Programming with TypeScript and the D3 Library

Setup Time: 60 minutes

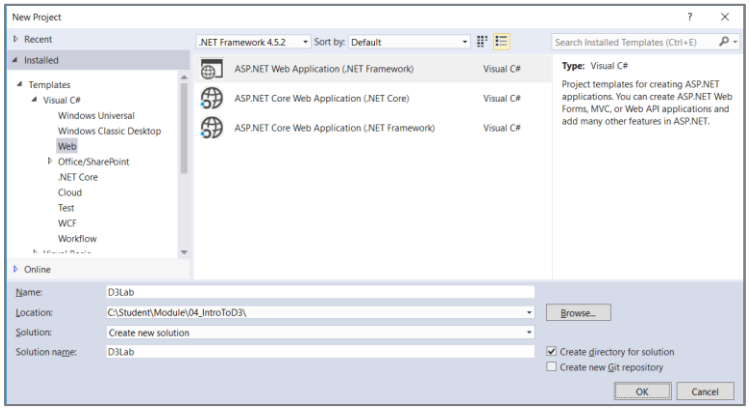
Lab Folder: C:\Student\Module\01_GettingStarted\Lab

Overview: The lab has been designed to provide you with hands-on experience developing with TypeScript and the D3.js library. While this lab has been created for developers who are learning to work with Power BI, the lab itself will not involve the Power BI platform in any way. Instead, you will create a simple web application project in Visual Studio and you will configure this project by adding NuGet packages for jQuery and the D3 library along with the NuGet packages for their typed definitions which provides the strongly-typed when programming in TypeScript.

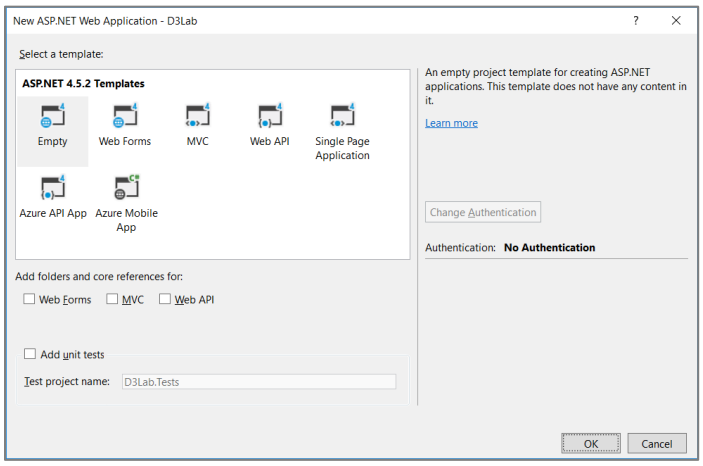
Exercise 1: Create a Visual Studio Project to use TypeScript, jQuery and D3

In this exercise, you will create a new ASP.NET Web Application project using Visual Studio 2017 named **D3Lab** and you will add the required NuGet packages to support programming in Typescript using jQuery and the D3 library. After that, you will add HTML, CSS and TypeScript to get the D3Lab web application project up and running.

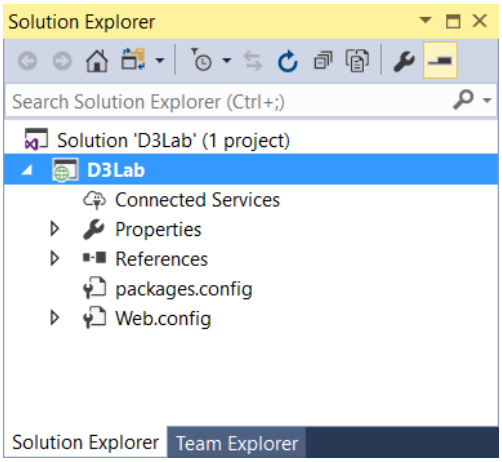
1. Create the new solution in Visual Studio 2017:
   a) Create a new project by selecting the menu command **File > New > Project**.
   b) In the **New Project** dialog…
      i) Select the **ASP.NET Web Application (.NET Framework)** project template under the **Templates > Visual C# > Web** section.
      ii) Enter a name of **D3Lab**.
      iii) Enter a location of **C:\Student\Module\01_GettingStarted\Lab**.
      iv) Click the **OK** button.



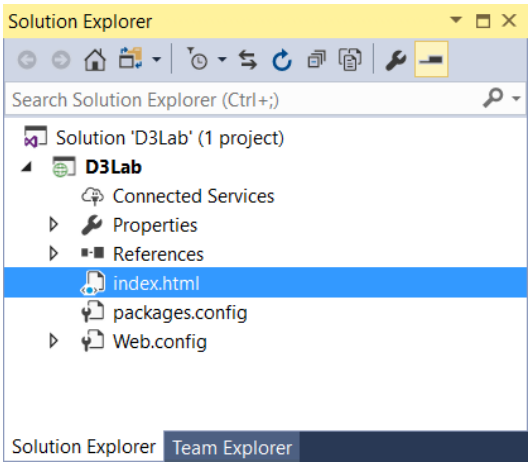
   c) In the New **ASP.NET Web Application** dialog, select the **Empty** project template.
   d) Make sure **Authentication** is set to **No Authentication**.
   e) Make sure the **Host in the cloud** checkbox is unchecked.
   f) Click the **OK** button to create the new project.



2. Once Visual Studio has created the project, familiarize yourself with the structure of the project in the **Solution Explorer**. As you can see the project doesn't contain many files at this point.



3. Add a new HTML file named **index.html** to the root folder.
   a) In Solution Explorer, right-click the top-level project node and select **Add > HTML Page**.
   b) Enter an **Item name** of **index.html**.
   c) You should now see **index.html** in the Solution Explorer in the root folder of the **D3Lab** project.

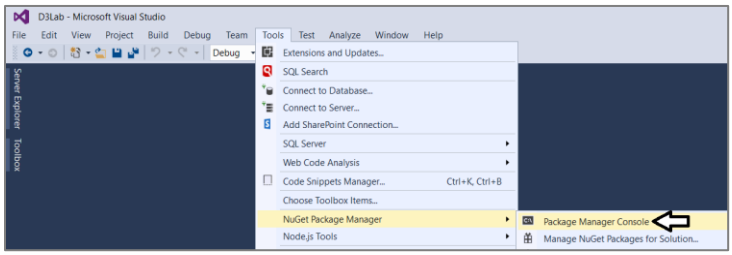


d) Copy and paste the following starter HTML code into **index.html**.

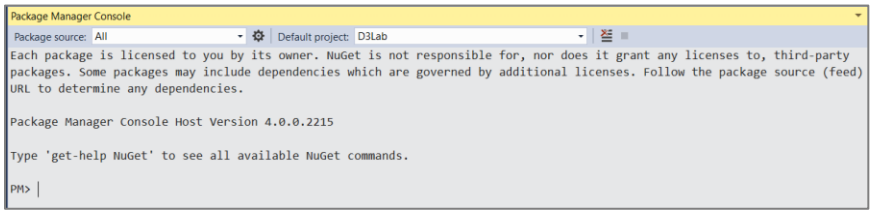
```
<!DOCTYPE html>
<html>
<head>
  <title>D3 Lab</title>
  <meta charset="utf-8" />
</head>
<body>

  <div id="banner">
    <div id="app-icon"></div>
    <div id="app-title">
      D3 Lab<span id="visualName"></span>
    </div>
  </div>

  <div id="left-nav" class="navigationPane navigationPaneCollapsed">
  </div>

  <div id="content-body" class="contentBody contentBodyNavigationCollapsed">
  </div>

</body>
</html>
```

4. Configure the **D3Lab** project with the required set of NuGet packages

a) From the Visual Studio menu, select the command **Tools > NuGet Package Manager > Package Manager Console**.



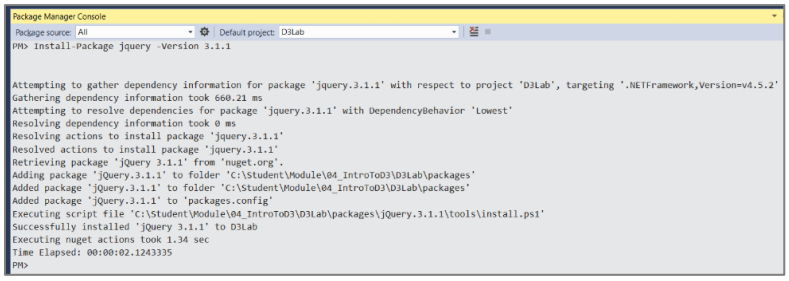
b) You should now see the **Package Manage Console** with a **PM>** command prompt as shown in the following screenshot



c) Type in the following command and press **ENTER** to install the NuGet package for **jQuery** version **3.1.1**.

```
Install-Package jquery -Version 3.1.1
```

d) You should be able to see the details as the installation completes in the **Package Manager Console** window.



e) Type in the following command and press **ENTER** to install the **DefinitelyTyped** package for **jQuery** version **3.1.1**.

```
Install-Package jquery.TypeScript.DefinitelyTyped -Version 3.1.1
```

f) Type in the following command and press **ENTER** to install the NuGet package for **D3** version **3.5.17**.

```
Install-Package d3 -Version 3.5.17
```

g) Type in the following command and press **ENTER** to install the **DefinitelyTyped** package for **D3** version **2.7.3**.

```
Install-Package d3.TypeScript.DefinitelyTyped -Version 2.7.3
```

h) Type in the following command and press **ENTER** to install the NuGet package for **bootstrap** version **3.1.1**.

```
Install-Package bootstrap -Version 3.1.1
```

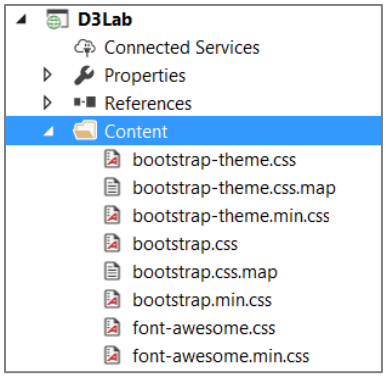
i) Type in the following command and press **ENTER** to install the NuGet package for **FontAwesome** version **4.7.0**.

```
Install-Package FontAwesome -Version 4.7.0
```

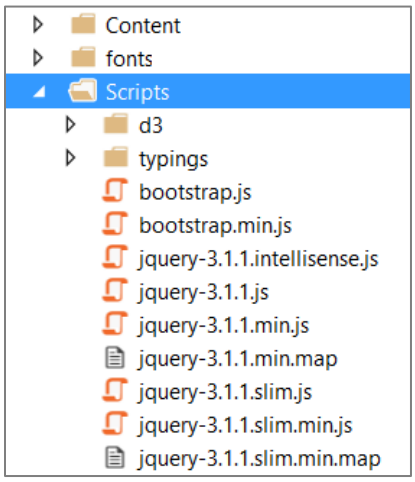
You have now added all the external packages you will need for the **D3Lab** project.

5. Examine the 3rd part package files that have been added to the project.

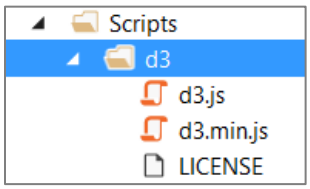
a) In Solution Explorer, expand the **Content** folder and the **fonts** folder to see the files added from the **bootstrap** package.



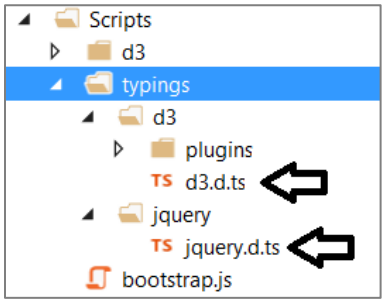
b) Expand the **Scripts** folder to see the files for the JavaScript library packages.



c) Expand the **Scripts/d3** folder. You should see the main JavaScript library file for D3 library named d3.js.



d) Expand the **Scripts/typings** folder and locate the two **d.ts** files in the **DefinitelyTyped** NuGet packages for **jQuery** and **D3**.



Note that you will not have to take any additional steps to include or reference **d3.d.ts** or **jquery.d.ts**. Once you have installed the DefinitelyTyped packages to your project, you will be able to program against jQuery and the D3 library in a strongly-typed fashion from within any TypeScript source file you add to the project.

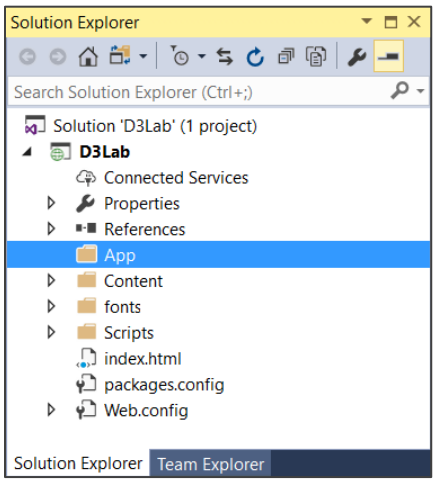
6. Add CSS links to bootstrap and font-awesome.
   a) Inside **index.html**, update to the **head** section with links to **bootstrap.css**, **bootstrap-theme.css** and **font-awesome.css**.

```
<head>
  <title>D3 Lab</title>
  <meta charset="utf-8" />
  <link href="Content/bootstrap.css" rel="stylesheet" />
  <link href="Content/bootstrap-theme.css" rel="stylesheet" />
  <link href="Content/font-awesome.css" rel="stylesheet" />
</head>
```

7. At the bottom of the page just before the closing **body** tag, add script links to **jquery-3.1.1.js**, **bootstrap.js** and **d3.js**.

```
<body>

  <div id="banner"></div>

  <div id="left-nav"></div>

  <div id="content-body"></div>

  <script src="Scripts/jquery-3.1.1.js"></script>
  <script src="Scripts/bootstrap.js"></script>
  <script src="Scripts/d3/d3.js"></script>

</body>
</html>
```

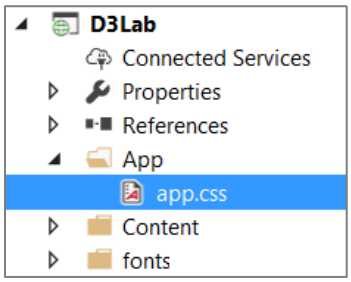
8. Create a new top-level folder in the **D3Lab** project named **App**.
   a) Right-click on the top-level project node for the **D3Lab** project and select the **Add > New Folder** command.
   b) Give the new folder a name of **App**.



9. Import an existing CSS file named **app.css** into your project to provide the CSS styles you will use in this lab.
   a) Right-click on the **App** folder and select the **Add > Existing Item…** command.
   b) When prompted for a file path, enter the following path to the CSS source file named **app.css**.

```
C:\Student\Module\01_GettingStarted\Lab\StarterFiles\app.css
```

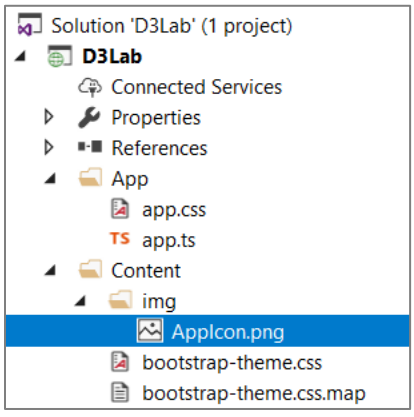
   c) The source file named **app.css** should now be located inside the **App** folder.



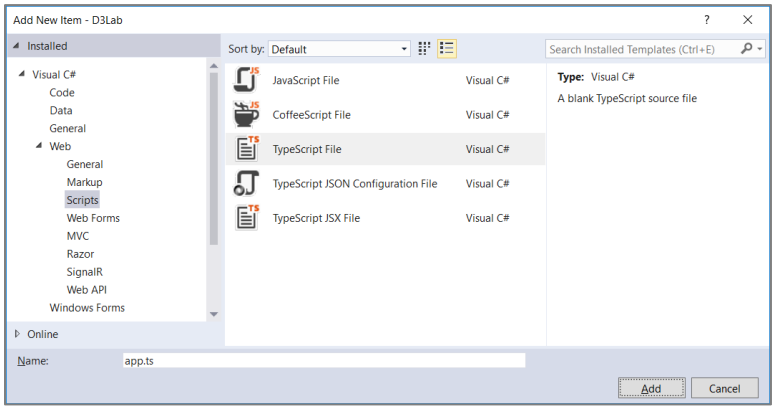
10. Add an image file for the app icon named **AppIcon.png**.
   a) Create a new child folder inside the **Content** folder named **img**.
   b) Locate the image file at the following path and copy it into the **img** folder.

```
C:\Student\Module\01_GettingStarted\Lab\StarterFiles\AppIcon.png
```

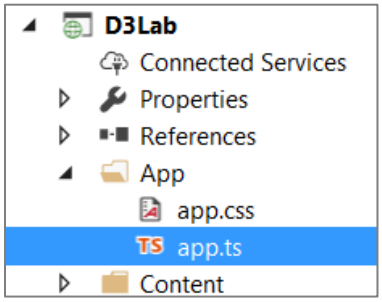
   c) The image file named **AppIcon.png** should now be located inside the **img** folder.



11. Create a new TypeScript source file in the **App** folder named **app.ts**.
    a) Right-click on **App** folder and select the **Add > New Item** menu command.
    b) On the left side of the N**ew Item** dialog, select **Visual C# > Web > Scripts**.
    c) Select **TypeScript File** as the item template.
    d) Provide a file name of **app.ts**.
    e) Click the **Add** button in the bottom right of the dialog.



    f) There should now be a TypeScript file in the **App** folder named **app.ts**.



12. Implement the traditional "hello world" application code in **app.ts**.
    a) Add the following code in **app.ts** to create a startup function using the jQuery document ready event.

```
module myApp {

  $( function() {
    // initialize the app
  });

}
```

Wait just a second. Don't use the same old boring JavaScript syntax to create an anonymous function. TypeScript developers will consider you "old school". Instead, use the trendy new arrow function syntax to show folks how much you love TypeScript.

    a) Modify your code to use TypeScript arrow function syntax as shown in the following code listing.

```
module myApp {

  $( () => {

    // initialize the app

  });

}
```

b) Create a variable named **contentBoxDiv** and initialize it with a jQuery object which wraps the div element in **index.html** which has an id of **content-body**. Make sure to declare the **contentBoxDiv** variable so it is strongly-typed to the **JQuery** type.

```
// initialize the app
let contentBoxDiv: JQuery = $("#content-body");
```

c) Execute the **text** method and the **css** method on the **contentBoxDiv** object using the following TypeScript.

```
module myApp {

  $( () => {

    // initialize the app
    let contentBoxDiv: JQuery = $("#content-body");

    contentBoxDiv
      .text("Hello TypeScript")
      .css({
        "padding": "12px",
        "color": "Blue",
        "font-size": "32px"
      });

  });

}
```

d) Save your changes to **app.ts**.

13. Update **index.html** with links to **app.css** and **app.js**.

a) In the **head** section of the **index.htm**, add a new link to **app.css**. under the links to the 3rd party CSS files.

```
<head>
  <title>D3 Lab</title>
  <meta charset="utf-8" />
  <link href="Content/bootstrap.css" rel="stylesheet" />
  <link href="Content/bootstrap-theme.css" rel="stylesheet" />
  <link href="Content/font-awesome.css" rel="stylesheet" />
  <link href="App/app.css" rel="stylesheet" />
</head>
```

b) At the bottom of the page, add a new script link to **app.js** underneath the existing script links to 3rd party libraries.

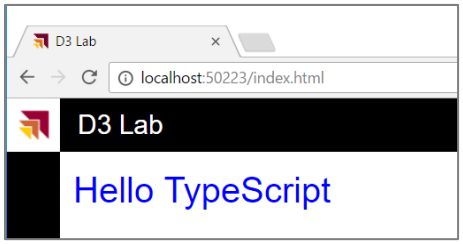
```
  <script src="Scripts/jquery-3.1.1.js"></script>
  <script src="Scripts/bootstrap.js"></script>
  <script src="Scripts/d3/d3.js"></script>
  <script src="App/app.js"></script>

</body>
</html>
```

c) Save your changes **index.html**.

14. Test out the **D3Lab** project using the Visual Studio Debugger

a) Press the **{F5}** key to start up the project in the Visual Studio debugger.

b) When the project starts, it should appear in the browser and match the following screenshot.



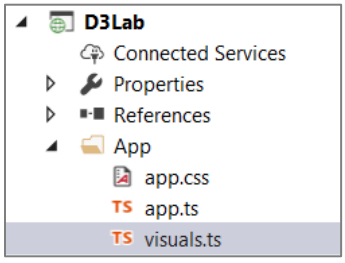
c) Once you have tested **D3Lab**, close the browser window, return to Visual Studio and stop the debugger.

In this initial exercise, you got the D3Lab project up and running and you have begun to write and test code written in TypeScript. You have also added typed definition files that make it possible to program in TypeScript against jQuery in a strongly typed fashion. In the following exercise, you will also begin to program against the D3.js library and its typed definitions as well.

Exercise 2: Implement an Application Design to Load Custom Visuals

In this exercise, you will add two TypeScript interfaces to define the behavior of a custom visual which can be loaded on demand. You will then add some more HTML and TypeScript code to provide user experience to select and load custom visuals.

1. Create a new TypeScript source file named **visuals.ts**.in the **App** folder.
   a) Right-click on **App** folder and select the **Add > New Item** menu command.
   b) On the left side of the N**ew Item** dialog, select **Visual C# > Web > Scripts**.
   c) Select **TypeScript File** as the item template.
   d) Provide a file name of **visuals.ts**.
   e) Click the **Add** button in the bottom right of the dialog.
   f) There should now be a TypeScript file in the **App** folder named **visuals.ts**.



2. Inside **visuals.ts**, add two interfaces named **IViewPort** and **ICustomVisual** and a class named **Viz01**.
   a) Add a new top-level module named **myApp**.
   b) Inside the **myApp** module, add two interfaces named **IViewPort** and **ICustomVisual** and a class named Viz01 as shown in the following code.

```
module myApp {

  export interface IViewPort {}

  export interface ICustomVisual {}

  export class Viz01 { }

}
```

Note that defining these interfaces and class using the **export** keyword makes them available across TypeScript source file boundaries. This will make it possible to program against types defined inside **visuals.ts** when writing code inside **app.ts**.

3. Modify the interface definitions for **IViewPort** and **ICustomVisual**.
   a) Modify the definition of the **IViewPort** interface using the following code.

```
export interface IViewPort {
  width: number;
  height: number;
}
```

   b) Modify the definition of the **ICustomVisual** interface using the following code.

```
export interface ICustomVisual {
  name: string;
  load(container: HTMLElement): void;
  update(viewport: IViewPort): void;
}
```

4. Implement the **Viz01** class.
   a) Modify the top line of the **Viz01** class to implement the **ICustomVisual** interface.

```
export class Viz01 implements ICustomVisual {
```

   b) Update the **Viz01** class with the following code to provide a minimal implementation of the **ICustomVisual** interface.

```
export class Viz01 implements ICustomVisual {

  name = "Visual 1: Hello jQuery";

  load(container: HTMLElement) { }

  public update(viewport: IViewPort) { }

}
```

   c) Underneath the **name** property, add two new private properties named **container** and **message**. Define both of these properties with a type annotation using the **JQuery** type.

```
export class Viz01 implements ICustomVisual {

  name = "Visual 1: Hello jQuery";
  private container: JQuery;
  private message: JQuery;

  load(container: HTMLElement) { }

  public update(viewport: IViewPort) { }
}
```

   d) Implement the **load** method using the following code.

```
load(container: HTMLElement) {

  this.container = $(container);

  this.message = $("<div>")
    .text("Hello jQuery")
    .css({
      "display": "table-cell",
      "text-align": "center",
      "vertical-align": "middle",
      "text-wrap": "none",
      "background-color": "yellow"
    });

  this.container.append(this.message);
}
```

   e) Implement the **update** method using the following code.

```
update(viewport: IViewPort) {

  let paddingX: number = 2;
  let paddingY: number = 2;
  let fontSizeMultiplierX: number = viewport.width * 0.15;
  let fontSizeMultiplierY: number = viewport.height * 0.4;
  let fontSizeMultiplier: number = Math.min(...[fontSizeMultiplierX, fontSizeMultiplierY]);

  this.message.css({
    "width": viewport.width - paddingX,
    "height": viewport.height - paddingY,
    "font-size": fontSizeMultiplier
  });

}
```

   f) Save your changes to **visuals.ts**.

5. Modify **index.html** with a script link to **visuals.ts**.
   a) Move to the bottom of the body section of **index.html** where you have added script links.
   b) Add a new script link to **visuals.js** after the scripts links to 3rd party JavaScript libraries but before the script link to **app.js**.

```
  <script src="Scripts/jquery-3.1.1.js"></script>
  <script src="Scripts/bootstrap.js"></script>
  <script src="Scripts/d3/d3.js"></script>
  <script src="App/visuals.js"></script>
  <script src="App/app.js"></script>

</body>
</html>
```

   c) Save changes to **index.html**.

6. Modify the HTML inside **index.html** to provide the user with a left navigation menu for loading custom visuals.
   a) Inside **index.html**, locate the **div** elements with the id values of **left-nav** and **content-body**.

```
<body>

  <div id="banner">
    <div id="app-icon"></div>
    <div id="app-title">
      D3 Lab<span id="visualName"></span>
    </div>
  </div>

  <div id="left-nav" class="navigationPane navigationPaneCollapsed">
  </div>

  <div id="content-body" class="contentBody contentBodyNavigationCollapsed">
  </div>

</body>
```

   b) Replace the existing **left-nav** div element inside **index.html** by copying and pasting the following HTML code.

```
<div id="left-nav" class="navigationPane navigationPaneCollapsed" >
  <div id="left-nav-body">
    <div class="leftNavSection">
      <div id="left-nav-toggle">
        <div class="leftNavIcon fa fa-bar-chart"></div>
        <div class="leftNavTitle leftNavItem leftNavHide">Visual Picker</div>
      </div>
      <ul id="visualPickerMenu" class="leftNavMenu leftNavItem leftNavHide">
      </ul>
    </div>
  </div>
</div>
```

   c) Replace the existing **content-body** div element inside **index.html** by copying and pasting the following HTML code.

```
<div id="content-body" class="contentBody contentBodyNavigationCollapsed">
  <div id="viz" />
</div>
```

7. Replace the TypeScript code you have written inside **app.ts** with some pre-provided code to initialize the app's navigation scheme.
   a) Using Windows Explorer, locate the file named **app.cs.txt** in the **Students** folder.

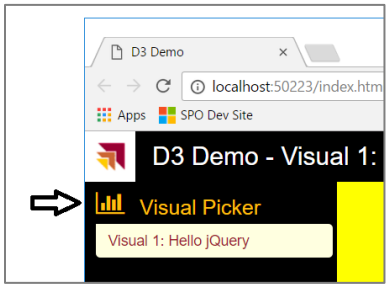
```
C:\Student\Module\01_GettingStarted\Lab\StarterFiles\app.ts.txt
```

   b) Double-click on **app.ts.txt** to open this file in Notepad.
   c) Copy the contents of **app.ts.txt** to the Windows clipboard.
   d) Return to Visual Studio and place the cursor inside the editor window for **app.ts**.
   e) The code you have added inside **app.ts** should now match the code shown in the following screenshot.

```
module myApp {

  var leftNavCollapsed: boolean = true;
  var loadedVisual: ICustomVisual;

  var visuals: ICustomVisual[] = [
    new Viz01()
  ];

  $(() =>...);

  function onNavigationToggle()...

  function onVisualPickerSelect(evt)...

  function LoadVisual(visual: ICustomVisual)...

  export function updateUI()...

}
```

8. Test out the **D3Lab** project using the Visual Studio Debugger
   a) Press the **{F5}** key to start up the project in the Visual Studio debugger.
   b) When the project starts, it should appear in the browser and match the following screenshot.

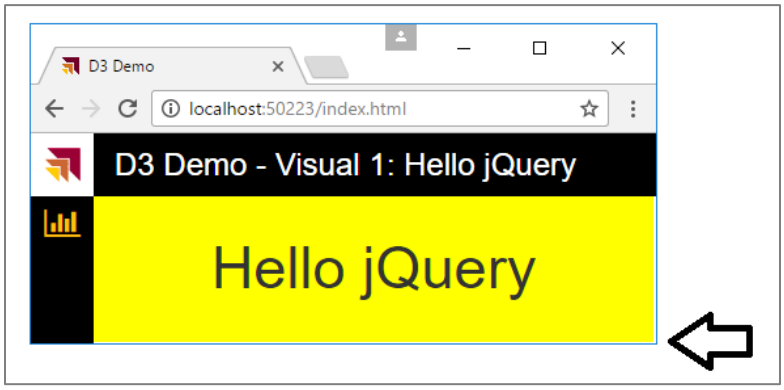
   c) Click on the bar chart icon in the left navigation menu to collapse it.



   d) Notice that the left navigation menu is now collapsed.

If you continue to click on bar chart icon, it will toggle the left navigation menu between a collapsed state and an expanded state.

e) Resize the browser window using the mouse by clicking and holding the left mouse button on the bottom right corner of the window and dragging the lower corner to change the size of the window.



You should see that the **Viz01** class implementation continually updates its output whenever the host window is resized.

Exercise 3: Create a Custom Visual with the D3 Library

In this exercise you will create a new TypeScript class named **Viz02** to implement another custom visual. However, this new custom visual will be different because you will implement using the D3 library.

1. Add a new class named **Viz02** into **visuals.ts**.
   a) Add a new class named **Viz02** inside **visuals.ts** below the **Viz01** class definition using the following starter code.

```
export class Viz02 implements ICustomVisual {

  name = "Visual 2: Hello D3";

  load(container: HTMLElement) { }

  update(viewport: IViewPort) { }

}
```

2. Add a set of private properties to the **Viz02** class implementation.
   a) Add a new private property named **svgRoot** and set its type to **d3.Selection<SVGElementInstance>**.
   b) Add a new private property named **ellipse** and set its type to **d3.Selection<SVGElementInstance>**.
   c) Add a new private property named **text** and set its type to **d3.Selection<SVGElementInstance>**.
   d) Add a new private property named **padding** with a type of **number** and a value of **20**.

```
public name = "Visual 2: Hello D3";

private svgRoot: d3.Selection<SVGElementInstance>;
private ellipse: d3.Selection<SVGElementInstance>;
private text: d3.Selection<SVGElementInstance>;
private padding: number = 20;
```

3. Implement the **load** method of the **Viz02** class.
   a) Begin by appending a new **svg** element to the HTML container element and assigning a reference to the **svgRoot** property.

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

}
```

b) Below the code which appends a new **svg** element, add the following code to append a new ellipse into the **svgRoot** element.

```
this.svgRoot = d3.select(container).append("svg");

this.ellipse = this.svgRoot.append("ellipse")
  .style("fill", "rgba(255, 255, 0, 0.5)")
  .style("stroke", "rgba(0, 0, 0, 1.0)")
  .style("stroke-width", "4");
```

c) Below the code which appends the **ellipse**, add the following code to append a text element to the **svgRoot** element.

```
this.text = this.svgRoot.append("text")
  .text("Hello D3")
  .attr("text-anchor", "middle")
  .attr("dominant-baseline", "central")
  .style("fill", "rgba(255, 0, 0, 1.0)")
  .style("stroke", "rgba(0, 0, 0, 1.0)")
  .style("stroke-width", "2");
```

d) Your implementation of **load** should now match the following code.

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.ellipse = this.svgRoot.append("ellipse")
    .style("fill", "rgba(255, 255, 0, 0.5)")
    .style("stroke", "rgba(0, 0, 0, 1.0)")
    .style("stroke-width", "4");

  this.text = this.svgRoot.append("text")
    .text("Hello D3")
    .attr("text-anchor", "middle")
    .attr("dominant-baseline", "central")
    .style("fill", "rgba(255, 0, 0, 1.0)")
    .style("stroke", "rgba(0, 0, 0, 1.0)")
    .style("stroke-width", "2");
}
```

4. Implement the **update** method of the **Viz02** class.

a) Begin by resizing the width and height of the **svgRoot** element to the width and height of the incoming **viewport** parameter.

```
public update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

}
```

b) Next, create a new variable named **plot** and initialize it with new object that has 4 properties named **xOffset**, **yOffset**, **width** and **height** as shown in the following code listing.

```
var plot = {
  xOffset: this.padding,
  yOffset: this.padding,
  width: viewport.width - (this.padding * 2),
  height: viewport.height - (this.padding * 2),
};
```

c) Use the **plot** variable to resize the **ellipse** element by modifying the four **ellipse** element attributes named **cx**, **cy**, **rx** and **ry** as shown in the following code listing.

```
this.ellipse
  .attr("cx", plot.xOffset + (plot.width * 0.5))
  .attr("cy", plot.yOffset + (plot.height * 0.5))
  .attr("rx", (plot.width * 0.5))
  .attr("ry", (plot.height * 0.5));
```

You need to adjust the font size according to the size of the window.

d) Add three variables of type **number** named **fontSizeForWidth**, **fontSizeForHeight** and **fontSize** and initialize their values using the following code.

```
var fontSizeForWidth: number = plot.width * .20;   // make font smaller as width gets smaller
var fontSizeForHeight: number = plot.height * .35; // make font smaller as height gets smaller
var fontSize: number = d3.min([fontSizeForWidth, fontSizeForHeight]);
```

e) Use the **plot** variable and the **fontSize** variable to resize the **text** element by modifying the five **text** element attributes named **x**, **y**, **width**, **height** and **font-size** as shown in the following code listing.

```
this.text
  .attr("x", plot.xOffset + (plot.width * 0.5))
  .attr("y", plot.yOffset + (plot.height * 0.5))
  .attr("width", plot.width)
  .attr("height", plot.height)
  .attr("font-size", fontSize);
```

f) Your implementation of **update** should now match the following code.

```
public update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding,
    yOffset: this.padding,
    width: viewport.width - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  this.ellipse
    .attr("cx", plot.xOffset + (plot.width * 0.5))
    .attr("cy", plot.yOffset + (plot.height * 0.5))
    .attr("rx", (plot.width * 0.5))
    .attr("ry", (plot.height * 0.5));

  var fontSizeForWidth: number = plot.width * .20;   // make font smaller as width gets smaller
  var fontSizeForHeight: number = plot.height * .35; // make font smaller as height gets smaller
  var fontSize: number = d3.min([fontSizeForWidth, fontSizeForHeight]);

  this.text
    .attr("x", plot.xOffset + (plot.width * 0.5))
    .attr("y", plot.yOffset + (plot.height * 0.5))
    .attr("width", plot.width)
    .attr("height", plot.height)
    .attr("font-size", fontSize);

}
```

g) Save your changes to **visuals.ts**.

You have now implemented the **Viz02** class as a custom visual that generated SVG graphic output using the D3 library. Next, you must modify the code inside **app.ts** to integrate the **Viz02** class into the application visual loading scheme. After you make a few minor modifications to **app.ts**, you will be able to test out the **D3Lab** project by running it inside in the Visual Studio debugger to see how the **Viz02** class renders its output.

5. Modify **app.ts** to integrate the new custom visual class named **Viz02**.
   a) Inside **app.ts**, locate the variable named **visuals**.
   b) Update the array that is used to initialize the **visuals** variable by adding a new instance of the class **Viz02**.

```
module myApp {

  var leftNavCollapsed: boolean = true;
  var loadedVisual: ICustomVisual;

  var visuals: ICustomVisual[] = [
    new Viz01(), new Viz02()
  ];
```

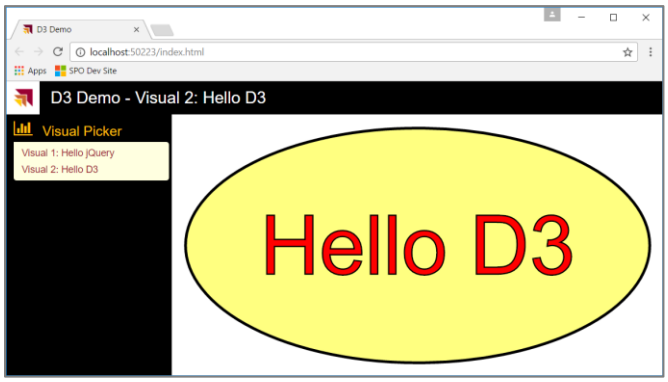
   c) Locate the code at the bottom of the document ready event handler that calls **LoadVisual** to automatically load a visual.

```
// automatically load first visual
LoadVisual(visuals[0]);
```

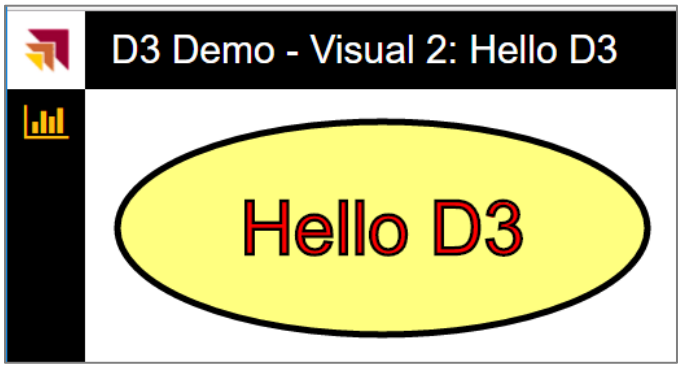
   d) Modify the parameter passed to **LoadVisual** to load the visual with the index of 1 instead of a

```
// automatically load first visual
LoadVisual(visuals[1]);
```

6. Test out the **D3Lab** project using the Visual Studio Debugger
   a) Press the **{F5}** key to start up the project in the Visual Studio debugger.
   b) When the project runs, you should see the visual implemented by the **Viz02** class matching the following screenshot.



   c) Collapse the left navigation menu and experiment by resize the window and watch the custom visual resize accordingly.



You have now programmed a custom visual using the D3 library and you added the code to resize the visual when its container is resized. In the next exercise you will continue your journey into D3 programming by binding a dataset to a set of SVG images to create a bar chart.

Exercise 4: Create a Bar Chart Visual using the D3 Library

In this exercise you will create a new class named **Viz03** that implements the **ICustomVisual** interface to generate a bar chart from a dataset which consists of an array of integer values. This will give you a chance to become familiar with the data binding techniques that are often used with D3 programming.

1. Create a new class named **Viz03** which implements the **ICustomVisual** interface.
   a) Add a new class named **Viz03** inside **visuals.ts** below the **Viz02** class definition using the following starter code.

```
export class Viz03 implements ICustomVisual {

  name = "Visual 3: Bar Chart";

  load(container: HTMLElement) { }

  update(viewport: IViewPort) { }

}
```

2. Add a set of private properties to the **Viz02** class implementation.
   a) Add a new private property named **dataset** and set its type to **number[]**.
   b) Initialize the dataset property with an array or integer values as shown in the following code listing.

```
name = "Visual 3 - Bar Chart";

private dataset: number[] = [440, 290, 340, 330, 400, 512, 368, 412];
```

   c) Place the cursor underneath the dataset property so you can add a few more properties.
   d) Add a new private property named **svgRoot** and set its type to **d3.Selection<SVGElementInstance>**.
   e) Add a new private property named **bars** and set its type to **d3.Selection<number>**.
   f) Add a new private property named **padding** with a type of **number** and a value of **12**.

```
private svgRoot: d3.Selection<SVGElementInstance>;
private bars: d3.Selection<number>;
private padding: number = 12;
```

3. Implement the **load** method of the **Viz03** class.
   a) Begin by appending a new **svg** element to the HTML container element and assigning a reference to the **svgRoot** property.

```
load(container: HTMLElement) {
  this.svgRoot = d3.select(container).append("svg");
}
```

   b) Below the code which appends a new **svg** element, add the following code to append a set of SVG **rect** elements into the **svgRoot** element. This code creates a separate **rect** element for each integer value in the array behind the **dataset** property.

```
this.bars = this.svgRoot
  .selectAll("rect")
  .data(this.dataset)
  .enter()
  .append("rect");
```

   c) The code you have written in the load method should match the following code listing.

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.bars = this.svgRoot
    .selectAll("rect")
    .data(this.dataset)
    .enter()
    .append("rect");

}
```

4. Implement the **update** method of the **Viz03** class.

   a) Begin by resizing the width and height of the **svgRoot** element to the width and height of the incoming **viewport** parameter.

```
public update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

}
```

   b) Next, create a new variable named **plot** and initialize it with new object that has 4 properties named **xOffset**, **yOffset**, **width** and **height** as shown in the following code listing.

```
var plot = {
  xOffset: this.padding,
  yOffset: this.padding,
  width: viewport.width - (this.padding * 2),
  height: viewport.height - (this.padding * 2),
};
```

   c) Next, add a variable named **datasetSize** to track the length of the arrays in the private **dataset** field.

```
var datasetSize = this.dataset.length;
```

   d) Add two more variables named **xScaleFactor** and **yScaleFactor** and initialize their values using the following code.

```
var xScaleFactor = plot.width / datasetSize;
var yScaleFactor = plot.height / d3.max(this.dataset);
```

   e) Add another variable named **barWidth** and initialize its value using the following code.

```
var barWidth = (plot.width / datasetSize) * 0.95;
```

   f) Add the following code to update the **x**, **y**, **width**, **height** and **fill** property of each bar in the bar chart.

```
this.bars
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - (Number(d) * yScaleFactor); })
  .attr("width", (d, i) => { return barWidth; })
  .attr("height", (d, i) => { return (Number(d) * yScaleFactor); })
  .attr("fill", "teal");
```

   g) At this point, your implementation of **update** should match the following code listing.

```
update(viewport: IViewPort) {

  this.svgRoot
    .attr("width", viewport.width)
    .attr("height", viewport.height);

  var plot = {
    xOffset: this.padding,
    yOffset: this.padding,
    width: viewport.width - (this.padding * 2),
    height: viewport.height - (this.padding * 2),
  };

  var datasetSize = this.dataset.length;
  var xScaleFactor = plot.width / datasetSize;
  var yScaleFactor = plot.height / d3.max(this.dataset);
  var barWidth = (plot.width / datasetSize) * 0.92;

  this.bars
    .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)); })
    .attr("y", (d, i) => { return plot.yOffset + plot.height - (Number(d) * yScaleFactor); })
    .attr("width", (d, i) => { return barWidth; })
    .attr("height", (d, i) => { return (Number(d) * yScaleFactor); })
    .attr("fill", "teal");
}
```

h) Save your changes to **visuals.ts**.

You have now implemented the **Viz03** class as a custom visual that generated SVG graphic output using the D3 library. Next, you must modify the code inside **app.ts** to integrate the **Viz03** class into the application visual loading scheme so you will be able to test out your new bar chart visual by running the **D3Lab** project inside in the Visual Studio debugger.

5. Modify **app.ts** to integrate the new custom visual class named **Viz03**.
   a) Inside **app.ts**, locate the variable named **visuals**.
   b) Update the array that is used to initialize the **visuals** variable by adding a new instance of the class **Viz03**.

```
module myApp {

  var leftNavCollapsed: boolean = true;
  var loadedVisual: ICustomVisual;

  var visuals: ICustomVisual[] = [
    new Viz01(), new Viz02(), new Viz03()
  ];
```

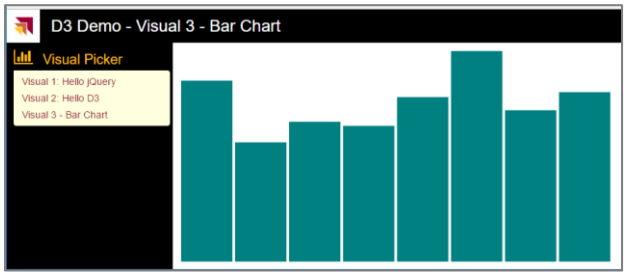
   c) Locate the code at the bottom of the document ready event handler that calls **LoadVisual** to automatically load a visual.

```
// automatically load first visual
LoadVisual(visuals[1]);
```

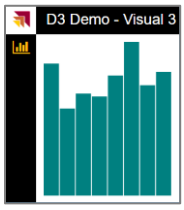
   d) Modify the parameter passed to **LoadVisual** to load the visual with the index of 1 instead of a

```
// automatically load first visual
LoadVisual(visuals[2]);
```

6. Test out the **D3Lab** project using the Visual Studio Debugger
   a) Press the **{F5}** key to start up the project in the Visual Studio debugger.
   b) When the project runs, you should see the visual implemented by the **Viz03** class matching the following screenshot.



   c) Collapse the left navigation menu and resize the window to test out how the visual adjusts to a new size.



7. Add text labels to the bars of the bar chart.
   a) Return to **visuals.ts** and the **Viz03** class definition.
   b) Add new private property named **labels**.

```
private svgRoot: d3.Selection<SVGElementInstance>;
private bars: d3.Selection<number>;
private labels: d3.Selection<number>;
private padding: number = 12;
```

c) Add the following code to the bottom of the **load** method.

```
this.labels = d3.select("svg").selectAll("text")
  .data(this.dataset)
  .enter()
  .append("text");
```

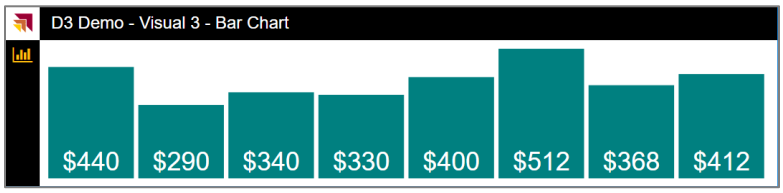
d) Add the following code to the bottom of the **update** method.

```
var yTextOffset = (d3.min(this.dataset) * yScaleFactor) * 0.2;
var textSize = (barWidth * 0.3) + "px";

this.labels.text((d, i) => { return "$" + d; })
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)) + (barWidth / 2); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - yTextOffset; })
  .attr("fill", "white")
  .attr("font-size", textSize)
  .attr("text-anchor", "middle")
  .attr("alignment-baseline", "middle");
```

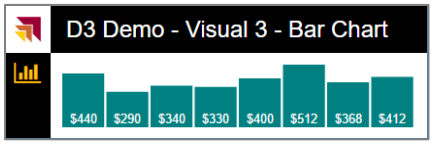
8. Test out the **D3Lab** project using the Visual Studio Debugger

a) Press the **{F5}** key to start up the project in the Visual Studio debugger.

b) When the project runs, you should see the visual implemented by the **Viz03** class matching the following screenshot.



c) Resize the window and verify that the font size changes as the window gets larger and smaller.

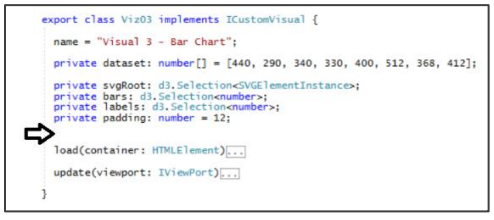


Exercise 5: Extend Your Bar Chart by Adding a Scale and a Y Axis

In this exercise you will extend the bar chart visual class named **viz03** you created in the previous exercise by adding a scale and a Y axis to the left hand side of the bars.

1. Add the private fields to create a scale and an Y axis for the vertical dimension.

a) Inside the class definition for **Viz03**, place the cursor after the existing private fields but before the **load** method.



b) Copy and paste the following code to add the private fields you will need to implement the new scale and Y axis.

```
private plotArea: d3.Selection<SVGElementInstance>;
private axisGroup: d3.Selection<SVGElementInstance>;
private xAxisOffset: number = 50;
private yScale: d3.scale.Linear<number, number>;
private yAxis: d3.svg.Axis;
```

c) Move into the implementation of the load method locate the first line which appends the **svgRoot** element.

```
this.svgRoot = d3.select(container).append("svg");
```

d) Place the cursor after the first line which appends the **svgRoot** element and add the following code to create a new SVG rectangle to provide a light yellow background behind the chart plotting area.

```
this.plotArea = this.svgRoot.append("rect")
  .attr("fill", "lightyellow")
  .attr("stroke", "black")
  .attr("stroke-width", 1);
```

e) Next, move down and locate the following code.

```
this.bars = this.svgRoot
  .selectAll("rect")
  .data(this.dataset)
  .enter()
  .append("rect");
```

f) Modify the first line by adding **append("g")** after the **svgRoot** as shown in the following code listing.

```
this.bars = this.svgRoot.append("g")
  .selectAll("rect")
  .data(this.dataset)
  .enter()
  .append("rect");
```

What's the purpose of adding **append("g")** after the **svgRoot** element to append a new SVG group element? This is required due to the fact that a new SVG rectangle has been added for the plot area background. The SVG rectangle elements in the bar chart must be isolated for all other rectangles to ensure that the D3 data binding works correctly. This extra code ensures that the SVG rectangle elements for the bar chart are created in their own isolated group.

g) Move down in the **load** method and locate the code that initializes the **labels** field. This code does not need to be modified.

h) Place your cursor at the end of the **load** method and add the following code to initialize a new scale object and an axis object.

```
this.axisGroup = this.svgRoot.append("g");
this.yScale = d3.scale.linear();
this.yAxis = d3.svg.axis();
```

i) At this point, your implementation of **load** should match the following code listing.

```
load(container: HTMLElement) {

  this.svgRoot = d3.select(container).append("svg");

  this.plotArea = this.svgRoot.append("rect")
    .attr("fill", "lightyellow")
    .attr("stroke", "black")
    .attr("stroke-width", 1);

  this.bars = this.svgRoot.append("g")
    .selectAll("rect")
    .data(this.dataset)
    .enter()
    .append("rect");

  this.labels = d3.select("svg").selectAll("text")
    .data(this.dataset)
    .enter()
    .append("text")

  this.axisGroup = this.svgRoot.append("g");
  this.yScale = d3.scale.linear();
  this.yAxis = d3.svg.axis();

}
```

j) Move into the implementation of the **update** method and locate the code that initializes the **plot** variable.

```
var plot = {
  xOffset: this.padding,
  yOffset: this.padding,
  width: viewport.width - (this.padding * 2),
  height: viewport.height - (this.padding * 2),
};
```

k) Modify this code by adding the **xAxisOffset** value to the **xOffset** property and subtracting the **xAxisOffset** value from the **height** property.

```
var plot = {
  xOffset: this.padding + this.xAxisOffset,
  yOffset: this.padding,
  width: viewport.width - (this.padding * 2) - this.xAxisOffset,
  height: viewport.height - (this.padding * 2),
};
```

l) Move down a few lines and comment out the line of code which defines the **yScaleFactor** variable.

```
var datasetSize = this.dataset.length;
var xScaleFactor = plot.width / datasetSize;
// var yScaleFactor = plot.height / d3.max(this.dataset);
var barWidth = (plot.width / datasetSize) * 0.92;
```

m) Below the **barWidth** variable, create a new variable named **barXStart**.

```
var barWidth = (plot.width / datasetSize) * 0.92;
var barXStart = (plot.width / datasetSize) * 0.04
```

n) Below the **barXStart** variable, add the following code to assign the domain and the range of the **yScale** field.

```
var yDomainStart: number = d3.max(this.dataset) * 1.05;
var yDomainStop: number = 0;
var yRangeStart: number = 0;
var yRangeStop: number = plot.height;

this.yScale
  .domain([yDomainStart, yDomainStop])
  .range([yRangeStart, yRangeStop]);
```

o) After the code that assigns the domain and the range of the **yScale** field, add the following code to resize the background rectangle for the plot area.

```
this.plotArea
  .attr("x", plot.xOffset)
  .attr("y", plot.yOffset)
  .attr("width", plot.width)
  .attr("height", plot.height);
```

p) Move down and locate the code that accesses the **bars** field to resize the bars of the bar chart.

```
this.bars
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - (Number(d) * yScaleFactor); })
  .attr("width", (d, i) => { return barWidth; })
  .attr("height", (d, i) => { return (Number(d) * yScaleFactor); })
  .attr("fill", "teal");
```

You will now modify the code that uses the **yScaleFactor** variable to use D3 scale object referenced by the **yScale** field instead.

q) Modify the line of code update the **y** attribute with the following code which uses the **yScale** field.

```
.attr("y", (d, i) => { return plot.yOffset + this.yScale(Number(d)); })
```

r) Modify the line of code update the **height** attribute with the following code which uses the **yScale** field.

```
.attr("height", (d, i) => { return (plot.height - this.yScale(Number(d))); })
```

s) Your code to update the attributes of the **bars** field should now match the following code listing.

```
this.bars
  .attr("x", (d, i) => { return plot.xOffset + barXStart + (i * (xScaleFactor)); })
  .attr("y", (d, i) => { return plot.yOffset + this.yScale(Number(d)); })
  .attr("width", (d, i) => { return barWidth; })
  .attr("height", (d, i) => { return (plot.height - this.yScale(Number(d))); })
  .attr("fill", "teal");
```

t) Comment out the code that declares the **yTextOffset** variable and replace it with code the following code.

```
//var yTextOffset = (d3.min(this.dataset) * yScaleFactor) * 0.2;
var yTextOffset = this.yScale(d3.min(this.dataset)) * 0.5;
```

u) Next, add a new variable named **textSize** and initialize it with the following code.

```
var textSize = (barWidth * 0.3) + "px";
```

v) The existing code that calls the **text** method on the **labels** field is fine as it is and does not need to be modified.

```
this.labels.text((d, i) => { return "$" + d; })
  .attr("x", (d, i) => { return plot.xOffset + (i * (xScaleFactor)) + (barWidth / 2); })
  .attr("y", (d, i) => { return plot.yOffset + plot.height - yTextOffset; })
  .attr("fill", "white")
  .attr("font-size", textSize)
  .attr("text-anchor", "middle")
  .attr("alignment-baseline", "middle");
```

w) Place your cursor at the end of the **update** method and add the following code.

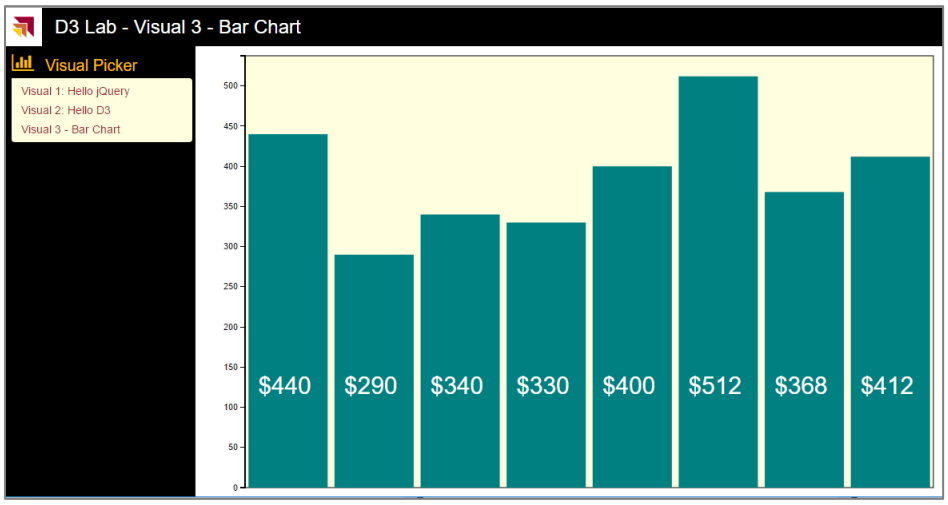
```
this.yAxis
  .scale(this.yScale)
  .orient('left')
  .ticks(10);

var transform = "translate(" + (this.padding + this.xAxisOffset) + "," + this.padding + ")";

this.axisGroup
  .attr("class", "axis")
  .call(this.yAxis)
  .attr({ 'transform': transform });
```

x) Save your changes to **visuals.ts**.

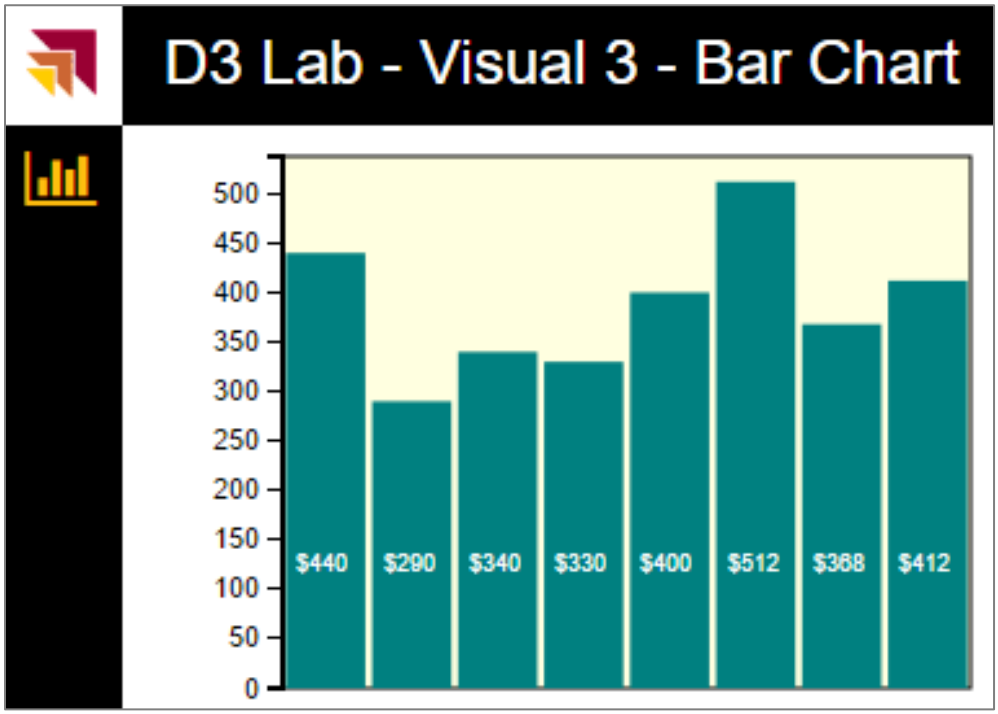
2. Test out the **D3Lab** project using the Visual Studio Debugger

a) Press the **{F5}** key to start up the project in the Visual Studio debugger.

b) When the project runs, you should see the visual implemented by the **Viz03** class matching the following screenshot.



c) Experiment by resizing the window and observing how the axis scales in a horizontal and a vertical dimension.



Congratulations. You have made it to the end of this lab.