALDUE FROM CSCSTP WHERE UPPER
            commandStr = commandStr + string.Format("WHERE {0}", string.Join(",", fieldList));
        }

        break;
```

You will notice that the commandStr variable is slightly different based on the SQL command. After the switch statement on the commandType, the command object instance is created and the command parameters are added to the Parameters collection using the **iDB2Parameter** ADO.NET object. This is known as using Parameterized Queries, which provides the following benefits:

- Can improve performance
- Cleaner and more flexible code
- Prevents SQL Injection




```
        // Create a command
        var command = new iDB2Command(commandStr, connection);

        // Add Parameters to SQL Query
        foreach (var parm in parameterDict)
        {
            command.Parameters.Add(new iDB2Parameter(parm.Key, parm.Value));
        }

        return command;
    }
```

The _PopulateEntityFromReader() method reads the records from the returned reader and maps the data into the customer entity using the _DataMaps collections and adds each record to the entityCollection. Take a look at the _CreateCommand method. Here are some things we want to point out.

```
401  /// <summary>
402  /// Populates the entity from reader.
403  /// </summary>
404  /// <param name="reader">The reader.</param>
405  /// <param name="entityCollection">The enity collection.</param>
406  private void _PopulateEntityFromReader(iDB2DataReader reader, ObservableCollection<CustomerEntity> entityCollection)
407  {
408      // Read Rows
409      while (reader.Read())
410      {
411          // Populate Entity with Data
412          var customerRecord = new CustomerEntity();
413
414          var i = 0;
415          foreach (var map in _DataMaps)
416          {
417              // Get Property from Entity
418              var prop = customerRecord.GetType().GetProperties().FirstOrDefault(p => p.Name == map.Key);
419
420              if (prop == null) continue;
421
422              if (prop.PropertyType == typeof(int?))
423              {
424                  prop.SetValue(customerRecord, reader.GetInt32(i));
425              }
426              else if (prop.PropertyType == typeof(decimal?))
427              {
428                  prop.SetValue(customerRecord, reader.GetDecimal(i));
429              }
430              else
431              {
432                  prop.SetValue(customerRecord, reader.GetString(i).Trim());
433              }
434
435              i += 1;
436          }
437
438          entityCollection.Add(customerRecord);
439      }
440  }
441
```

Read Rows returned from ExecuteReader Method

Create Customer Entity instance customerRecord

Loop through _DataMaps collection and get corresponding property from the customer entity

Based on the property type, use the correct "Get" method to set the property value in the customer entity - ex: GetInt32, GetDecimal, GetString, etc.

Add customerRecord to entityCollection

9. The Insert, Update and Delete methods will need to do similar logic to the Select method.
    a. Connection object is created based on the IBMiConnectionString in the web.config and within a Using statement so that the connection object instance is disposed.
    b. The Open() method is called on the connection object instance to open the connection.
    c. The command object instance is created based on the SQL operation of Insert, Update and Delete. Any values are passed as parameters on the command object (Parameterized Queries).
    d. The Execute method is called on the command object Instance. In the case of the Select operation, the ExecuteReader() method was used which returns an **iDB2DataReader** object. The Insert, Update and Delete Operations do not need to return anything. So we can use the ExecuteNonQuery() method which executes the command and ignores any result set.

e. The open of the connection, creation of the command and execution of the command is all done within Try/Catch/Finally statements. The connection is closed in the finally statement which is always executed regardless if an exceptions happens or not.

Add the following highlighted code to the Insert, Update and Delete methods to execute the ExecuteNonQuery() method on the command object instance.

```
116 /// <summary>
117 /// Inserts the specified customer entity.
118 /// </summary>
119 /// <param name="customerEntity">The customer entity.</param>
120 /// <returns>ReturnArgs&lt;CustomerEntity&gt;.</returns>
121 public ReturnArgs<CustomerEntity> Insert(CustomerEntity customerEntity)
122 {
123     using (var connection = new iDB2Connection(ConfigurationManager.ConnectionStrings["IBMiConnectionString"].ConnectionString))
124     {
125         try
126         {
127             // Open Connection
128             connection.Open();
129
130             // Create Command
131             using (var command = _CreateCommand(_CommandTypes.Insert, custo
132             {
133                 // Execute Command - No rows returned
134                 command.ExecuteNonQuery();
135
136                 return new ReturnArgs<CustomerEntity>("OK", customerEntity);
137             }
138         }
139         catch (Exception ex)
140         {
141             _Logger.am_WriteLog(GetType().Name, AB_LogLevel.Error, System.Reflection.MethodBase.GetCurrentMethod().Name, ex.Message);
142             return new ReturnArgs<CustomerEntity>("ER", ex.Message);
143         }
144         finally
145         {
146             // Close the connection
147             if (connection.State != System.Data.ConnectionState.Closed)
148             {
149                 connection.Close();
150             }
151         }
152     }
153 }
```

This is the code added to the Insert Method. Do the same for the Update and Delete Methods.

10. Sometimes you need to convert data from 1 data type to another. In the case of the Cloud Services 24x7 database, the dates are stored as a *Numeric* 8 data type and the customer entity uses the *DateTime* data type for the date. In order to support this, you need to convert the data from 1 data type to another. You need to convert from *DateTime* to *Numeric* on an Insert and Update and convert from *Numeric* to *DateTime* on a Select.

The _DataConversionToDatabase() method is called from the _CreateCommand() method when creating the Insert and Update command Object. You need to convert the *DateTime* data to a *Numeric*. You can do this by converting the *DateTime* value to a *string* of numeric data that can then be inserted and updated. Note: The Cloud Services 24x7 database is storing the Order date in the format MMddyyyy. Add the highlighted code to the _DataConversionToDatabase() method.

```
392     private static string _DataConversionToDatabase(PropertyInfo prop, object val)
393     {
394         if (val == null)
395         {
396             // File does not allow null; set null data to blank or 0
397             return prop.PropertyType == typeof(string) ? string.Empty : "0"
398         }
399
400         if (prop.PropertyType == typeof(DateTime?))
401         {
402             // Convert from DateTime to mmddyyyy
403             var date = (DateTime)val;
404
405             var newDate = date.Date.ToString("MMddyyyy");
406
407             return newDate;
408         }
409
410         return val.ToString();
411
412     }
413
```

If the Property Type is DateTime, then convert the value to a string using the format "MMddyyyy".

11. For the Select method, you need to convert the data from *Numeric* to *DateTime*. You can do this in the _PopulateEntityFromReader() method which is called from the Select() Method. Add the highlighted code to the _PopulateEntityFromReader() method.

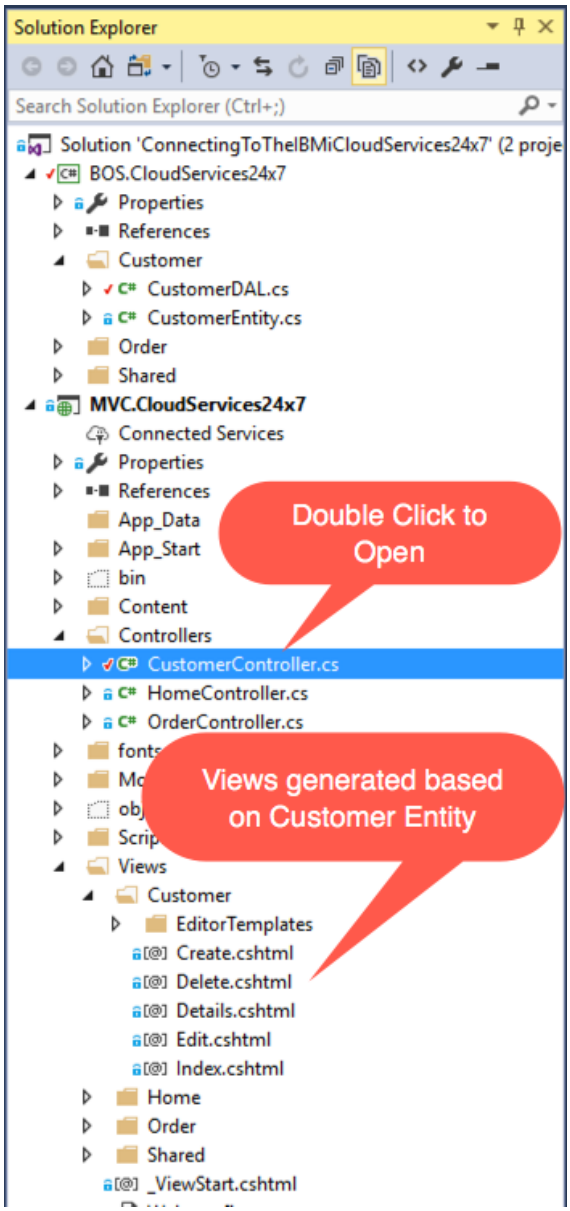
```
419     private void _PopulateEntityFromReader(iDB2DataReader reader, ObservableCollection<CustomerEntity> entityCollection)
420     {
421         // Read Rows
422         while (reader.Read())
423         {
424             // Populate Entity with Data
425             var customerRecord = new CustomerEntity();
426
427             var i = 0;
428             foreach (var map in _DataMaps)
429             {
430                 // Get Property from Entity
431                 var prop = customerRecord.GetType().GetProperties().FirstOrDefault(p => p.Name == map.Key);
432
433                 if (prop == null) continue;
434
435                 if (prop.PropertyType == typeof(int?))
436                 {
437                     prop.SetValue(customerRecord, reader.GetInt32(i));
438                 }
439                 else if (prop.PropertyType == typeof(decimal?))
440                 {
441                     prop.SetValue(customerRecord, reader.GetDecimal(i));
442                 }
443                 else if (prop.PropertyType == typeof(DateTime?))
444                 {
445                     // Convert integer to DateTime
446                     if (reader.GetInt32(i) != 0)
447                     {
448                         var strDate = reader.GetInt32(i).ToString().PadLeft(8, '0'); ;
449
450                         var dateConversion = DateTime.ParseExact(strDate,
451                             "MMddyyyy",
452                             CultureInfo.InvariantCulture,
453                             DateTimeStyles.None);
454                         prop.SetValue(customerRecord, dateConversion);
455                     }
456                 }
457                 else
458                 {
459                     prop.SetValue(customerRecord, reader.GetString(i).Trim());
460                 }
461
462                 i += 1;
463             }
464
465             entityCollection.Add(customerRecord);
466         }
467     }
```

Convert Integer in format "MMddyyyy" to DateTime

## 3.4. Review MVC Customer Controller

In the MVC project, the Controller along with the html views to display the data (Index, Create, Delete, Details and Edit pages) where auto generated based on the entity model using Visual Studio templates. With the MVC pattern, all requests come in through the controller. Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction.

1. Go ahead and open the CustomerController.cs in Visual Studio.



2. The Controller will be calling the Data Access Layer, which returns a model that will be passed to the view. You will not do any coding in the controller, but we want to highlight some important code.

```
14  public class CustomerController : Controller
15  {
16      private CustomerDAL _CustomerDAL = new CustomerDAL();
17
18      // GET: Customer
19      public ActionResult Index(CustomerEntity customerEntity)
20      {
21
22          return View(_CustomerDAL.Select(customerEntity).EntityCollection);
23      }
24
25      // GET: Customer/Details/5
26      public ActionResult Details(int? id)
27      {
28          if (id == null)
29          {
30              return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
31          }
32          var customerEntity = _CustomerDAL.Select(new CustomerEntity() { CustomerNumber = id }).EntityCollection.FirstOrDefault();
33          if (customerEntity == null)
34          {
35              return HttpNotFound();
36          }
37          return View(customerEntity);
38      }
39
```

Create Instance of Customer DAL

The Index method will output a listing of customers. The select method on the DAL is called which returns a collection of customers that is passed to the Index view.

The Details method will output a single customer. The select method on the DAL is called passing in the CustomerNumber which returns a single customer that is passed to the Detail view.

```
46      // POST: Customer/Create
47      // To protect from overposting attacks, please enable the specific properties you want to bind to, for
48      // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
49      [HttpPost]
50      [ValidateAntiForgeryToken]
51      public ActionResult Create(CustomerEntity customerEntity)
52      {
53          if (ModelState.IsValid)
54          {
55
56              var retArgs = _CustomerDAL.Insert(customerEntity);
57              if (retArgs.ReturnCode == "OK")
58              {
59                  return RedirectToAction("Index");
60              }
61
62              ViewBag.ErrorMessage = retArgs.ReturnMessage;
63          }
64
65
66          return View(customerEntity);
67      }
68
```

The Create Method will receive a Customer Entity, validate the data and then call the Insert method on the DAL

```
84      // POST: Customer/Edit/5
85      // To protect from overposting attacks, please enable the specific properties you want to bind to, for
86      // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
87      [HttpPost]
88      [ValidateAntiForgeryToken]
89      public ActionResult Edit(CustomerEntity customerEntity)
90      {
91          if (ModelState.IsValid)
92          {
93              var retArgs = _CustomerDAL.Update(customerEntity);
94              if (retArgs.ReturnCode == "OK")
95              {
96                  return RedirectToAction("Index");
97              }
98              ViewBag.ErrorMessage = retArgs.ReturnMessage;
99          }
100         return View(customerEntity);
101     }
```

The Edit Method will receive a Customer Entity, validate the data and then call the Update method on the DAL

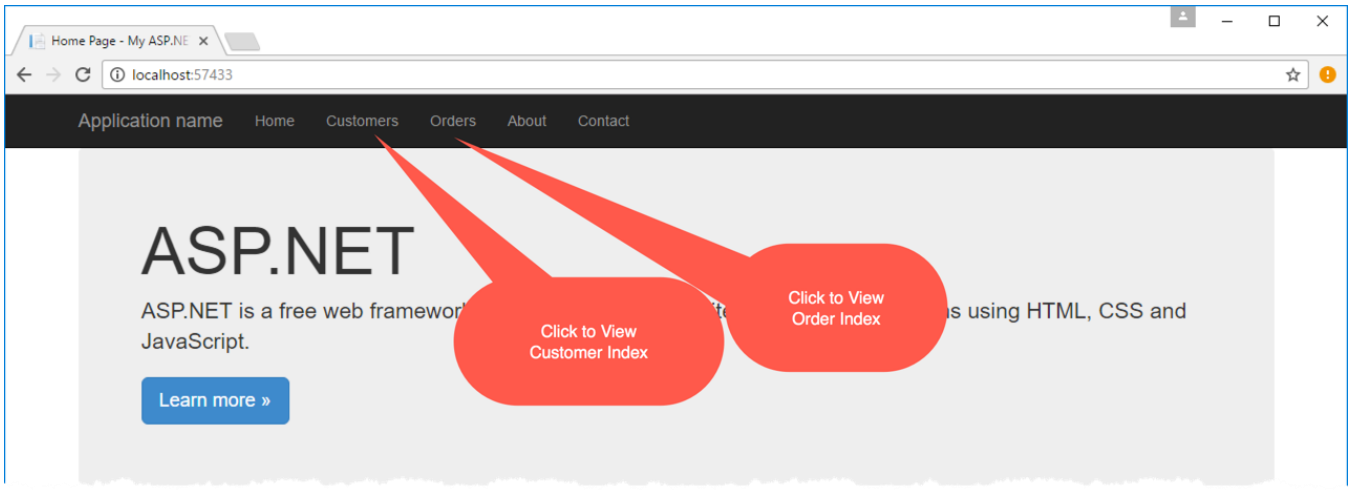
```
103     // GET: Customer/Delete/5
104     public ActionResult Delete(int? id)
105     {
106         if (id == null)
107         {
108             return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
109         }
110         var customerEntity = _CustomerDAL.Select(new CustomerEntity() { CustomerNumber = id }).EntityCollection.FirstOrDefault();
111         if (customerEntity == null)
112         {
113             return HttpNotFound();
114         }
115         return View(customerEntity);
116     }
117
118     // POST: Customer/Delete/5
119     [HttpPost, ActionName("Delete")]
120     [ValidateAntiForgeryToken]
121     public ActionResult DeleteConfirmed(int id)
122     {
123         var retArgs = _CustomerDAL.Delete(new CustomerEntity() { CustomerNumber = id });
124
125         return RedirectToAction("Index");
126
127     }
128
```

The Delete Method will receive a Customer Number. The Select method on the DAL is called passing in the Customer Number which returns a single customer that is passed to the Delete view.

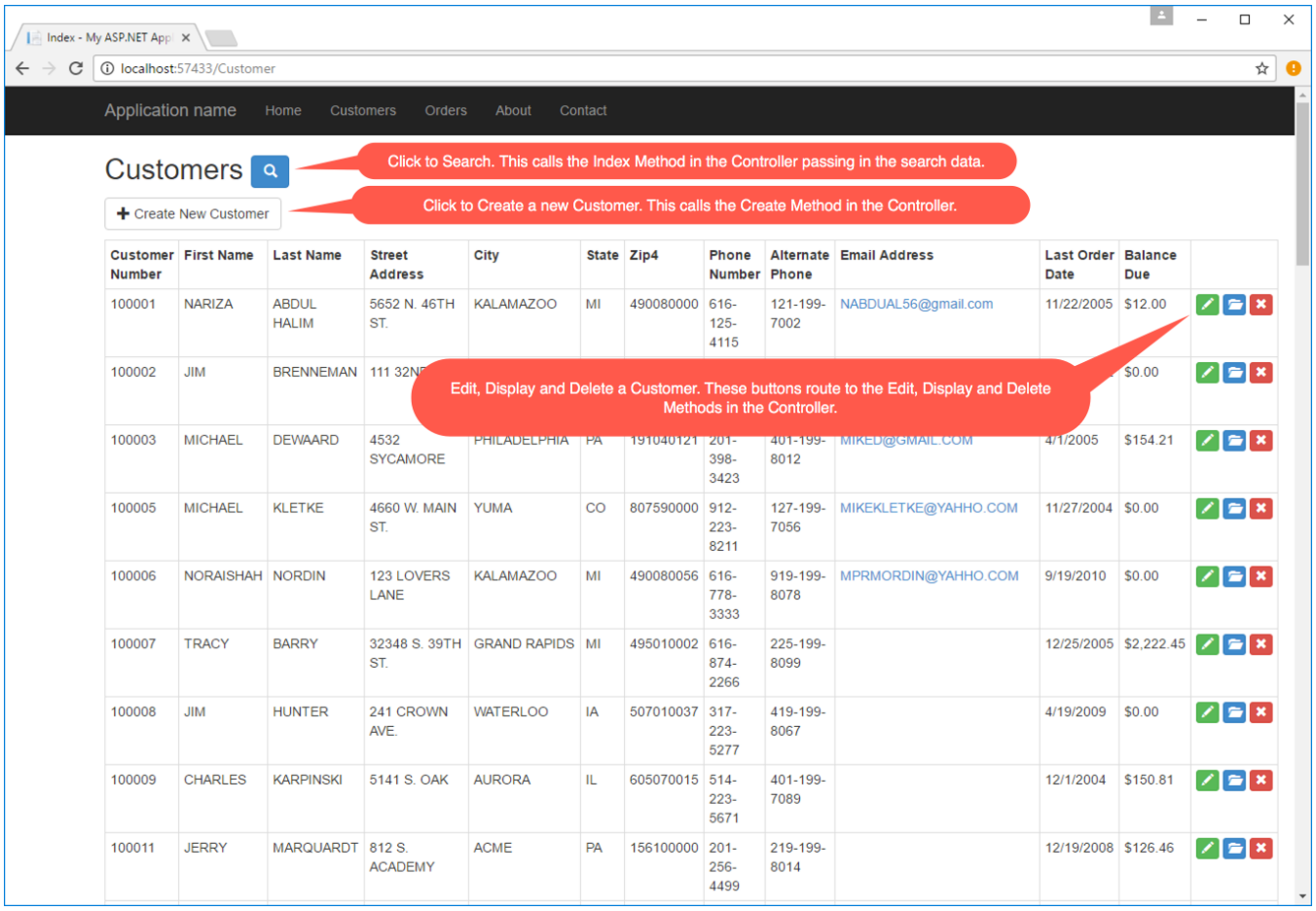
The DeleteConfirmed Method will receive a Customer Number. The Delete method on the DAL is called passing in the Customer Number.

## 3.5. Run the MVC UI

1. Run the MVC UI by clicking the Start Debug button in Visual Studio. Visual Studio will allow you to run the site using any installed browser. You can click the drop down on the button to change the browser to run.




2. Go ahead and Search, Create, Edit, Display and Delete Customers and Orders. The Order Maintenance code was provided with the code start.

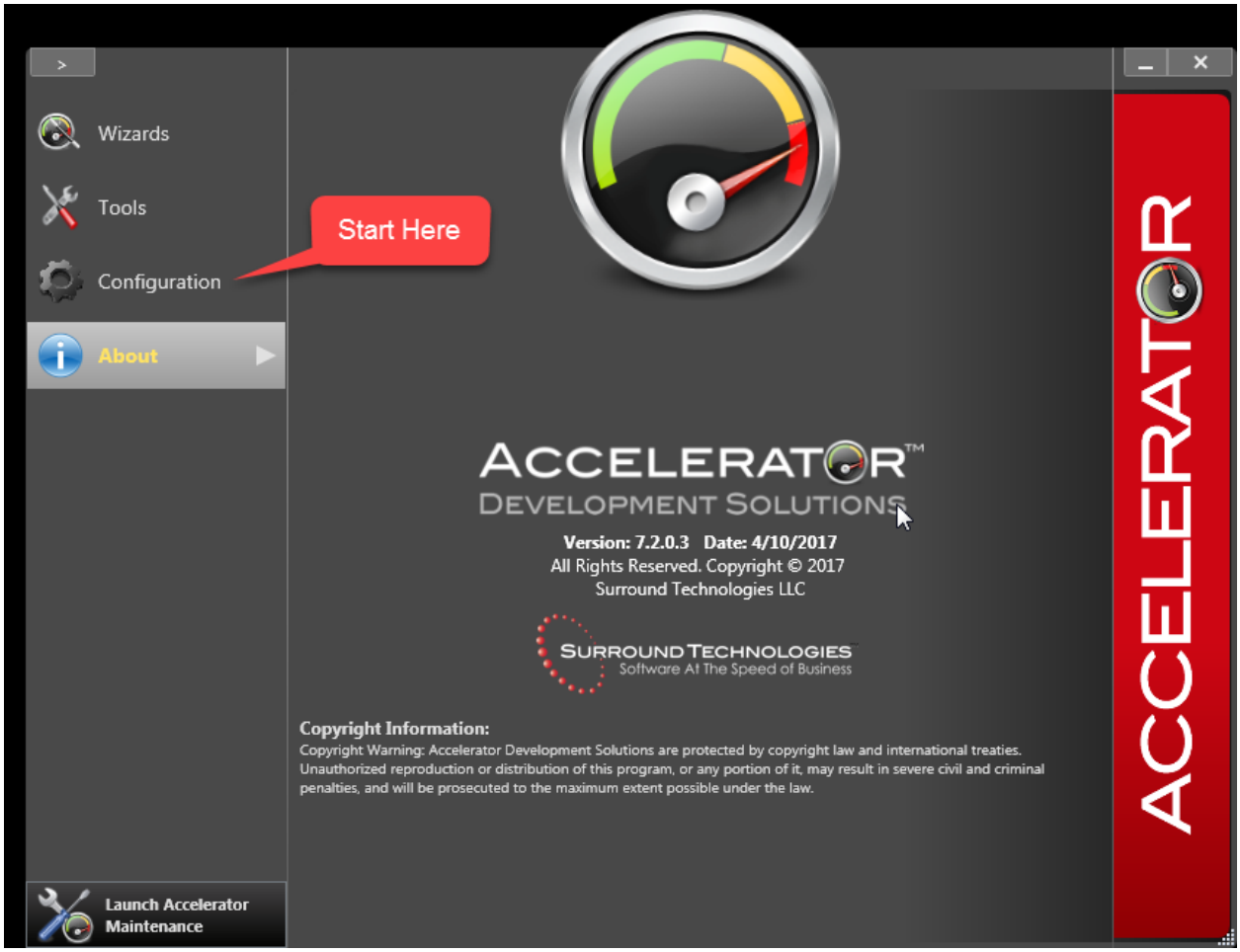


# 4. Building your Application with Accelerator

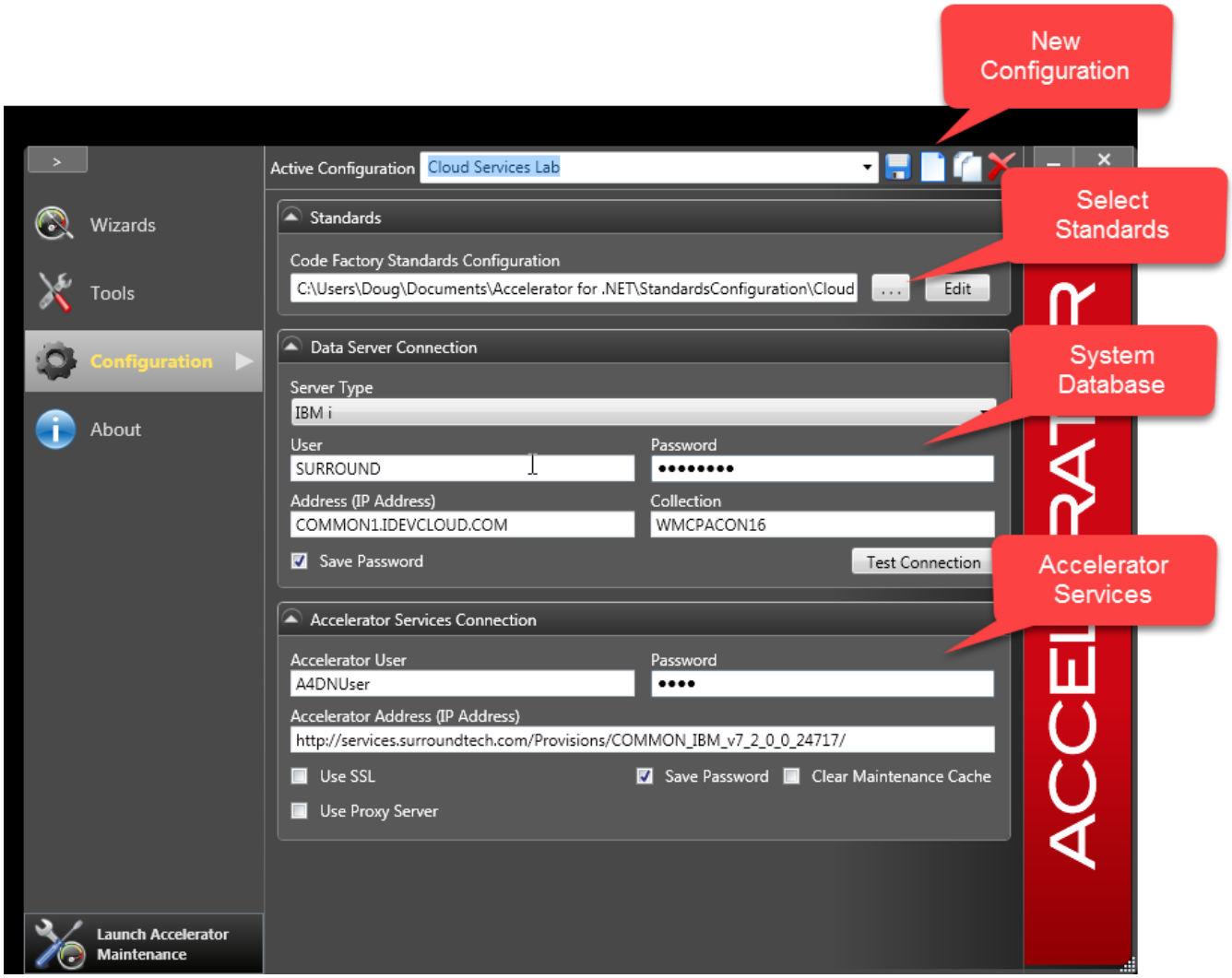
## 4.1. Run Launchpad

Find the Accelerator icon on your desktop or Start Menu and use it to run the Accelerator Launchpad.


The Launchpad has several sections on the left; we will start with the **Configuration** section.



## 4.2. Configuration




1. Click on the New Configuration icon (upper right) and in the **Active Configuration** textbox type in "Cloud Services Lab".
2. In the Standards section, click on the **...** button and browse to C:\Users\...\Documents\Accelerator for .NET\StandardsConfiguration\CloudServices24x7, where you can select the StandardsConfiguration.xml file for Cloud Services 24x7. This file provides presets for many of the settings in the System Wizard.
3. In the Data Server section, enter your system database information. Select IBM i as the database **Server Type**, and enter credentials, IP Address or hostname, and the Collection name for the Cloud Services 24x7 Legacy database. If you are attending a pre-arranged lab, your Surround Technologies instructor will provide you with the correct settings. You can use the Test Connection button to verify that Accelerator is able to communicate with the system database server.

4. In the Accelerator Services section, enter the credentials and URL for your Provisioned Environment. These are available in your Accelerator Trial welcome email, or will be provided by your Lab instructor.
5. Finally, click on the Save button at the top next to the Active Configuration textbox to save your configuration.

Now that your configuration is saved, make sure you have it selected in the Active Configuration box while starting the various Accelerator Wizards and Tools. They all make use of the Active Configuration to determine which system to work on. Also, you may want to check the "Save Password" check box for both the Data and Accelerator Server connections.

## 4.3. Run System Wizard

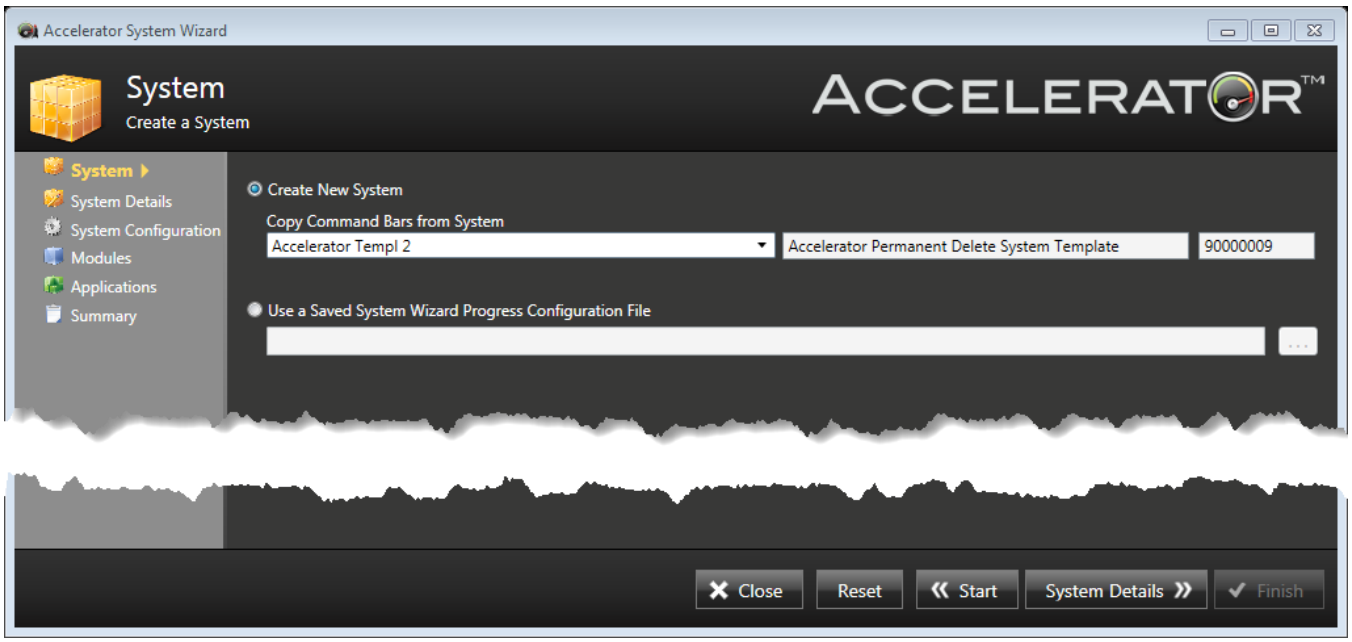
Select **Wizards** on the left, and then the System Wizard.


The System Wizard starts on a Welcome screen; click on the **System** button to begin.

### 4.3.1. Create New System

Choose the Create New System option, and leave Accelerator Template 2 selected. This sets up a default set of application Menus, Toolbars, and Commands. Click on **System Details** to proceed.

*At any point while using the System Wizard you can save your progress and exit; when you run the Wizard again you can choose Use a Saved System Wizard Progress Configuration file to continue where you left off.*

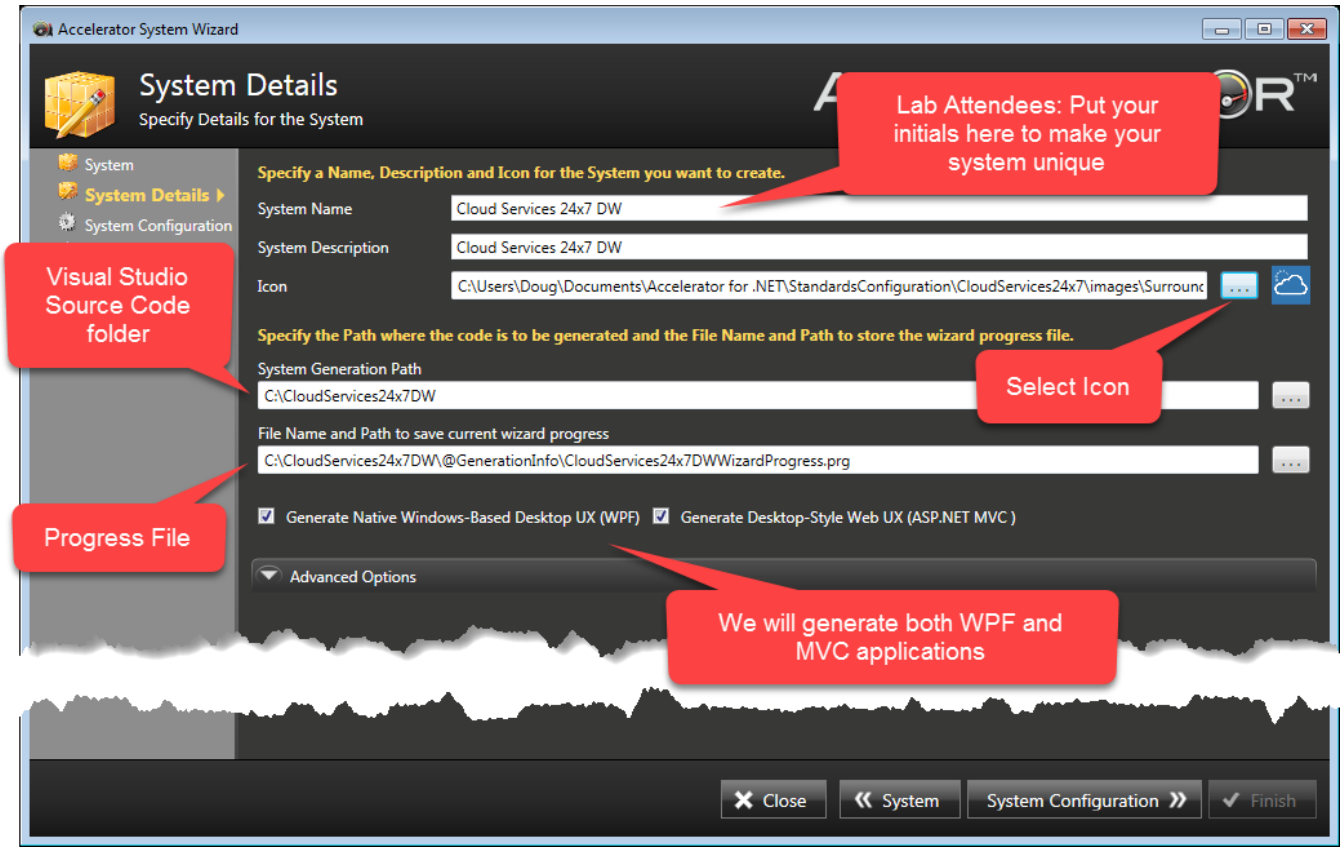



### 4.3.2. System Details

This page sets up the name of your application, where the C# source code will be placed, and the name and location of your Progress file in case you need to stop and continue the Wizard later.

1. Enter **Cloud Services 24x7** in the System Name box. If you are attending a Lab, please add your initials to the System Name to make it distinct.
2. Click on the **...** button next to the Icon field and browse to the Images folder in the Cloud Services 24x7 standards configuration. Select the SurroundEnterpriseSquare.ico icon file.

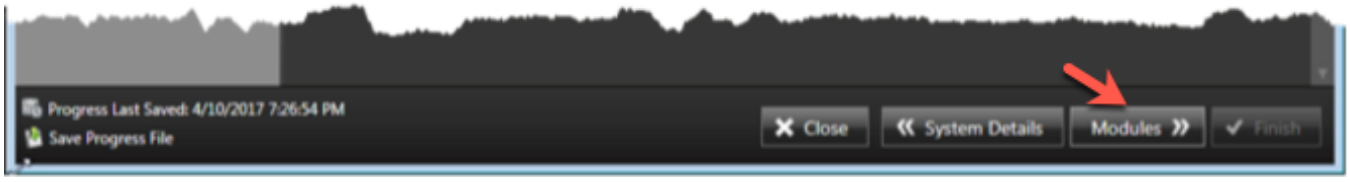
Click on **System Configuration** to proceed.




### 4.3.3. System Configuration

There are no settings to modify on the System Configuration screen for this lab. The information here has been pulled in from the Active Configuration that was set up earlier.

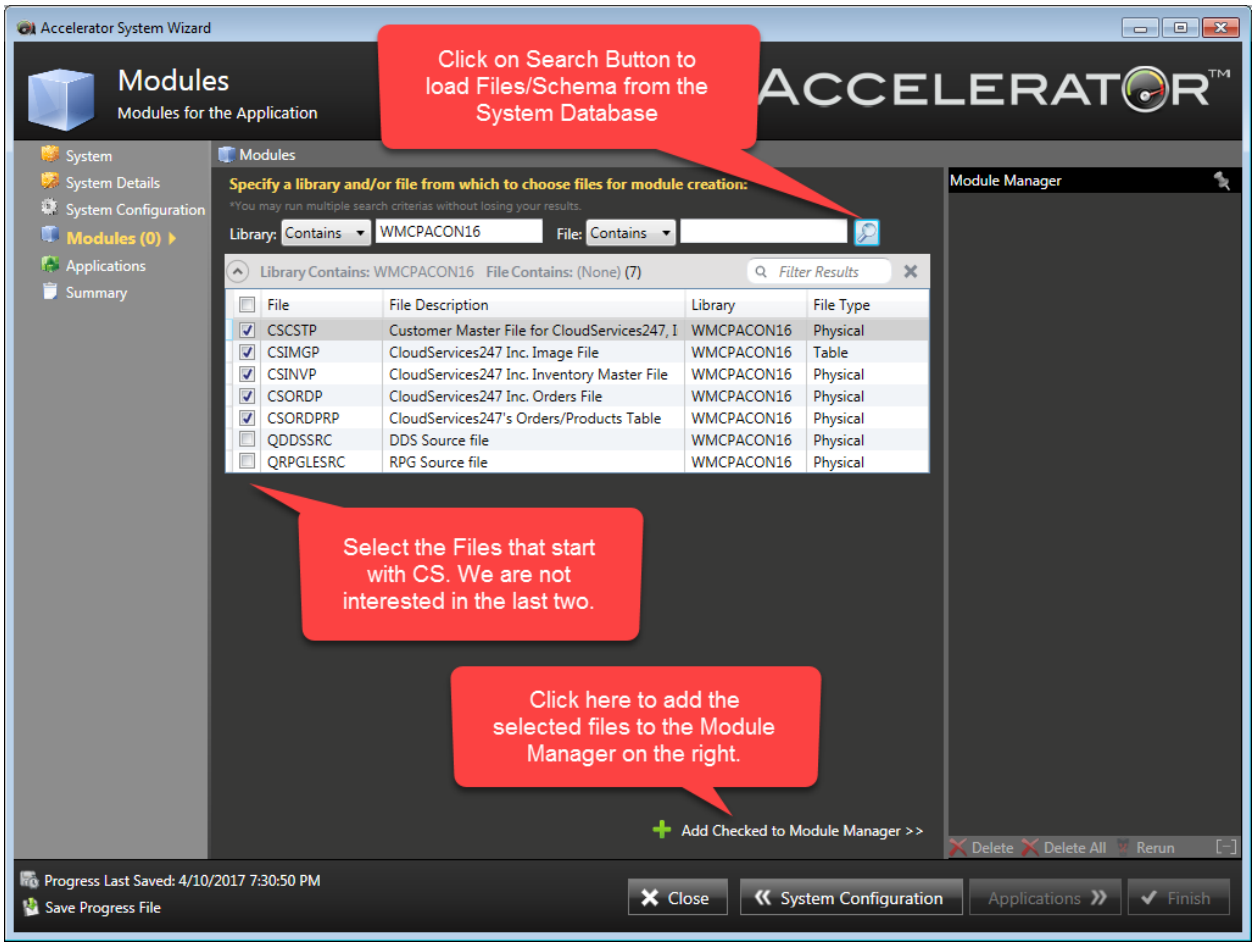
Click on **Modules** to proceed.



### 4.3.4. Modules

The Modules page lets you load Files and Schema information from the Cloud Services 24x7 system database, and choose the files to add to the Module Manager. Each Module will become an object in the application we are generating, with screens for searching, viewing, and editing the module data.

1. Click on the Search button (Magnifying Glass icon) to load a listing of all of the files in the database.
2. Select the five files whose names start with CS. These are the four files described in Section 2 plus a file that contains the product images.
3. Click on the Add Checked to Module Manager control at the bottom, which will create Modules for each of the five files.



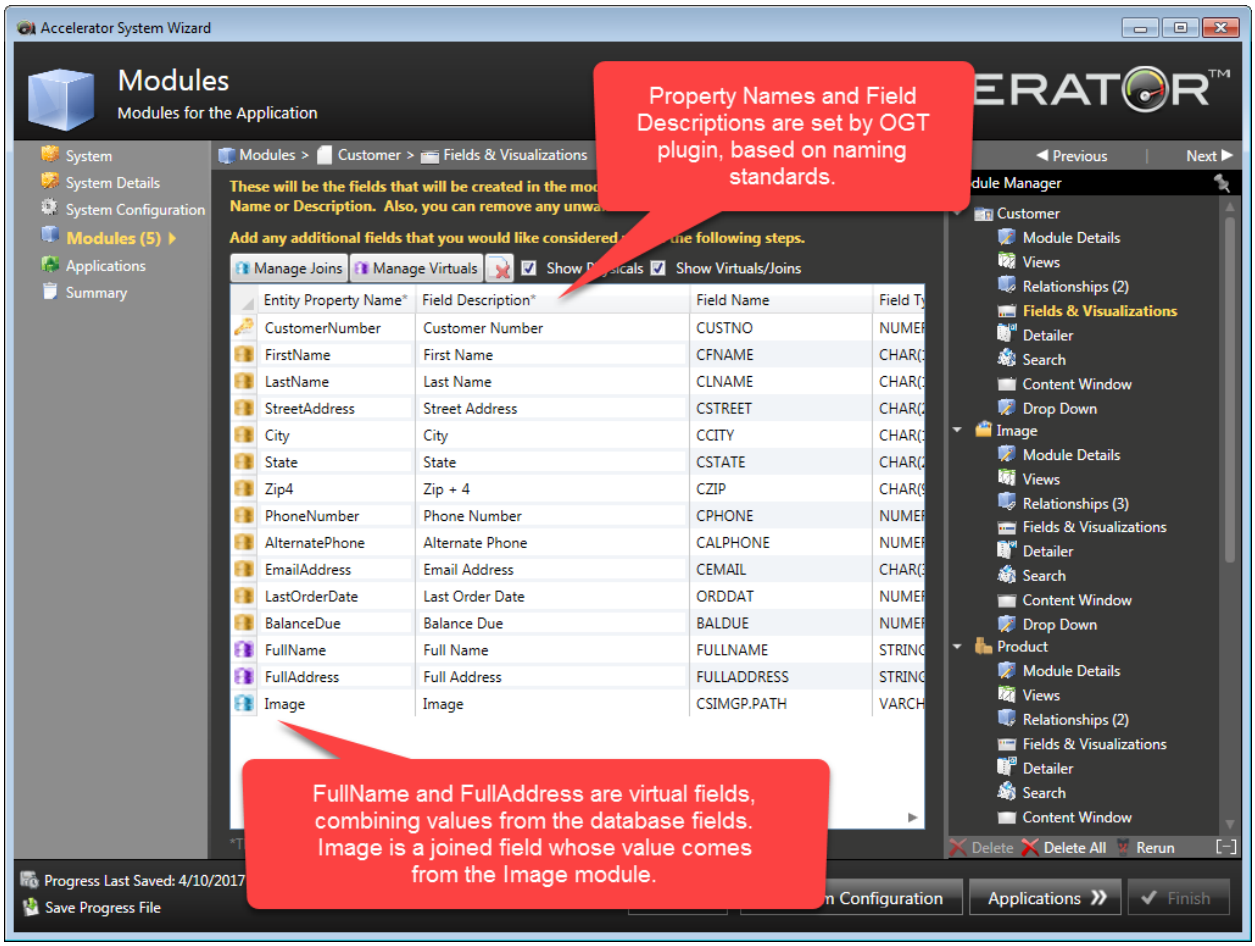
#### 4.3.4.1. Module Manager

Each file has become a Module, and in the Module Manager you can modify the Module's details. The OGT Plugin has done a lot of work for you here, setting naming conventions, choosing images, specifying Relationships, choosing fields for different displays, and so on. See Section 4.5 for a detailed list.

There's nothing that needs to be set here for the Lab, so feel free to click on **Applications** to proceed, and go on to the next section.

Here's one example of the customization that's been done: on Customer's Relationships, two joins have been added for the relationships to the Order module and the Image module. Accelerator knows that these are One-To-Many and One-To-One respectively, and will put that into the C# code and provide appropriate navigation, visualizations, and controls.

Another example worth noting is the Fields & Visualizations, which shows the Naming Standards, Virtual Fields, and Joined Fields that were set by the OGT Plugin. Everything here can be customized in the Wizard as well, and will be used during the generation of C# source code and setting up the Accelerator Framework Data for the application.

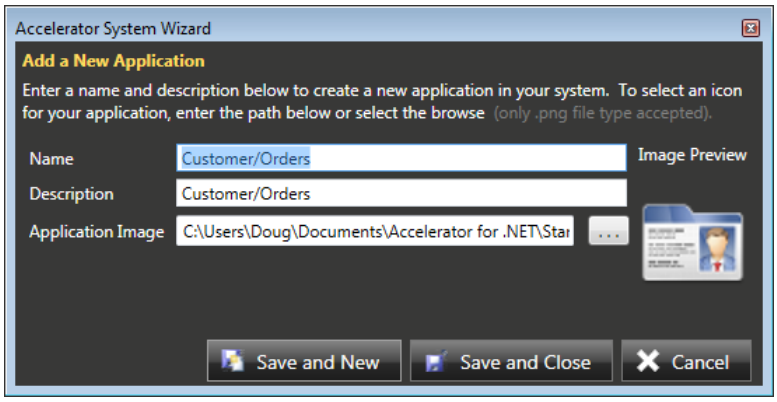



The other panels are:

- Module Details: Name, Description, Icon
- Views: Presets for Fields and Sort Order for datagrid Content Window display
- Detailer: Tabs and per-tab Fields to show on Record Detail display
- Search: Fields to include in Search form
- Content Window: Fields available for display in datagrid display, HTML custom markup for cells
- Drop Down: Customization for the Drop Down selection control for this module.

### 4.3.5. Applications

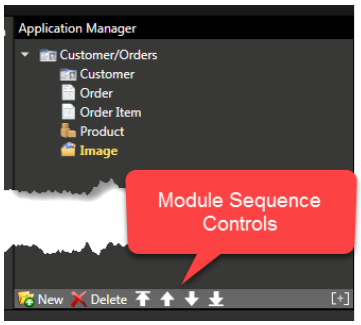
When you enter this page the Add a New Application dialog will appear. Enter **Customer/Orders** for the application name. The Description and Application Image will be automatically selected for you. Click on **Save and Close** to proceed.

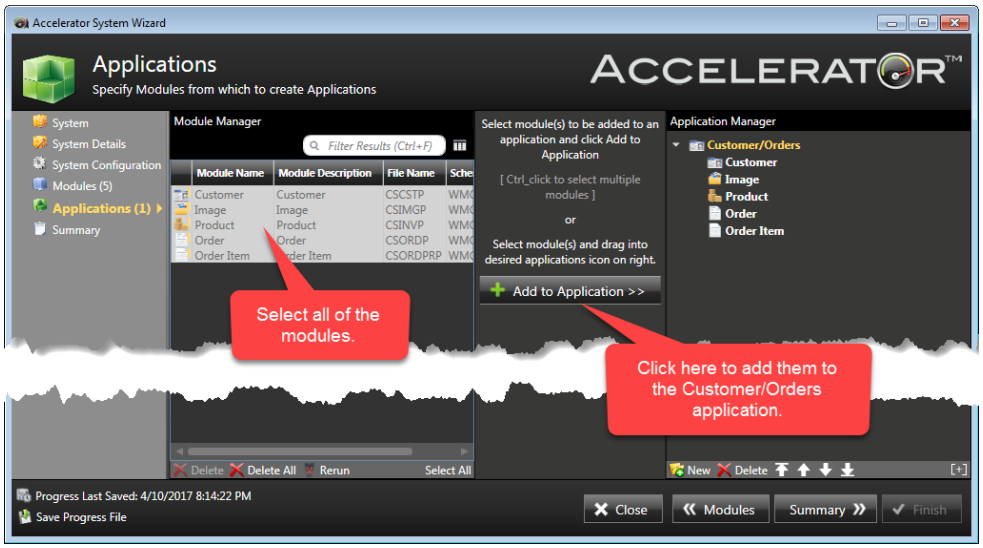



Applications are used to group Modules within the Cloud Services 24x7 Application we are generating. We will put them all into a single Customer/Orders application.

1. Select all of the modules on the left, and click Add to Application to move them to the right.
2. Use the Module Sequence controls to put the modules in this sequence: Customer, Order, Order Item, Product, and Image.

Click on **Summary** to proceed.



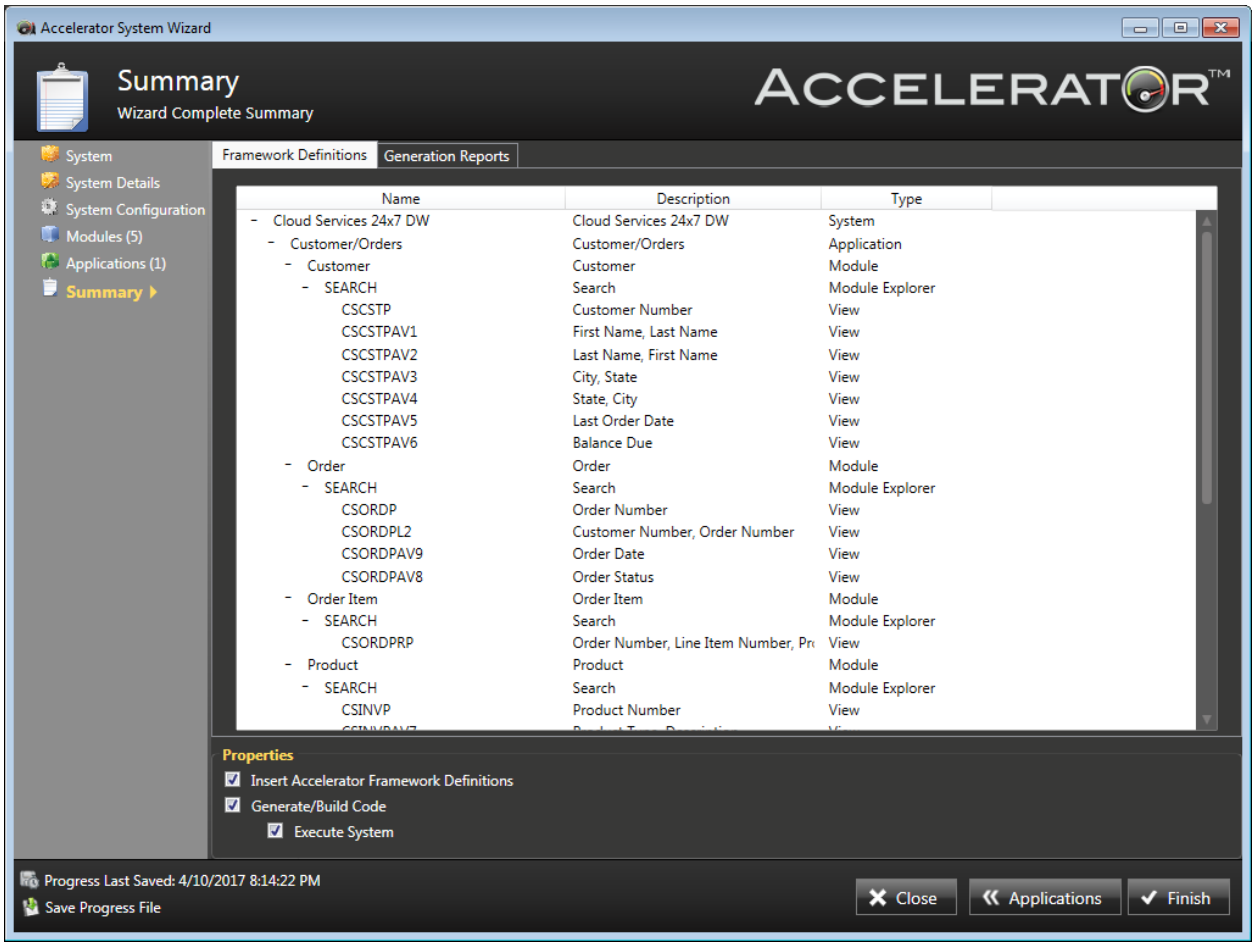




### 4.3.6. Summary

The Summary shows a list of everything that will be generated, and the three generation steps:

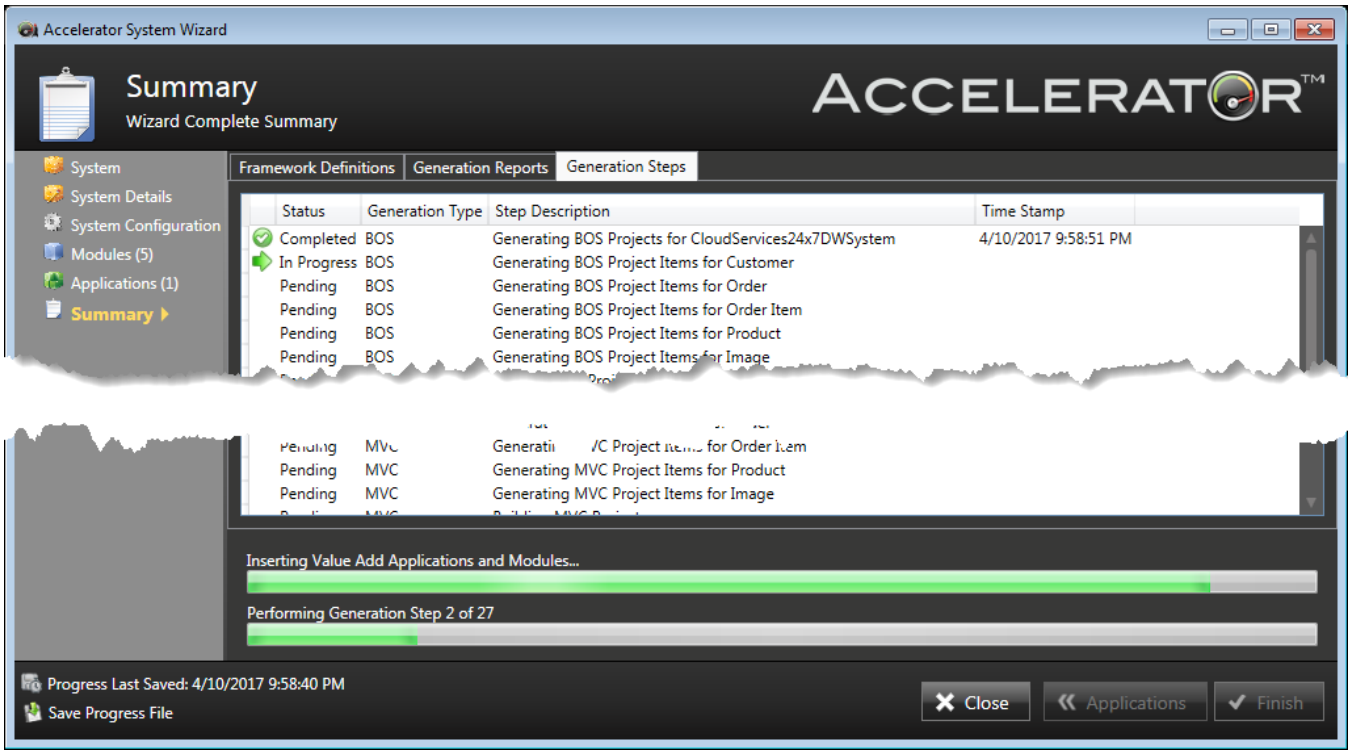
- Insert Accelerator Framework Definitions
- Generate / Build Code
- Execute System

After you click on **Finish**, the application will be generated and then both the WPF and MVC versions of the application will be executed. Click on **Finish** now.

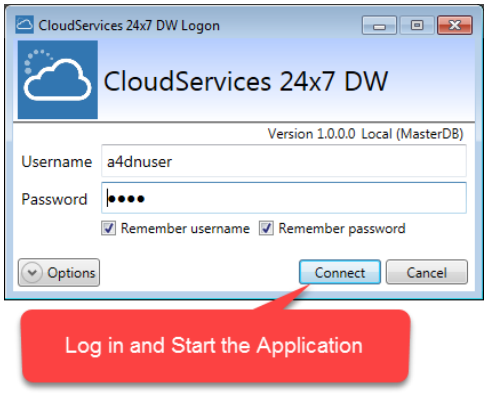


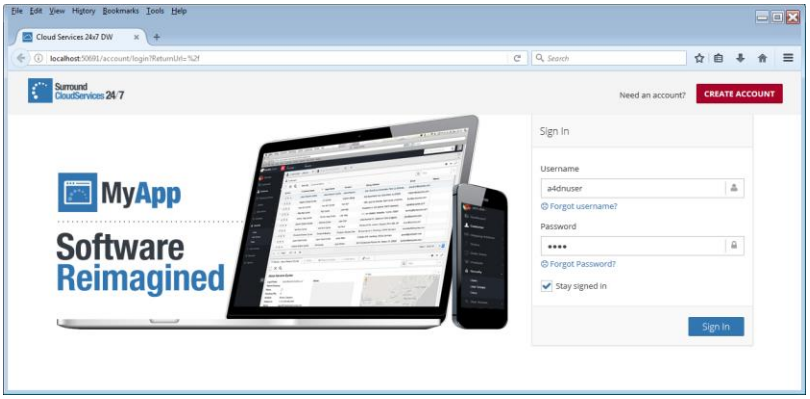

### 4.3.7. Finish

The Generation step will take a few minutes to complete, and progress is shown along the way.




Once Generation is complete, the applications will start. You can log in using the same credentials that were used for the Accelerator Framework Provisioned System. If you are attending a lab, or using a demo system, the default credentials are **username: a4dnuser** and **password: demo**.



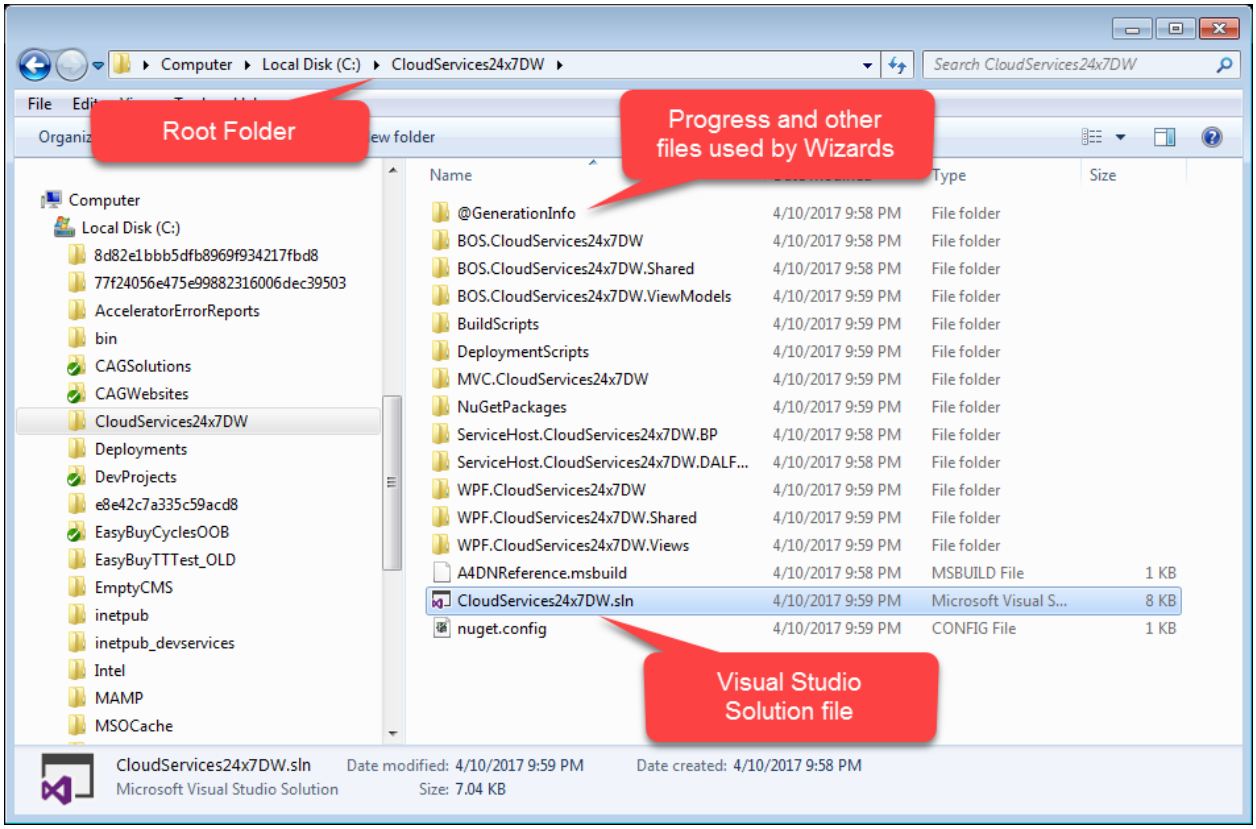




After logging in and exploring the applications (See Section 5) return to the System Wizard and click **Close** to finish with the Wizard. You'll be prompted to shut down Services that are used to help the application communicate with the Accelerator Framework and System Database. Close the applications as well.

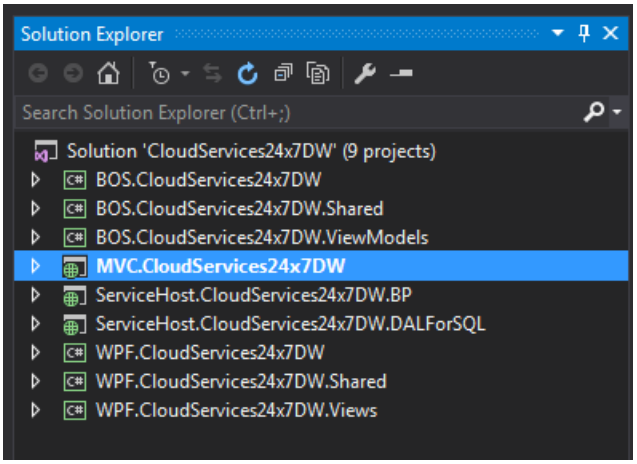
## 4.4. Visual Studio Solution

After closing the System Wizard, you can find the C# source code in C:\CloudServices24x7##, where ## are your initials. This path was set back on the System Details page.

Within this folder you'll find the Visual Studio solution file, CloudServices24x7##.sln, which you can use to load the application in Visual Studio. Also of note is the @GenerationInfo folder, which contains the Progress file and other Wizard files you can use to run the System Wizard again, or the Application or Module Wizards which can be used to add more Applications or Modules to the existing system.



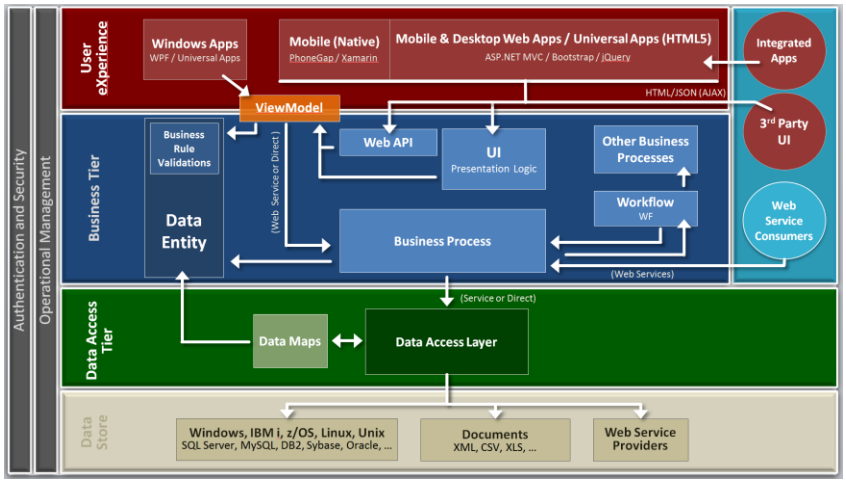
Within Visual Studio's Solution Explorer, you will find nine projects have been generated. Here is a quick summary of their functions:




- **BOS.CloudServices24x7**
  Business Logic and Data Access Logic for each of the Modules.
- **BOS.CloudServices24x7.Shared**
  Language resource files for text translations and other code shared across Modules.
- **BOS.CloudServices24x7.ViewModels**
  Per-module classes used to customize visual behaviors and access the data to display.
- **MVC.CloudServices24x7**
  The MVC Web app version of the Application.
- **ServiceHost.CloudServices24x7.BP**
  A Web Services host application for access to the Business Logic tier.
- **ServiceHost.CloudServices24x7.DALForSQL**
  A Web Services host application for access to the Data Access tier.
- **WPF.CloudServices24x7**
  The WPF desktop version of the Application
- **WPF.CloudServices24x7.Shared**
  Images and Theme resources used by the WPF application's Views
- **WPF.CloudServices24x7.Views**
  WPF XAML templates for each Module, which define the User Interface for Content Window, Search Form, Detail Display, and Drop Down control

These projects make up the Accelerator Framework Architecture.

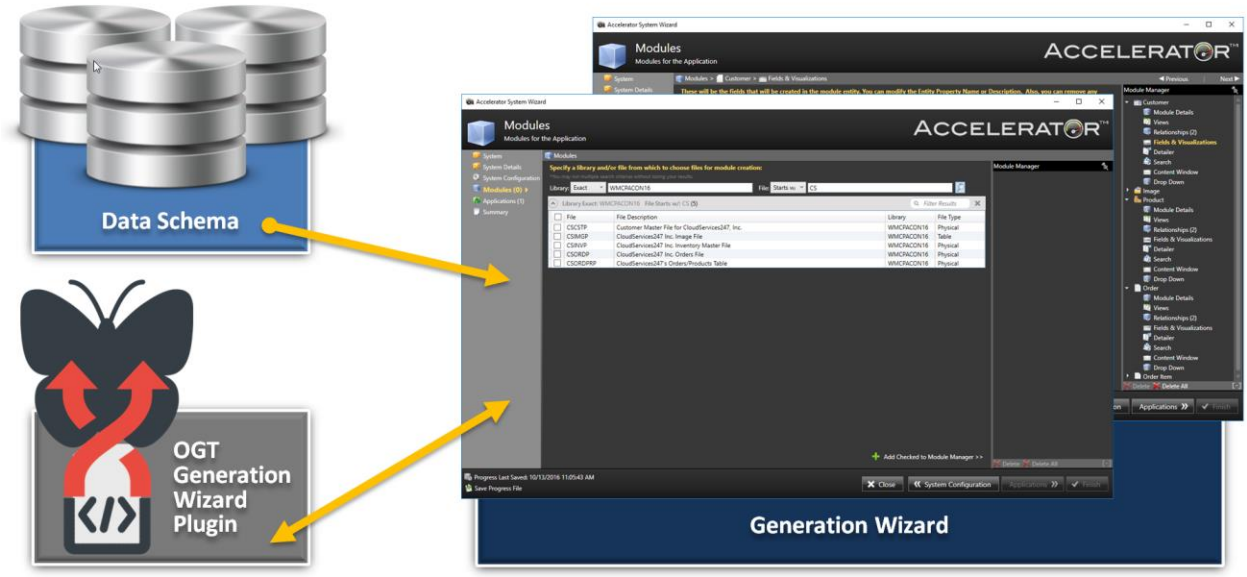
Learn more at [http://surroundtech.com/software-design-and-architecture](http://surroundtech.com/software-design-and-architecture)



## 4.5. What was in the OGT Plugin from the Generation Standards?

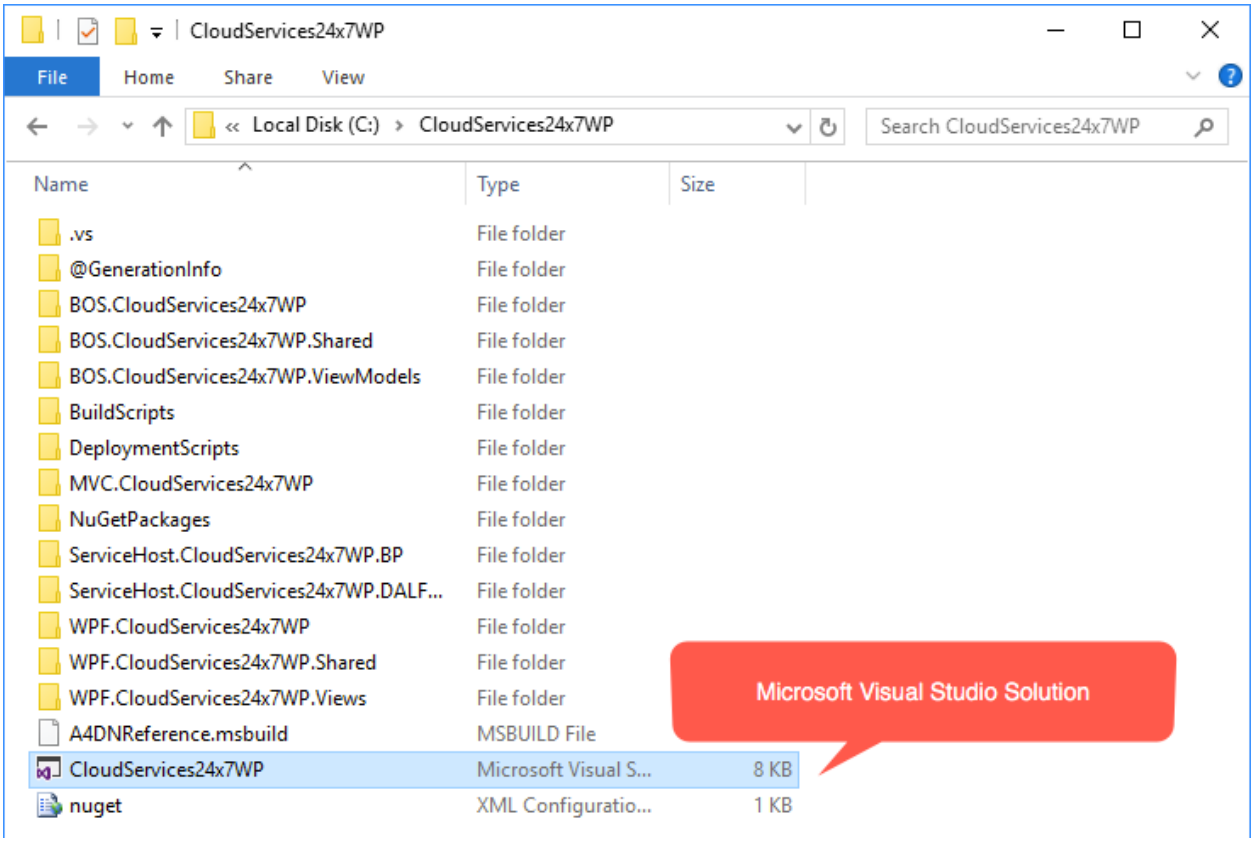
The Open Generation Technology Plugin allows you to implement C# code that is executed by the Accelerator System Wizard at various stages, and is generally used to set naming standards and identify file relationships that are not explicit in the database schema. For Cloud Services 24x7, the plugin includes the following:

- Removed keywords from file description to get a meaningful module name
- Changed all column/field descriptions to use title case
- Added module relationships
- Added joins to the image file based on module relationships
- Added virtual fields that concatenated any names or addresses
- Added a column rule for dates to use DateTime data type and date picker visualization
- Added column rules to set visualization, formatting and validation for Currency, Phone Numbers, Email, Checkboxes, Radio Buttons, etc.
- Set the checked state and sequence for all module options (Fields included in Searching, Data Grid Display, Detail Display, etc.)
- Added additional Accelerator Views (Accelerator Views allow an end user to select a preset sort and column arrangement)

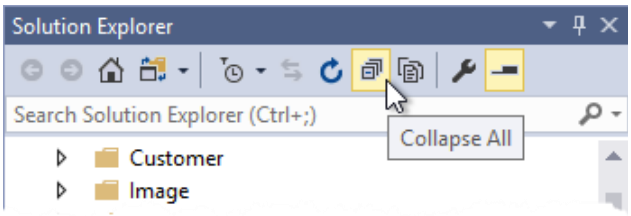


## 5. Highlights and Walkthrough of the Application

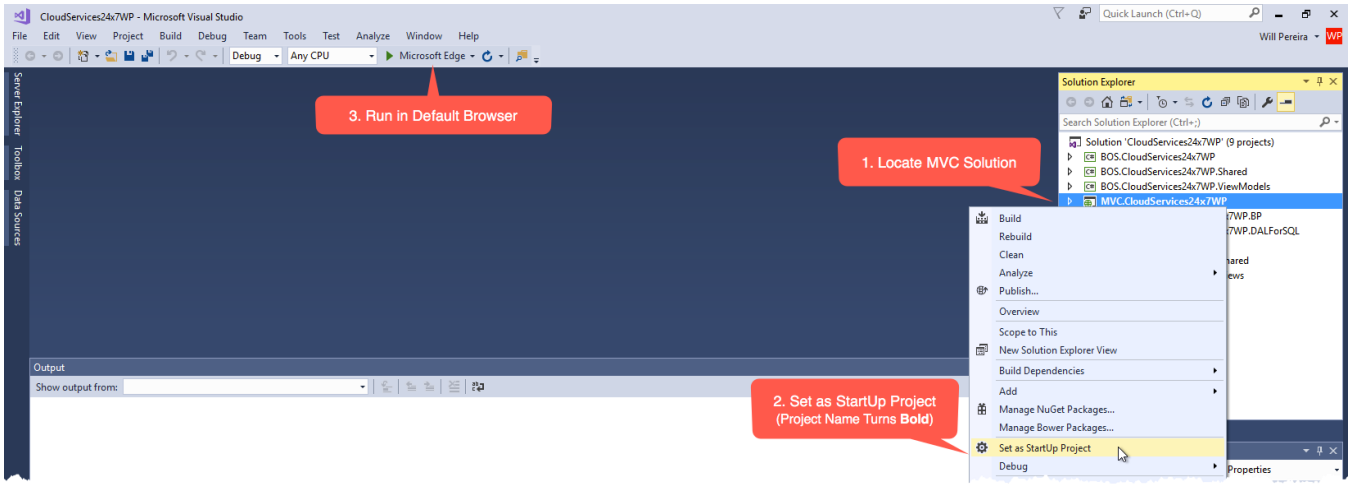
After generation, the systems you defined will automatically start up. If you need to reopen any of the systems you have generated, navigate to that systems root directory. For example, if you are using the CloudServices24x7 demo, navigate to C:\CloudServices24x7\. In this directory locate the CloudServices24x7.sln (Microsoft Visual Studio Solution) file and open the solution using Visual Studio 2017.



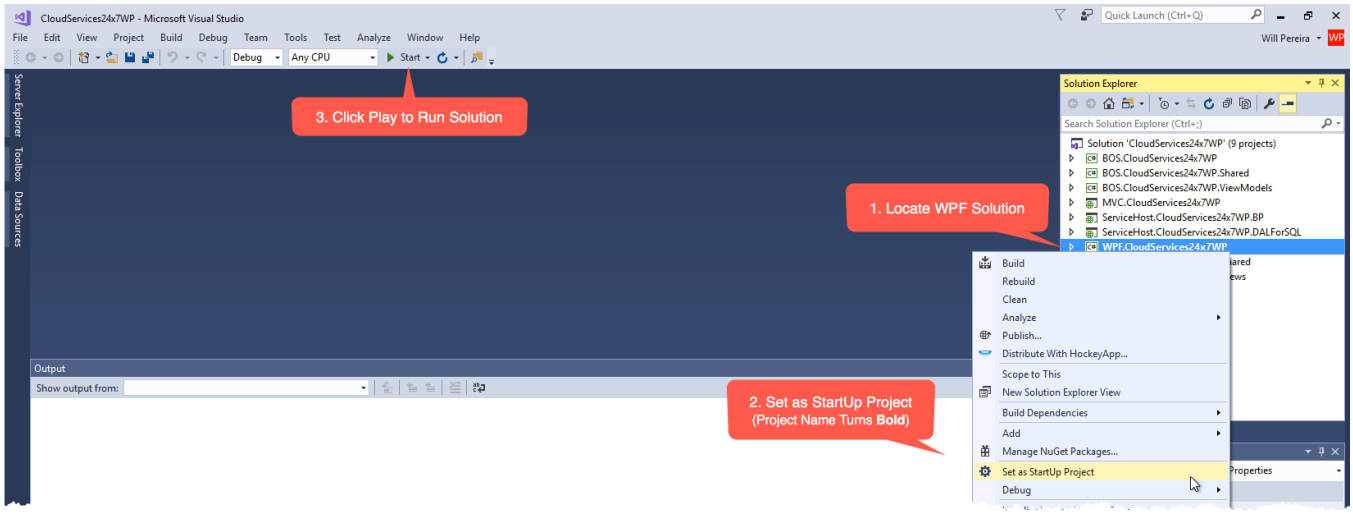

Depending on the system you wish to run, you will need to startup different solution files. Once the project has opened in Visual Studio, locate the Solution Explorer panel (usually on the top right of Visual Studio) and select Collapse All to make the solution explorer easier to navigate.



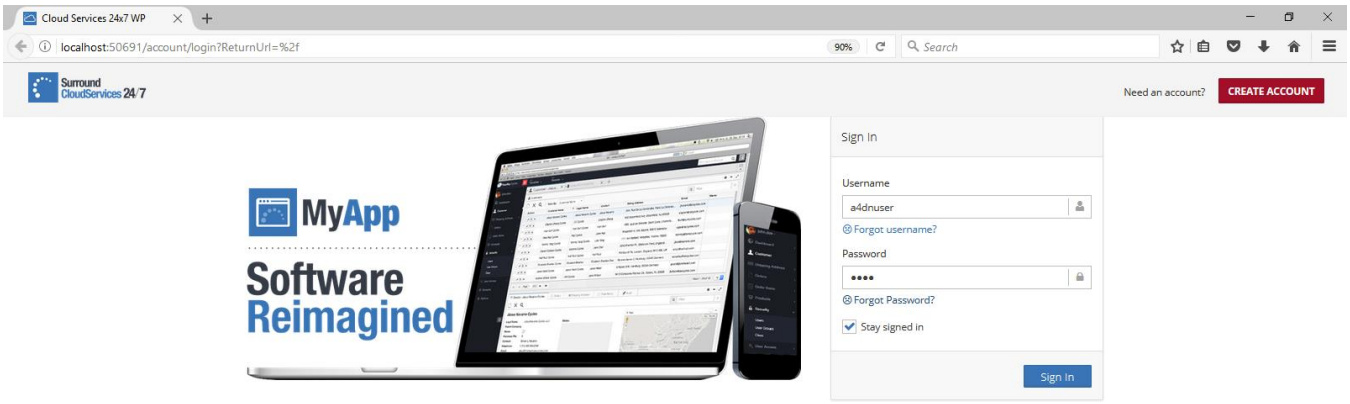
For the Responsive ASP.NET MVC web application, you will want to locate the MVC project in the Solution Explorer window. Click to highlight the project, then right-click and choose Set as StartUp Project. The project name will then appear in **bold** as a visual representation it is set as the default StartUp project. To start the solution, simply click the play icon on the top Visual Studio application toolbar to start the project in your default browser. If you wish to change the default browser, you can click the drop down arrow next to the browser name and select which browser you would prefer to use.



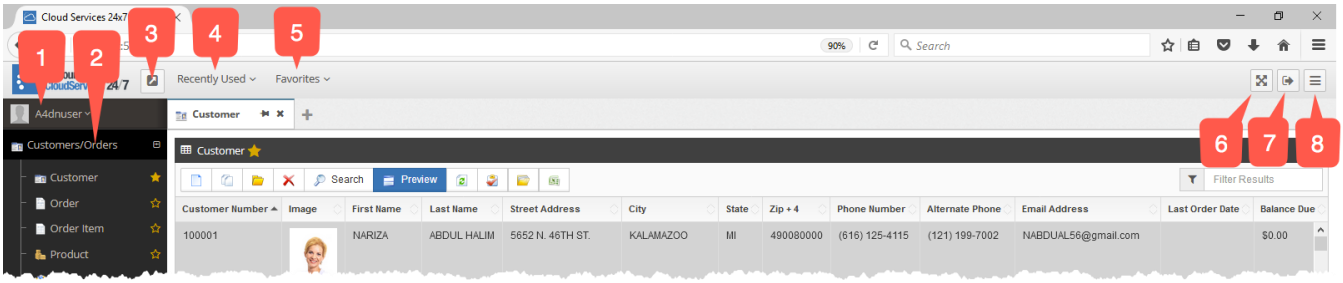

For the WPF application, you will want to locate the WPF project in the Solution Explorer window. Click to highlight the project, then right-click and choose Set as StartUp Project. The project name will then appear in **bold** as a visual representation it is set as the default StartUp project. To start the solution, simply click the play icon on the top Visual Studio application toolbar to start the project in Windows.



We will begin by reviewing the MVC Application first. Run the MVC application as instructed above. When the project loads in your browser. We will login using the credentials provided in your provision email, or if attending a lab by your speaker. If you're using an Accelerator demo, the default credentials are username: a4dnuser and password: demo. Login using the CloudServices24x7 credentials.

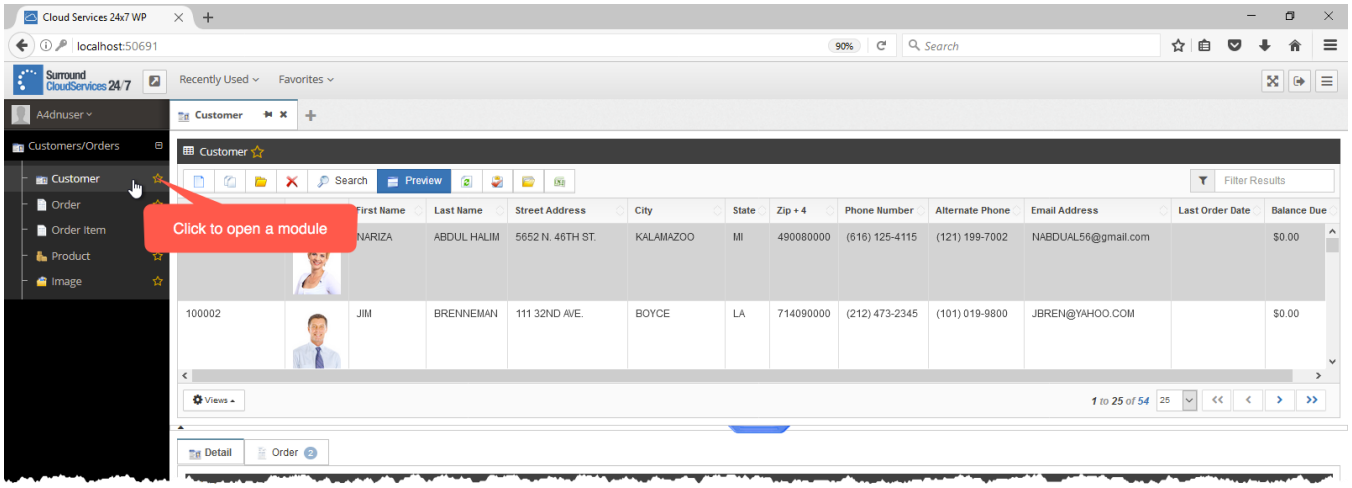



Once logged in, the first module in the navigator will automatically open in a tab if this is your first time logging in. Below you will find a list of features in the top most area of the responsive line-of-business application. These tools will help you navigate and explore the various areas and features of the system.

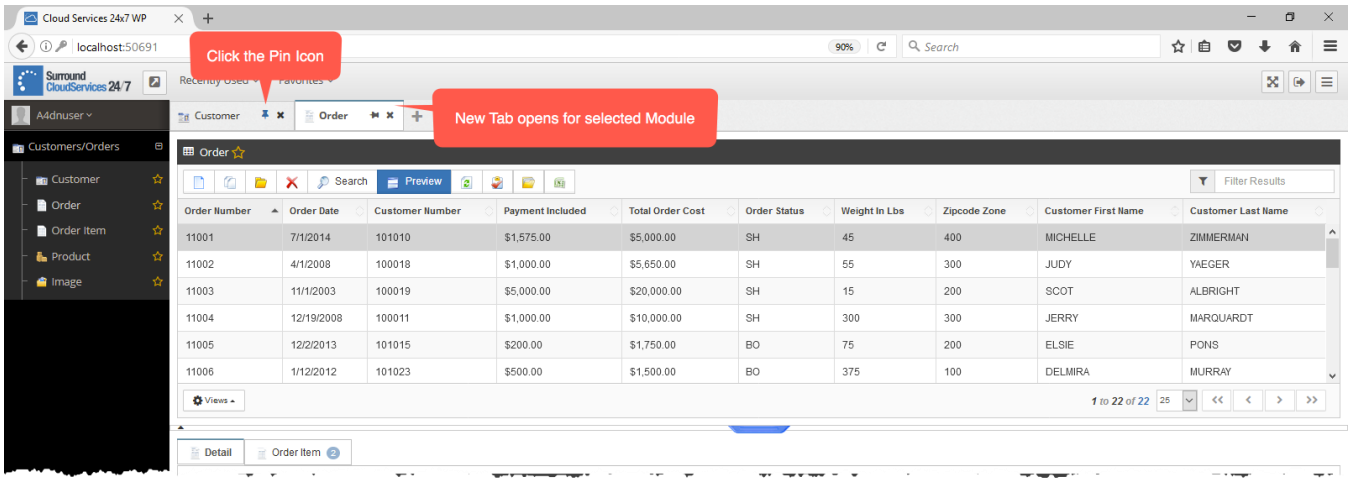



1. **User Account Settings**
   Update profile, email, change password & more
2. **Navigator**
   System Applications and Modules
3. **History, Folders and Hotlists**
   Quick access to apps and modules
4. **Recently Used Modules**
   Easy access to recently visited modules
5. **Favorites**
   Quickly access frequently used tools
6. **Full Screen Mode**
   Run web application in full screen
7. **Sign Out**
   Securely sign out of web application
8. **View/Hide Navigator**
   Navigator will hide off to the left of the screen until needed.

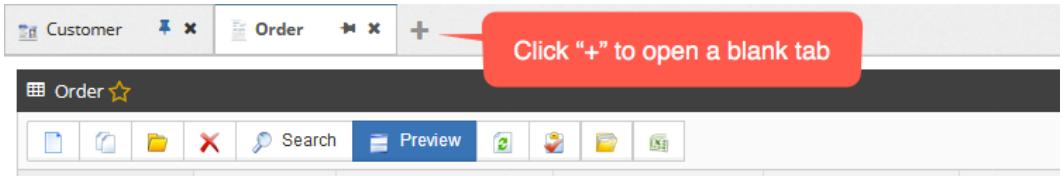
Navigating through the generated applications and modules is simple, and the Accelerator allows multiple modules to be opened and worked with simultaneously. Select a module from the navigator and it will open in the active tab.



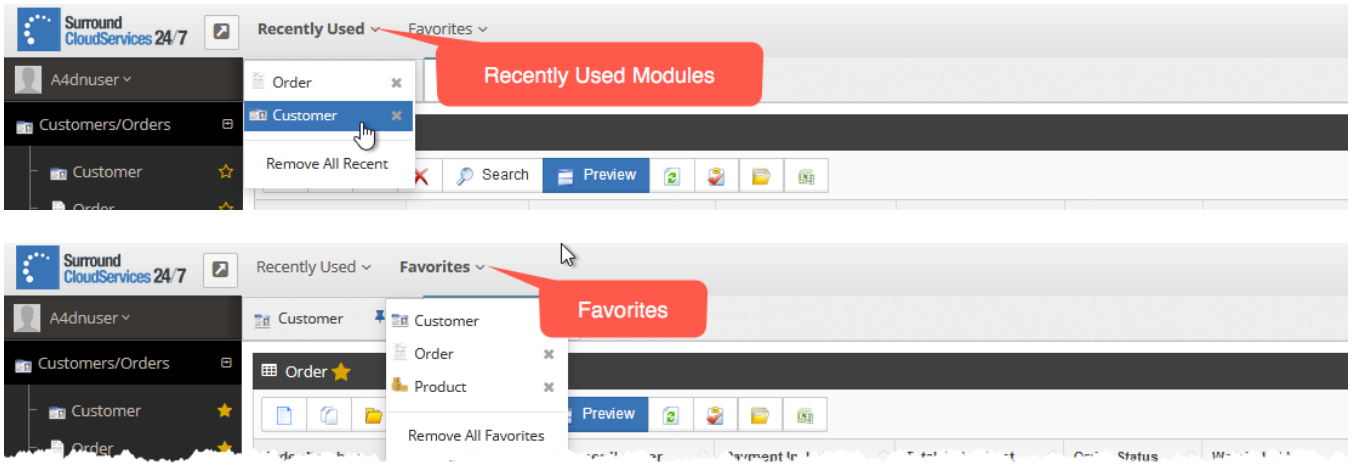

If you wish to keep the active tab open, and open subsequent modules in new tabs, click the Pin Icon in the tab that you wish to keep open. Try pinning a tab yourself, and then opening a module from the navigator. To close a tab, simple click the close icon ( “X” ) on the tab you wish to close.



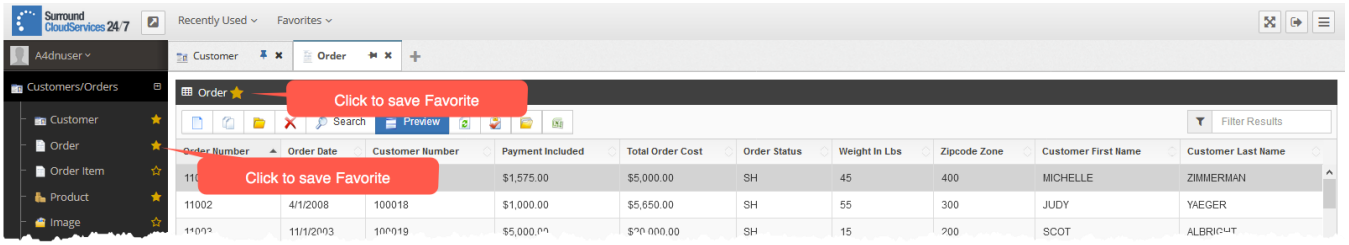
You can also click the " + " next to the tabs to open a new tab at any time. From the blank tab, you will have access to recently visited modules, favorites or you can choose a module from the navigator to open.



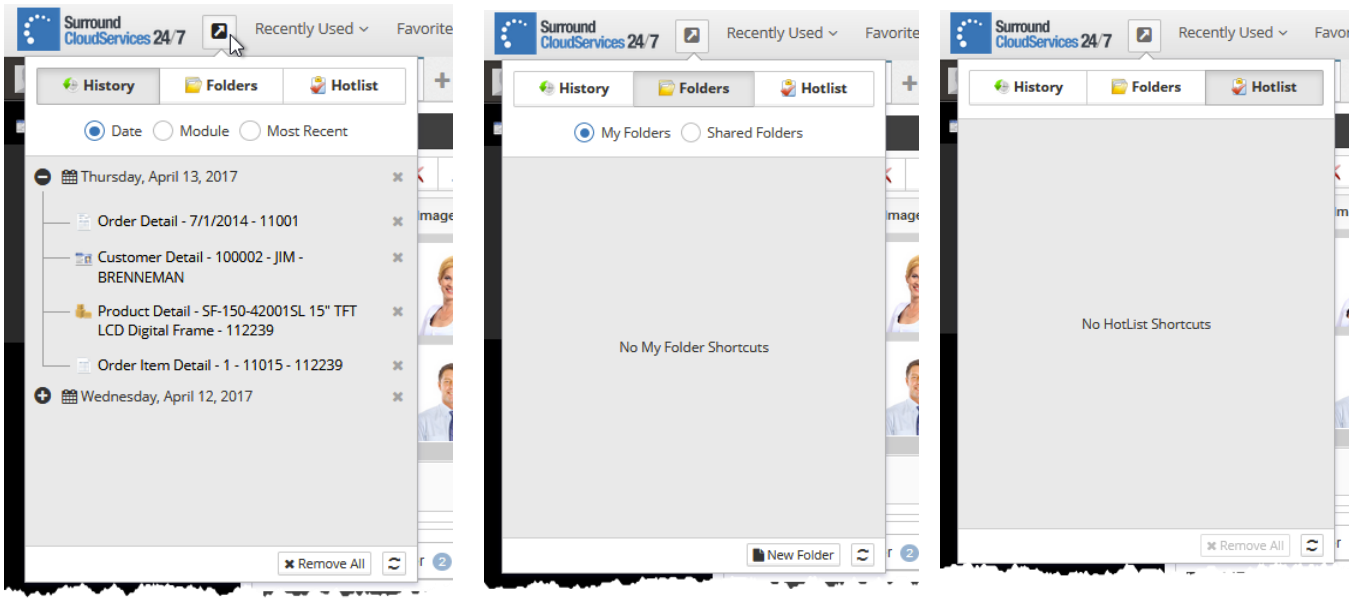

For existing users, or those with more time spent using the system, you may also choose to save your favorite modules to the Favorites list and open modules from there. You may also easily recall recently used modules by accessing them from the Recently Used drop down. Recently used lists the 10 most recent modules that have been used.



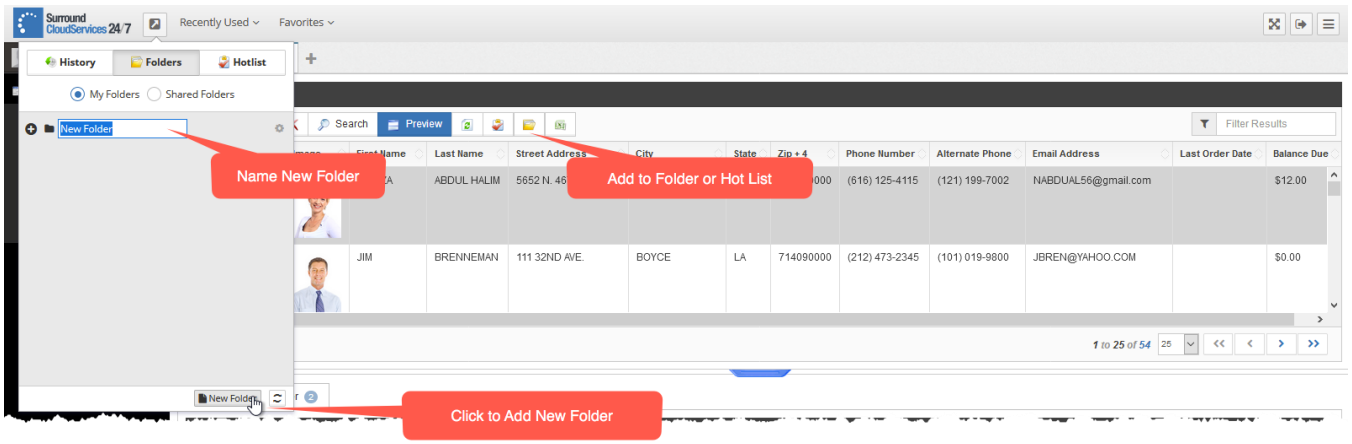

To add a module to the favorites list, click the favorite icon ★ next to the module name in the navigator or from the module toolbar .



If you need to reopen a recently used detail record, you can easily click the Shortcuts button in the application level toolbar to reveal History, Folders and Hotlists. History is capable of tracking activity throughout the system until it is cleared. It records detail records that have been opened. Should you need to return to something you were working on last week, the history can be easily used to find what you were working on by date, module or recent activity.




Folders and Hot Lists are a great way to stay organized. You can store any records you'd like and easily retrieve them at a later time. To create a new folder, simple click the New Folder button on the bottom right of the shortcuts window and you'll need to give the new folder a name.



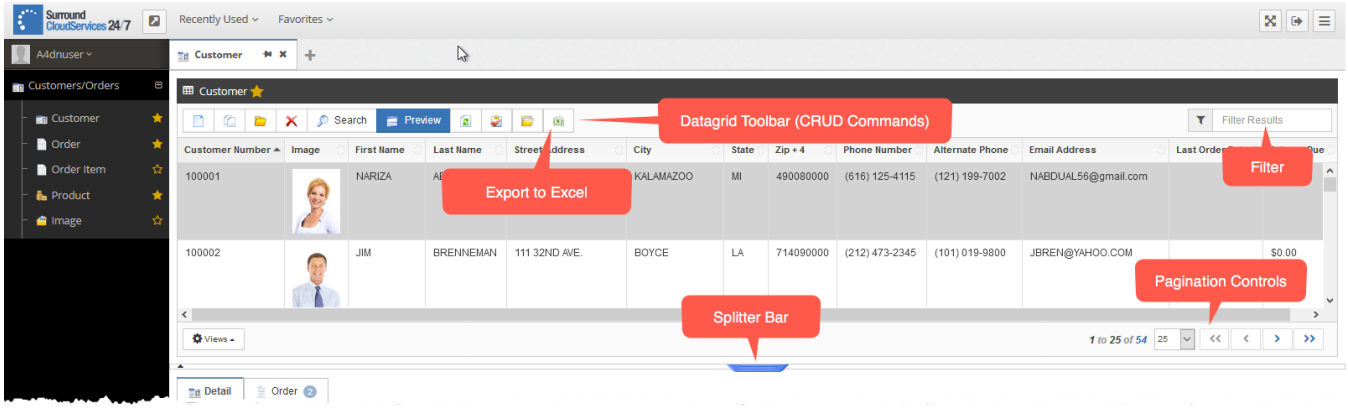

To add a record to a folder or hotlist, simply click the Add to Folder or Add to Hotlist icons in the datagrid toolbar when you have the desired record selected. If you're inside of a detail record already, you can

simply use the toolbar icons to add the active record to a folder or hotlist. The folder and hotlist features are available in the datagrid toolbar security permitting.

The System is designed to allow users to control how they navigate the application, so that they can build the system to meet their needs. By using pinned tabs, which open each time the system loads, history, favorites, folders, and hotlists, users can easily retrieve data at any level of the system making tasks more efficient and more productive.

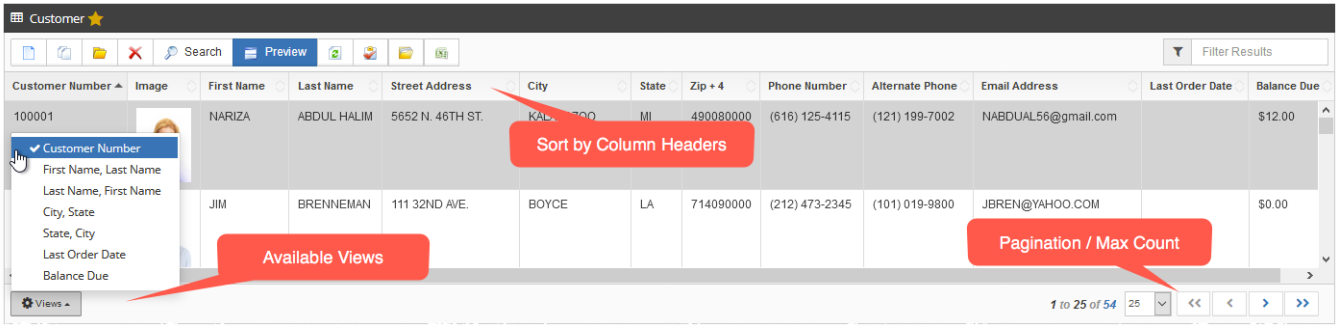
Now that we have covered the fundamental areas of the systems' navigation, let's take a look at the system data, its presentation, and accessibility throughout the various areas of the system.

The content area is the primary work space for each module. Here you will find the datagrid located on the top half of the content area. The datagrid lists the data available for the active module, and provides the ability to scroll and page through the results, as well as sort or filter to quickly find the information you are looking for. The datagrid tool bar provides access to typical CRUD commands (add, change, delete) as well as additional features like Add to Folder, Export to Excel and more depending on the needs of the selected module.

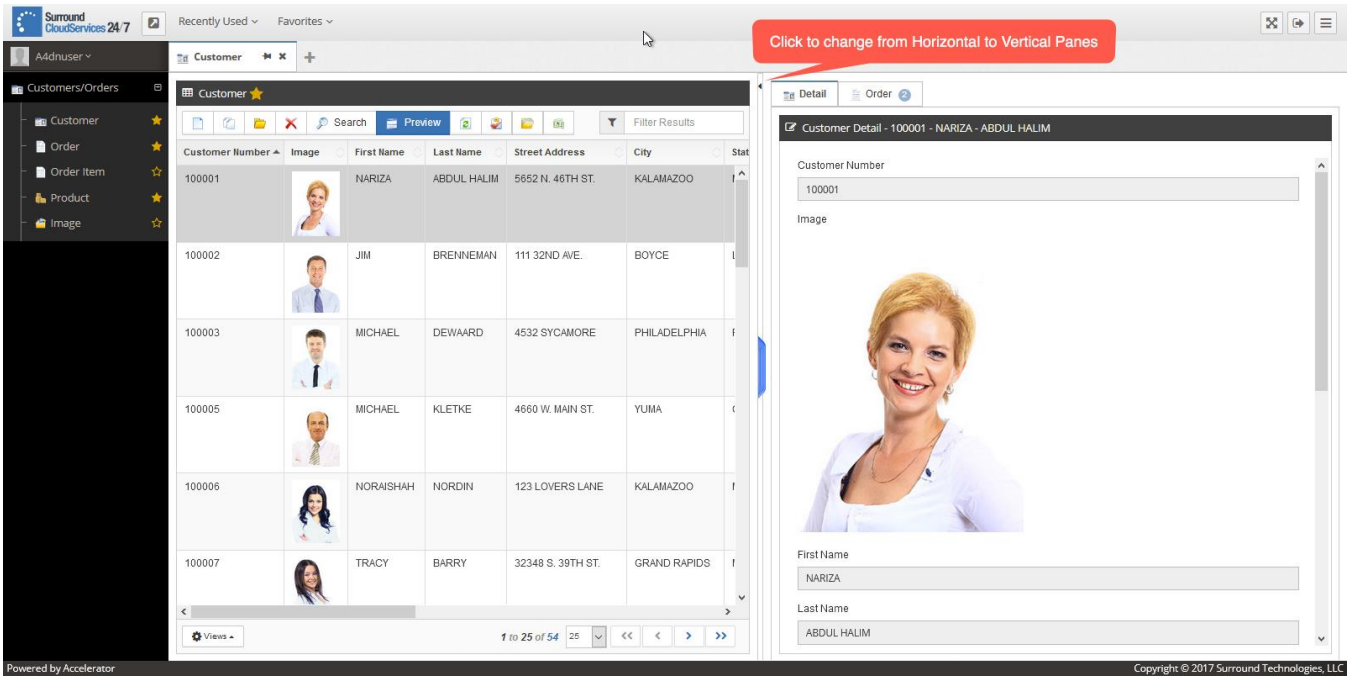



The data results in the module are displayed in a default view. The view controls the order in which the results will be displayed on load. The default view can be changed by selecting the View drop down on the bottom left of the datagrid and selecting an available view for the active module. Sorting the datagrid, by clicking on the column headers, can also modify views. By default, the datagrid will only show a limited number of results. You can change this at any time by changing the max record count in the pagination

controls on the bottom right of the data grid.



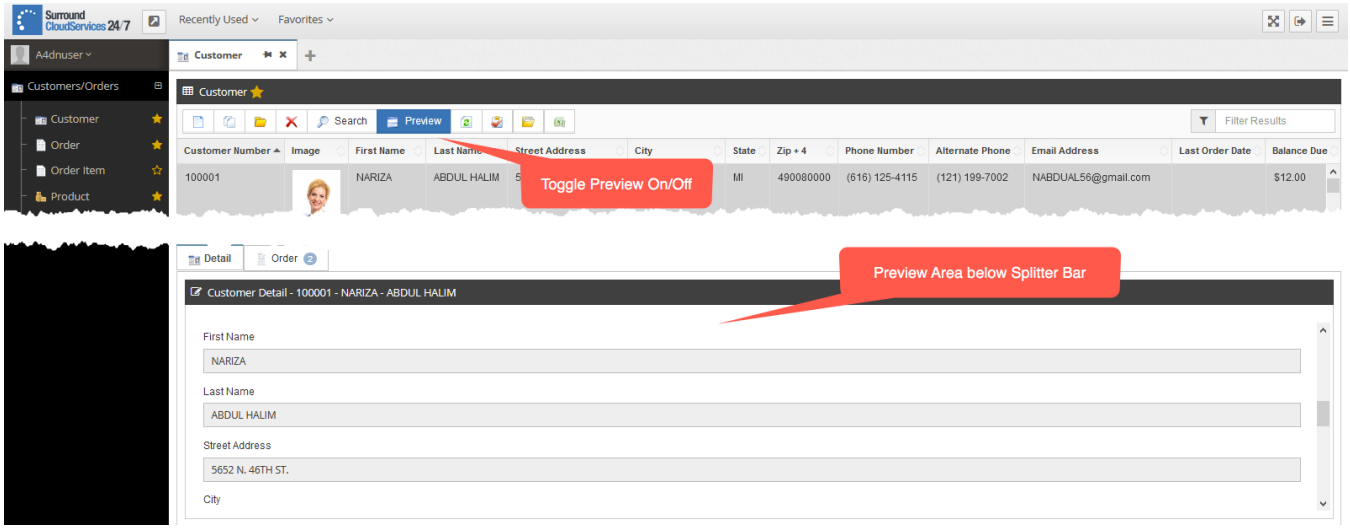

The default layout of the datagrid is half of the available window space when the web application is loaded. If you need more room to scroll through the records you can use the splitter bar to resize the content area panels, by clicking and holding your mouse while dragging the splitter to resize, or change the orientation of the content area from horizontal panes to vertical panes by clicking the reorientation button on the splitter bar.

The vertical content panes allow you more room to scroll through the datagrid while previewing the record on the right. The placement of tools and features is consistent with the horizontal view to make transitioning between horizontal and vertical panes simply a matter of preference and productivity.

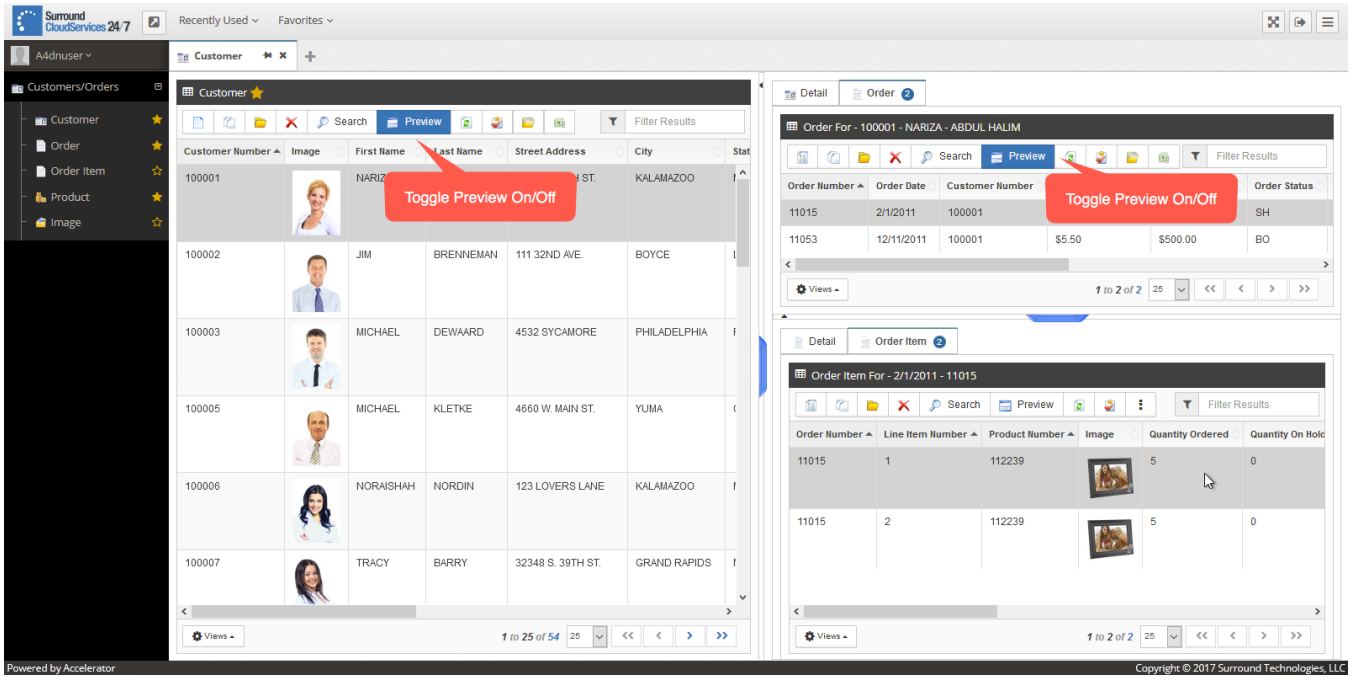
The bottom half of the content area, when in horizontal mode, is called the preview area. When a record is selected from the datagrid, a preview of this data is available in the preview area below it, or on the side if you have selected a vertical pane view of the content area.

To enable the preview area, you must toggle the Preview button on in the datagrid toolbar.

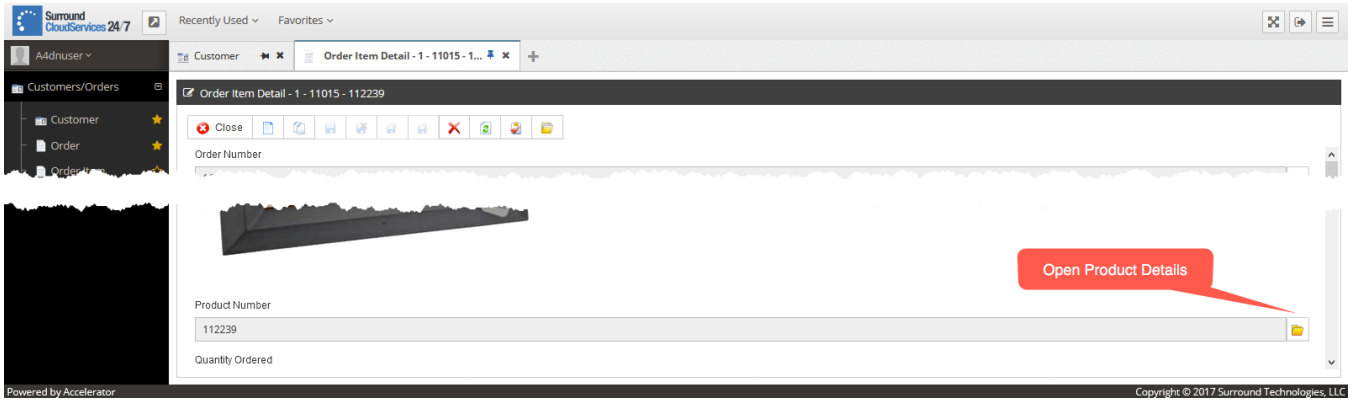



The Preview function is available in any datagrid where applicable. This allows users to get anywhere from anywhere without the need to open additional windows unless desired.

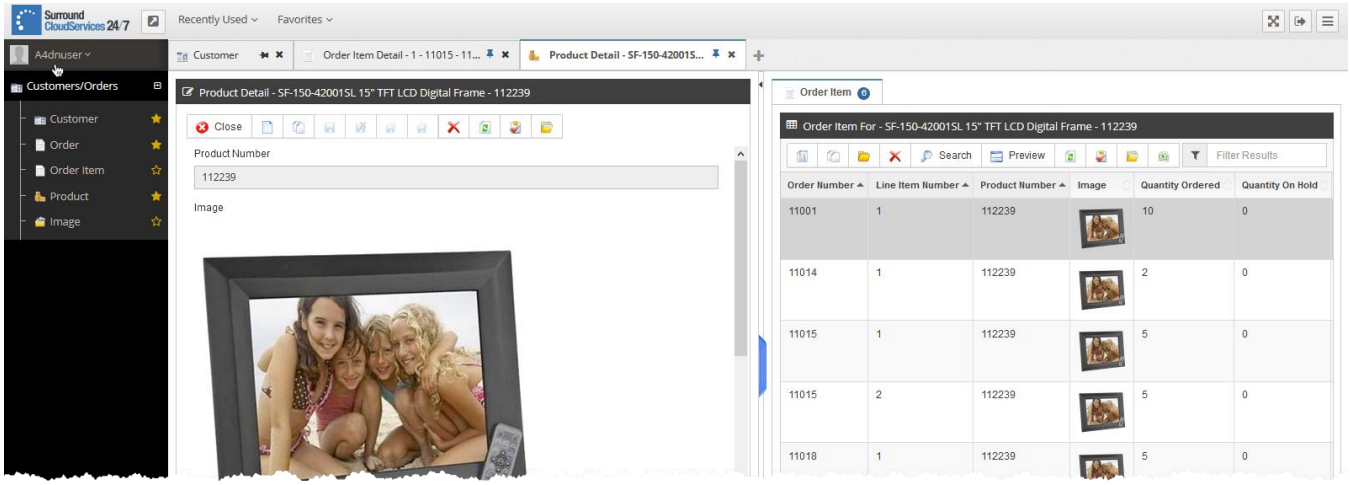
As an example, in the preview of a customer record, we can see orders placed by that customer. Using the same preview functionality in the order history of that customers record, we can drill down to see the items within the order we selected. From the order details, we can choose to see a preview of the item details, or double click the record to open it in a new tab.




Using the preview functionality has allowed us the ability to navigate through multiple modules without leaving the active module we began working with. By double clicking an opening a new tab, users have the ability to compare data side by side or in individual windows depending on the task at hand. From within a detail record, you can also navigate to other modules if needed.



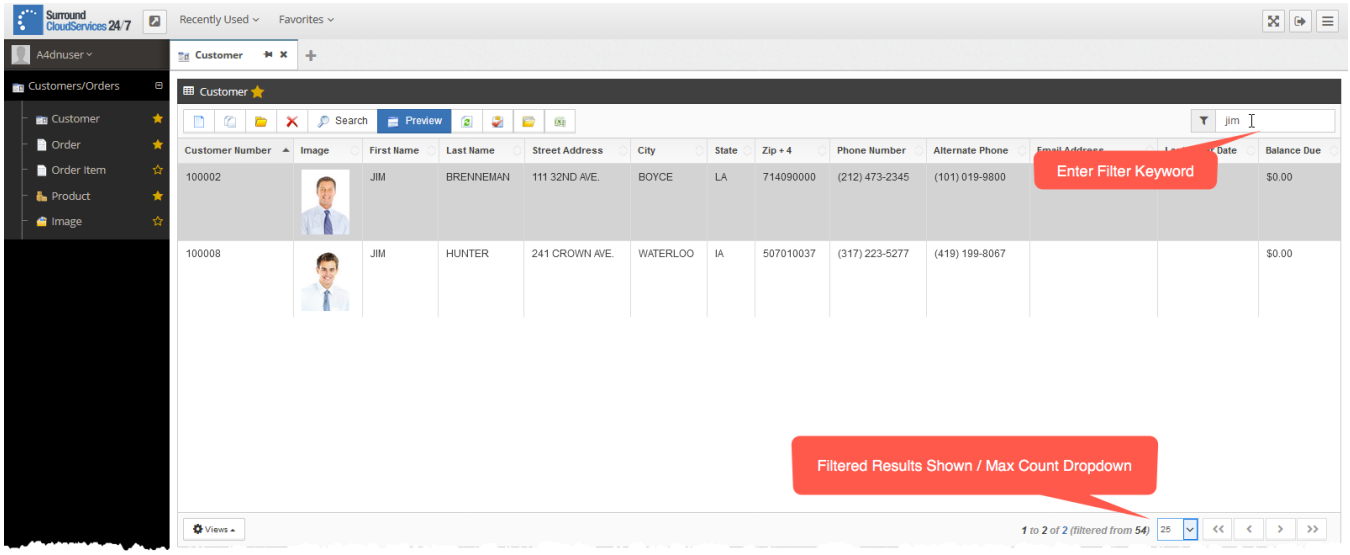

From the order item detail below, users can easily find orders associated with the item by clicking the open product details button. A new tab opens which will show all orders associated with the specified item.



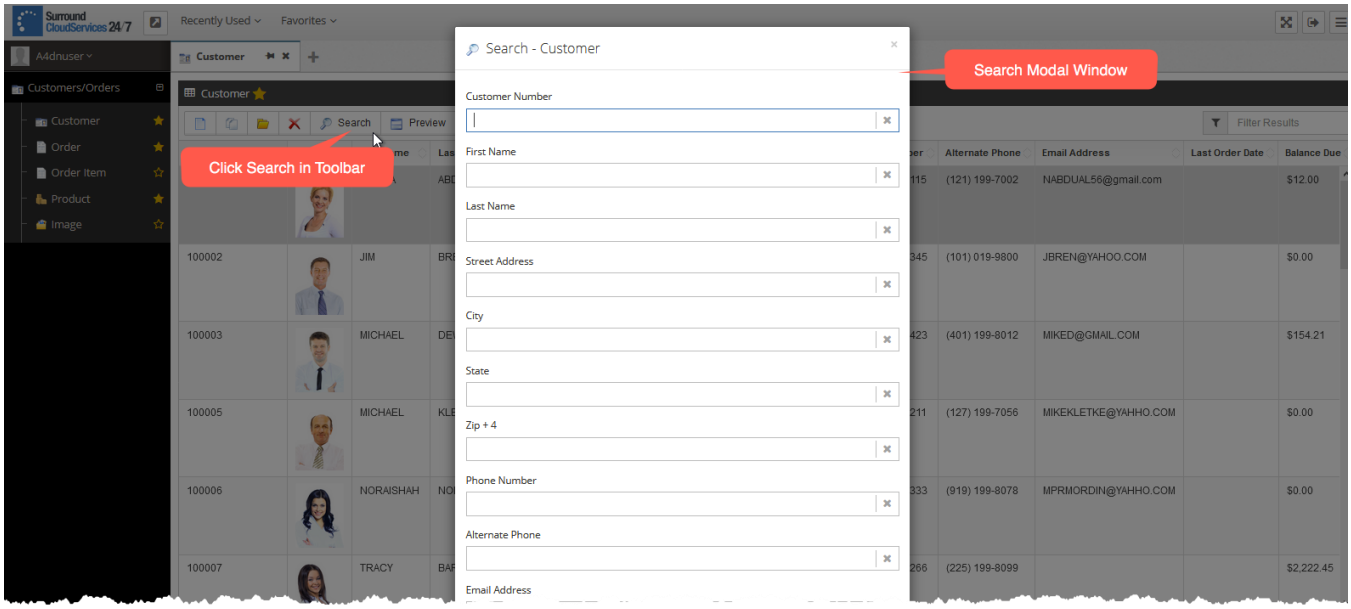

The system is designed so that data can be retrieved across files in the database with ease and without the need to have multiple detail records open at a time. Users are in control of how they navigate the system and how the data is presented.

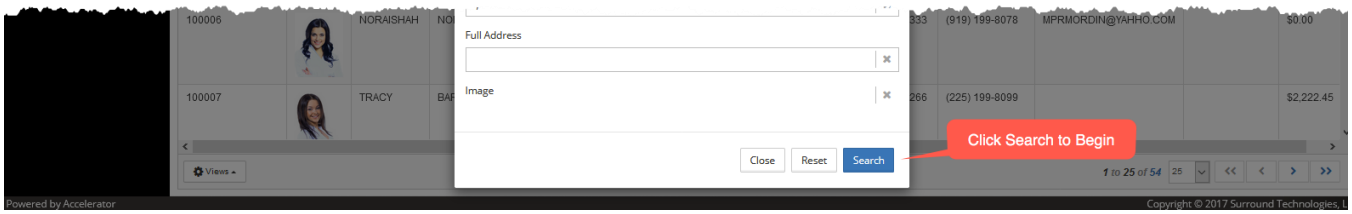
Another way to navigate to specific data would be to filter or search for the data from within the active module. Filtering for data will look for a specified keyword in the dataset currently loaded in the datagrid, whereas Searching for data will go back to the server and search across the entire file. Searching is also capable of using multiple criteria to return specific data. We will review both functions below.

Filtering will only function on data currently loaded in the datagrid. If the max count in the datagrid is set to 25, then the specified keyword will only be checked against the 25 records in the datagrid. To filter on more results change the max count, or use the Search functionality instead.

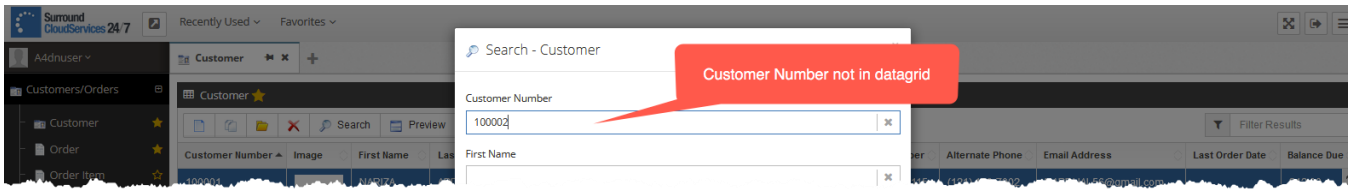


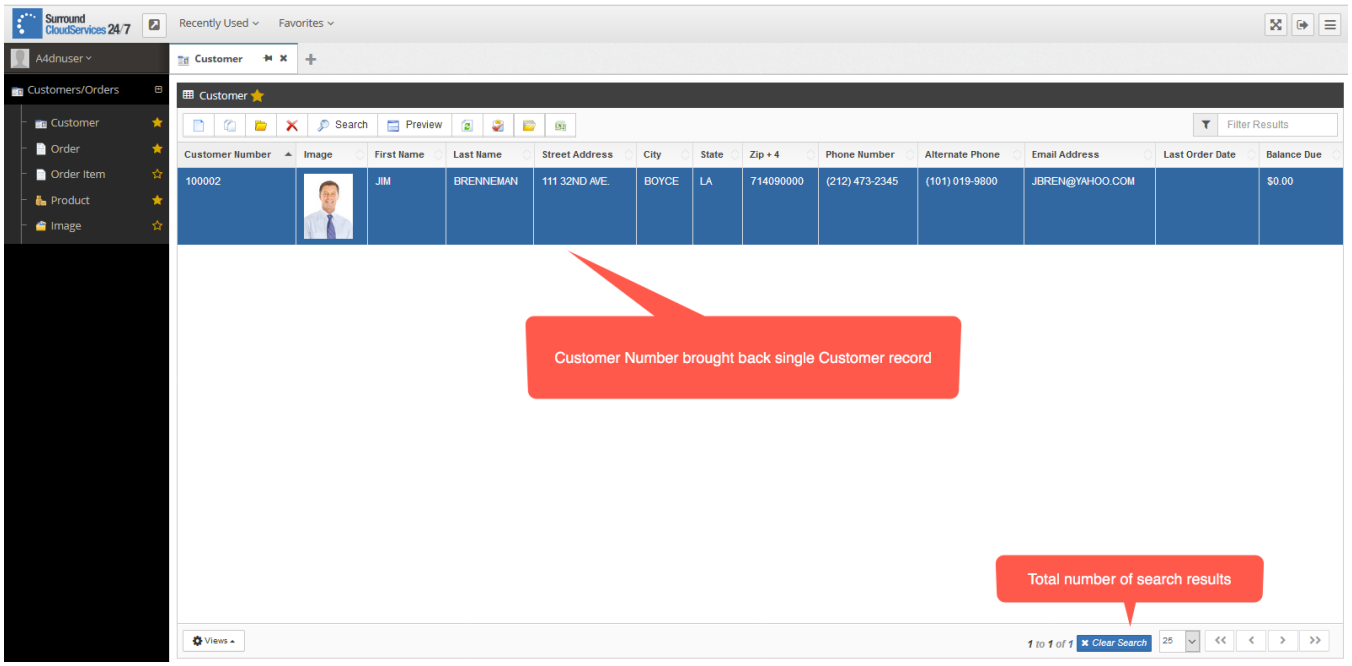

To search for data in the active module, click the Search button in the datagrid toolbar to bring up the Search Modal Window. From this modal popup you can enter whatever criteria you wish to search, and click Search at the bottom of the popup.





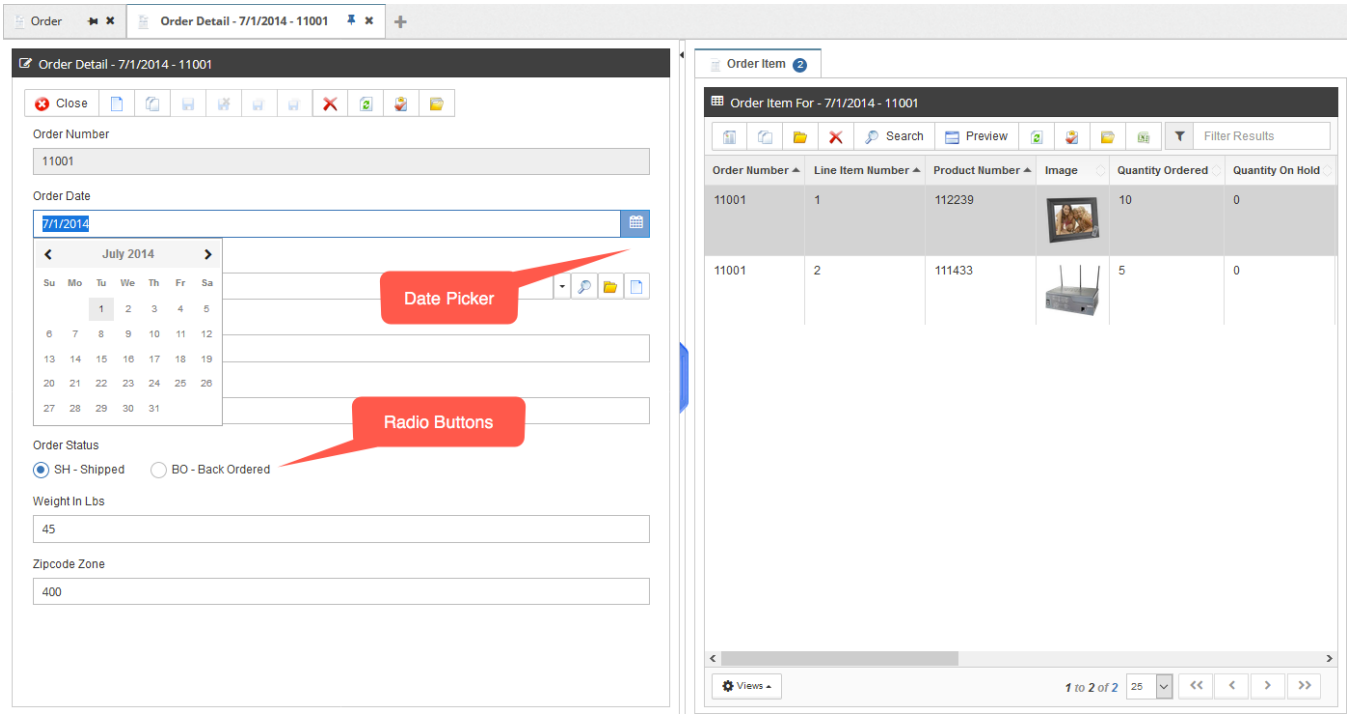

A search can take place across multiple files. When searching for a customer, we can specify a search criteria of customer number and the search will bring back the appropriate result(s).



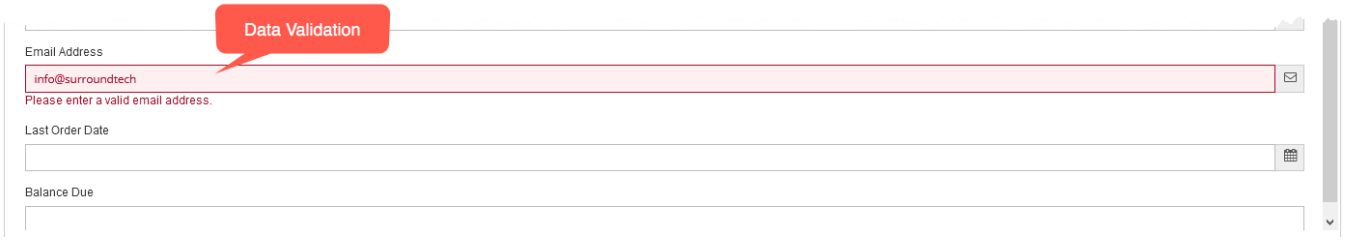




The search functionality is available to any module in the system and can be customized during generation by creating joins across files or tables. This allows users to search for data more efficiently and find what they are looking for faster.

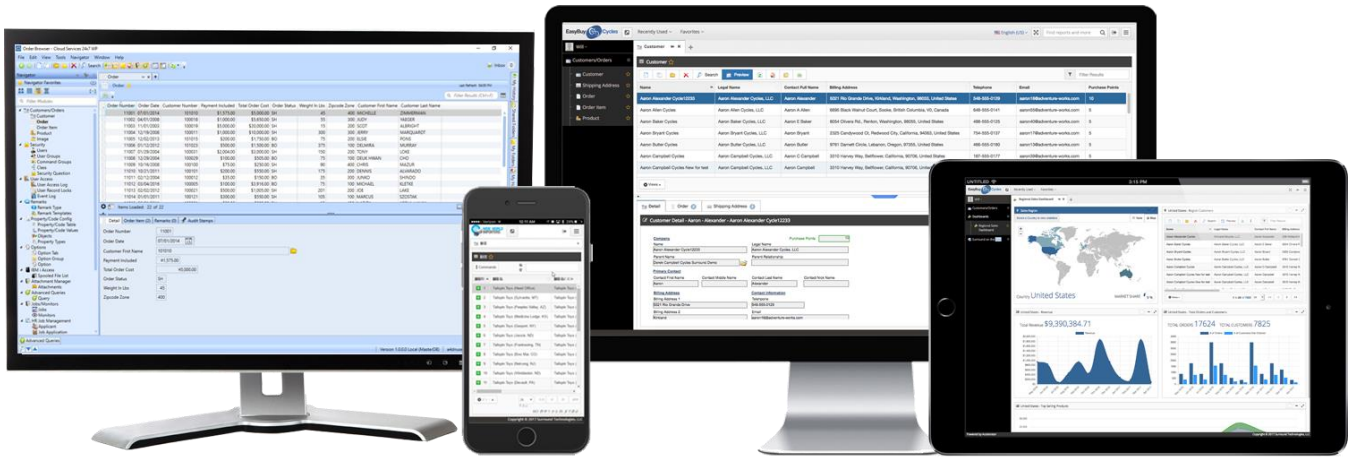
When working with data in the Accelerator there are various controls that can help with presenting it in the most productive way possible. Some examples of controls are Date Picker, Radio Buttons, Checkboxes, drop downs, and more. Each control serves a specific purpose and fulfills an expected response to the user interaction. For example, an order status may only be set to a single value. Checkboxes presented in this situation may imply multiple values are allowed, where as the radio button control implies only a single value may be selected at a time.




Controls play an integral part of a systems user experience, and help control the data integrity of the system. However, data validation at the field level is critical to an efficient and accurate workflow. Accelerator provides validation built-in with alerts guiding the user directly to the location and cause of the erroneous data input.



The Accelerator has been designed to generate a state-of-the-art user friendly system out of the box. With so much attention to detail with the user experience of the application, it was important that any user on any device have access to the same features and functionality regardless of what operating system or device they had.



The entire line of business web application is based off of the award winning WPF Accelerator application for Windows. Users of the Windows application are able to access their systems on any device with a browser and work in tandem with the Windows application or simply pick up where they left off if they need to walk away from their machine. The UI's are complimentary so that there is a minimal learning curve when transitioning between systems.

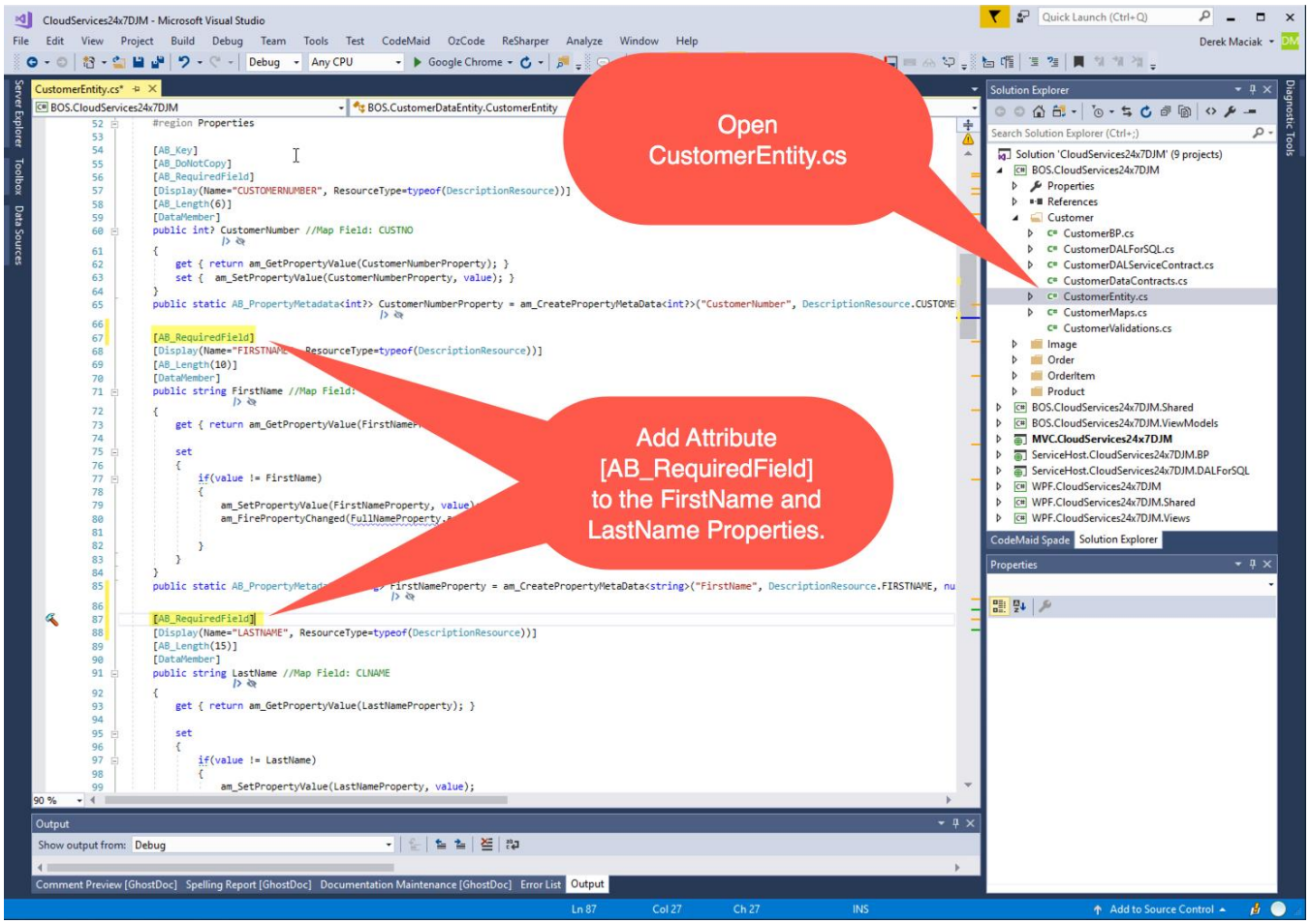
To see a full UX walk through of Accelerator for Windows, Web and Mobile devices visit http://www.surroundtech.com/accelerator-videos . You'll find quick previews, or extended tutorials from registering for free trial download to generating and enriching full functioning applications.

# 6. Customizing the Application

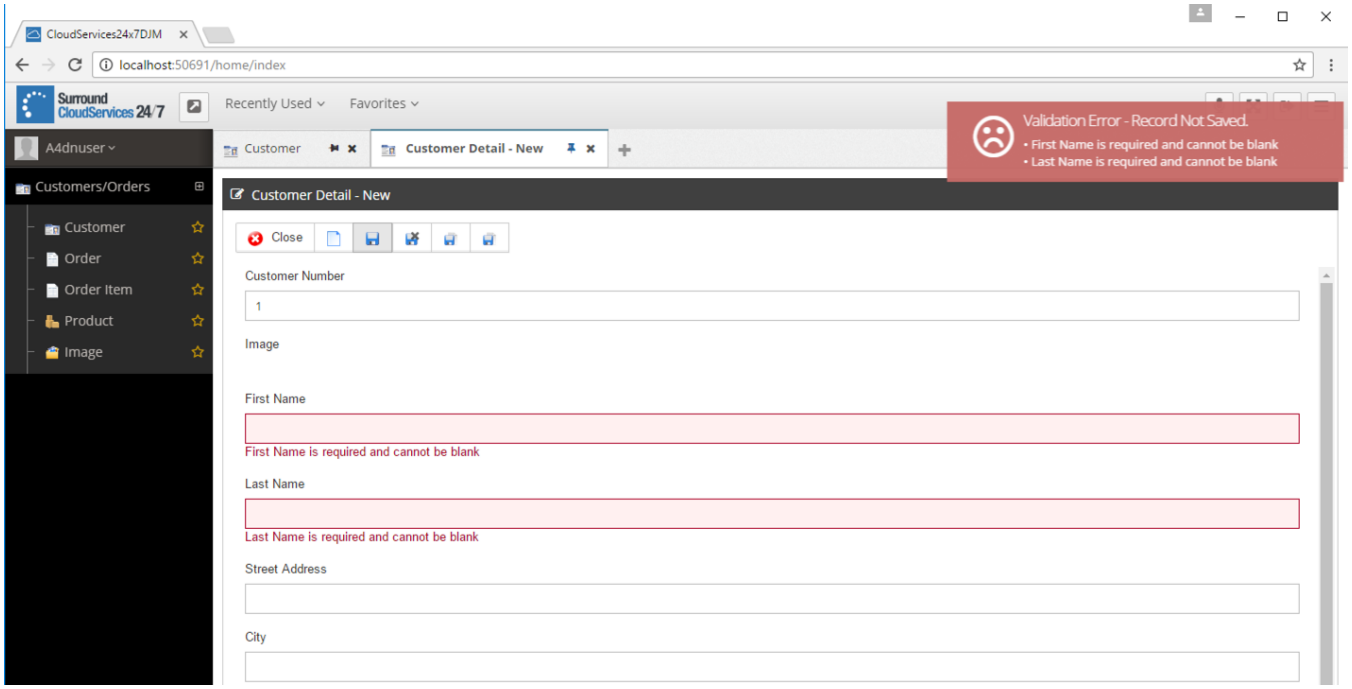
After Generation, all development can be done in Visual Studio. The Accelerator Product has many helper classes that can assist and accelerate many common developer tasks. This section will give you an idea of how easy it can be to customize the code.

## 6.1. Required Fields

Currently only the Customer Number is required on the customer detail. Let's make the First Name and Last Name required as well. Open the CustomerEntity.cs and add the AB_RequiredField attribute to the First Name and Last Name properties.



Run the MVC System and create a new customer. Just put in a customer number and try to save the record. You should get a client side validation error.



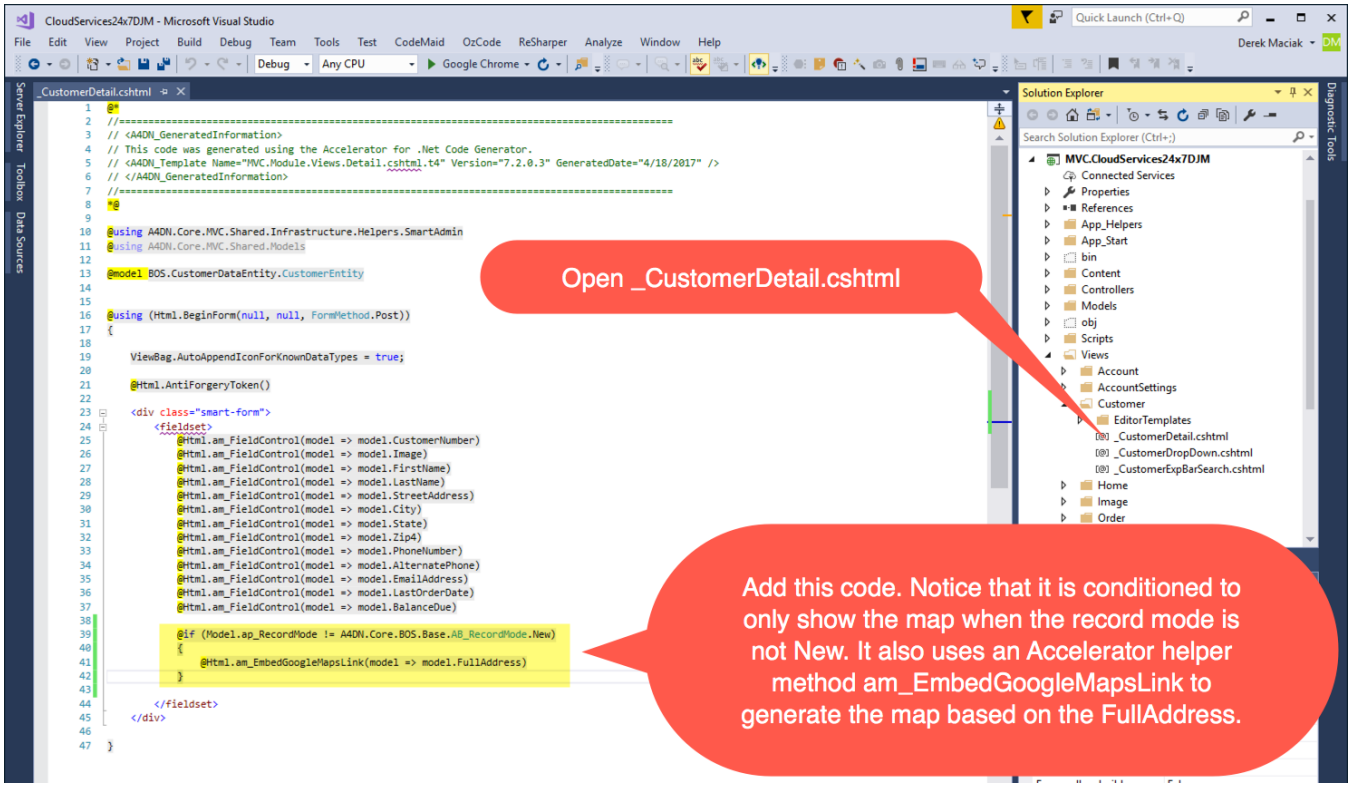

## 6.2. Add Google Map to Customer

When the Customer Entity was generated, the OGT Plugin created a Full Address property that concatenates the Street Address, City, State and Zip Code. This property can be used to display a Google map.

```
[Display(Name="FULLADDRESS", ResourceType=typeof(BOS.CloudServices24x7DJM.Shared.Properties.DescriptionResource))]
[AB_VirtualMember("StreetAddress", "City", "State", "Zip4")]
public string FullAddress //Map Field: FULLADDRESS
{
    get
    {
        return string.Join(" ", StreetAddress, City, State, Zip4);
    }
}
public static AB_PropertyMetadata<string> FullAddressProperty = am_CreatePropertyMetaData<string>("FullAddress", DescriptionResource.FULLADDRESS
```

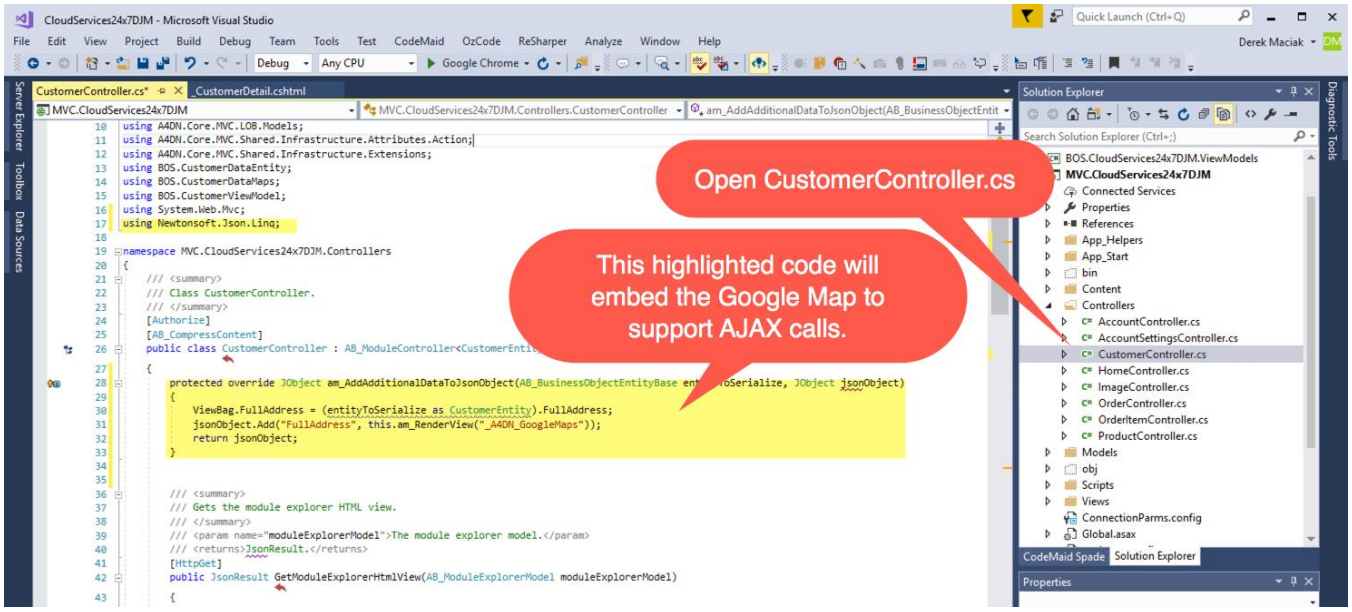
Open the Customer View _CustomerDetail.cshtml and add the highlighted code.

```
@if (Model.ap_RecordMode != A4DN.Core.BOS.Base.AB_RecordMode.New)
{
 @Html.am_EmbedGoogleMapsLink(model => model.FullAddress)
}
```

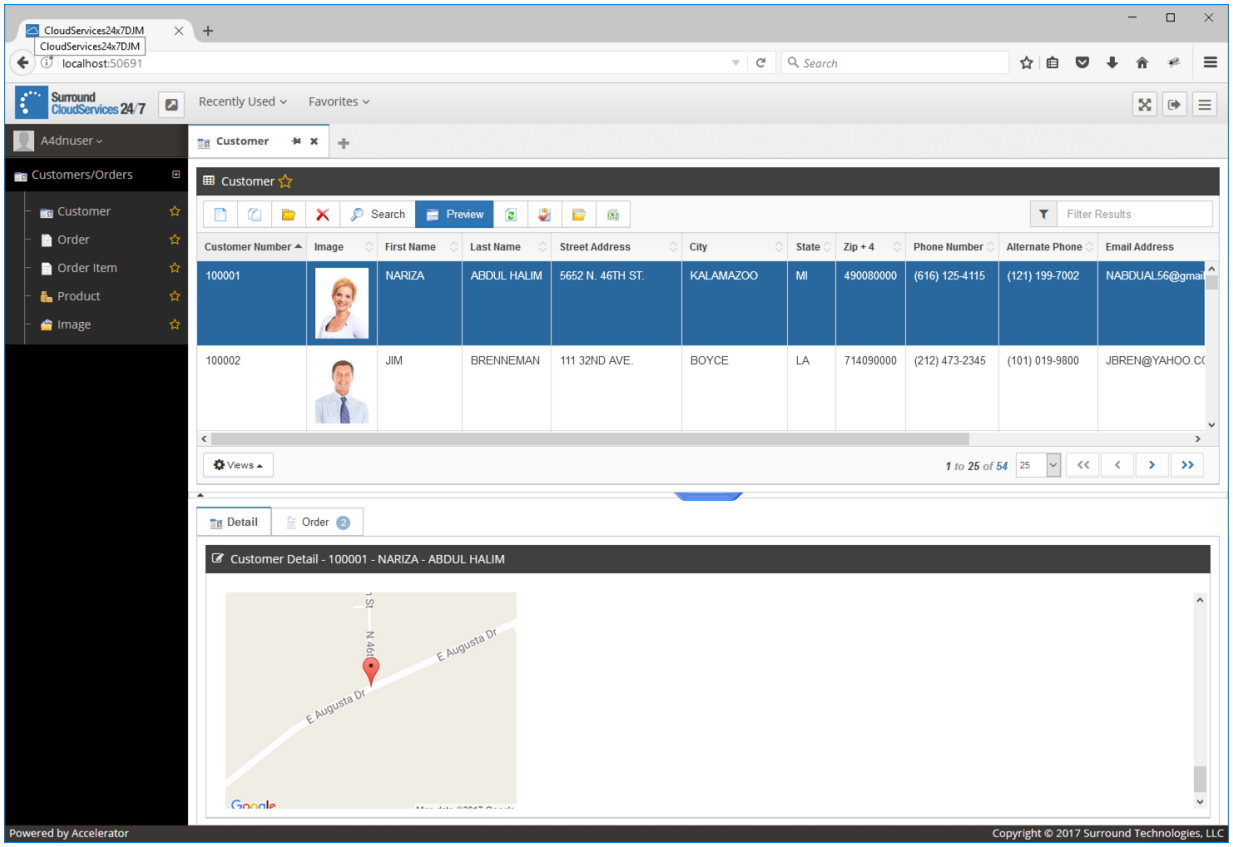



Open the Customer Controller _CustomerDetail.cshtml and add the highlighted code.

```
using Newtonsoft.Json.Linq;

protected override JObject am_AddAdditionalDataToJsonObject(AB_BusinessObjectEntityBase entityToSerialize, JObject jsonObject)
    {
      ViewBag.FullAddress = (entityToSerialize as CustomerEntity).FullAddress;
      jsonObject.Add("FullAddress", this.am_RenderView("_A4DN_GoogleMaps"));
      return jsonObject;
    }
```




Run the MVC System to see the Google Map. Select different customers and watch the Google map update.



# 7. Going Further with Mobile: Installed and Fully Native

## 7.1. Cordova

You can use Cordova (https://cordova.apache.org) to install your standard web technologies on platforms such as Android, Blackberry 10, iOS, OS X, Ubuntu, Windows and WP8.

Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc.

Use Apache Cordova if you are:

- a mobile developer and want to extend an application across more than one platform, without having to re-implement it with each platform's language and tool set.
- a web developer and want to deploy a web app that's packaged for distribution in various app store portals.
- a mobile developer interested in mixing native application components with a *WebView* (special browser window) that can access device-level APIs, or if you want to develop a plugin interface between native and WebView components.

https://cordova.apache.org/docs/en/latest/guide/overview/index.html

## 7.2. Xamarin

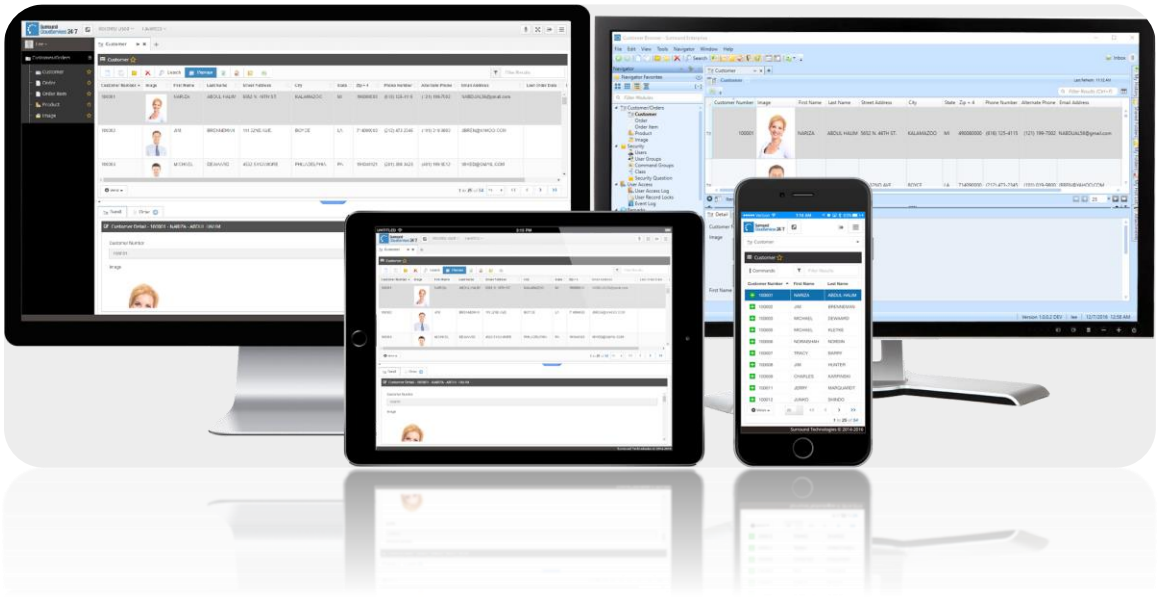
You can use Xamarin (https://www.xamarin.com) to deliver native Android, iOS, and Windows apps, using existing .NET skills, teams, and code.

Build native apps for multiple platforms on a shared C# codebase. Use the same IDE, language, and APIs everywhere.

- Native UI, native API access, and native performance
- Anything you can do in Objective-C, Swift, or Java you can do in C# with Xamarin
- Ship cutting-edge apps with same-day support for new OS releases

## 8. Accelerator Trial – Learn More

Surround CloudServices 24x7



CloudServices24x7 was originally designed as Surround Technologies solution to the **COMMON UI Modernization Challenge.**

To learn more about the CloudServices24x7 Demo System or the COMMON Challenge visit our microsite: http://cloudservices24x7.surroundtech.com

Want to take the Accelerator out for a test drive at home or at work? Go for it. Download the FREE Trial today and recreate everything you've seen in today's lab, or use your own data and start creating software your users will love!

Download your free copy of Accelerator:
**http://www.surroundtech.com/trial**