

# Practical Highcharts with Angular

Your Essential Guide to Creating  
Real-time Dashboards

—  
Sourabh Mishra

Apress®

[www.allitebooks.com](http://www.allitebooks.com)

# Practical Highcharts with Angular

Your Essential Guide  
to Creating Real-time  
Dashboards

Sourabh Mishra

Apress®

# ***Practical Highcharts with Angular***

Sourabh Mishra  
IECE Digital, Bangalore, India

ISBN-13 (pbk): 978-1-4842-5743-2

ISBN-13 (electronic): 978-1-4842-5744-9

<https://doi.org/10.1007/978-1-4842-5744-9>

Copyright © 2020 by Sourabh Mishra

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Louise Corrigan  
Development Editor: James Markham  
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [rights@apress.com](mailto:rights@apress.com), or visit [www.apress.com/rights-permissions](http://www.apress.com/rights-permissions).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at [www.apress.com/bulk-sales](http://www.apress.com/bulk-sales).

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484257432](http://www.apress.com/9781484257432). For more detailed information, please visit [www.apress.com/source-code](http://www.apress.com/source-code).

Printed on acid-free paper

*This book is dedicated to  
My Baba, Amma, Pappa, Mummy and Little Sister.*

# Table of Contents

<b>About the Author .....</b>	<b>ix</b>
<b>About the Technical Reviewer .....</b>	<b>xi</b>
<b>Acknowledgments .....</b>	<b>xiii</b>
<b>Introduction .....</b>	<b>xv</b>
<b>Chapter 1: Getting Started with Highcharts.....</b>	<b>1</b>
Benefits of Highcharts .....	2
History of Highcharts .....	3
Basics of Charting.....	3
Setup and Configuration .....	6
Creating Your First Chart.....	8
Summary.....	14
<b>Chapter 2: Concept of Highcharts.....</b>	<b>15</b>
Scalable Vector Graphics .....	15
Choosing the Right Chart Type Based on Requirements.....	16
Bar Charts.....	17
Line Charts .....	18
Scatter Plots .....	18
Maps.....	19
Setting Layouts .....	20
Alignment .....	21
Setting Up Chart Margins .....	22
Legends .....	22

## TABLE OF CONTENTS

Setting Up Plot Lines.....	23
Setting Credits .....	25
Summary.....	25
<b>Chapter 3: Integrating Highcharts with Angular .....</b>	<b>27</b>
What Is Angular? .....	27
Configuring Angular .....	28
Setting Up Node.js .....	28
Code Editor .....	31
Setting Up Angular CLI.....	32
TypeScript .....	36
Highcharts Angular Wrapper.....	36
Summary.....	45
<b>Chapter 4: Different Charting Types.....</b>	<b>47</b>
Pie Charts.....	48
Donut Chart.....	54
Drilldown Charts .....	57
Required Dependencies .....	57
Setting Up the Unique Name for a Series .....	57
Line Charts.....	63
Area Charts .....	65
Scatter Charts .....	72
Histogram Charts .....	76
Heat Map Series Charts .....	79
Stacked Bar Charts .....	82
Column Pyramid Charts .....	86
Gauge Charts .....	92
Summary.....	99

<b>Chapter 5: Working with Real-Time Data .....</b>	<b>101</b>
Web API .....	101
What Is REST? .....	103
Web API Development Using Visual Studio .....	104
Solution Explorer .....	106
ConfigureService() .....	108
Configure() .....	109
Routing .....	110
Attribute Routing .....	110
Database Creation .....	115
Adding Entity Framework .....	119
Angular-Highcharts UI Application .....	125
Services in Angular .....	125
Events in Highcharts .....	140
Drilldown Event .....	144
LegendItem Click Event .....	152
CheckBoxClick Event .....	155
Highcharts Wrapper for .NET .....	160
LineSeries Chart with a Highcharts Wrapper .....	160
Gauge Series Chart with a Highcharts Wrapper .....	166
SeriesData Classes .....	172
Summary .....	173
<b>Chapter 6: Themes and Additional Features of Highcharts .....</b>	<b>175</b>
Themes in Highcharts .....	175
Applying a Dash Style Series to a Line Chart .....	178
Combinations in Highcharts .....	181
Zoom Option in Highcharts .....	190
Setting an Image in a Chart Area .....	191

## TABLE OF CONTENTS

3D Charts .....	193
Cylinder Chart.....	198
Funnel 3D .....	201
Pyramid 3D .....	205
Pie 3D Chart.....	209
Exporting and Printing Charts .....	214
Additional Chart Features .....	216
Radar Chart .....	216
Pareto Chart.....	219
Bell Curve Chart.....	225
Organization Chart.....	227
Timeline Chart .....	231
Gantt Chart .....	234
Summary.....	238
<b>Chapter 7: Building a Real-Time Dashboard.....</b>	<b>239</b>
Real-Time Dashboard Application.....	239
Features of the App .....	240
Creating a Web API .....	241
Setting Up a Database.....	247
Creating a Database First Approach Using Entity Framework.....	250
Routing in an Angular App .....	265
Summary.....	294
<b>Index.....</b>	<b>295</b>



# About the Author



**Sourabh Mishra** is an entrepreneur, developer, speaker, author, corporate trainer, and animator. He is a Microsoft guy; he is very passionate about Microsoft technologies and is a true .NET warrior. Sourabh started his career when he was just 15 years old. He's loved computers from childhood. His programming experience includes C/C++,

ASP.NET, C#, VB.NET, WCF, SQL Server, Entity Framework, MVC, web API, Azure, jQuery, Highcharts, and Angular. Sourabh has been awarded Most Valuable Professional (MVP) status. He has a zeal to learn new technologies and share his knowledge on several online community forums.

He is the founder of IECE Digital and Sourabh Mishra Notes, an online knowledge sharing platform where one can learn new technologies very easily and comfortably.

He can be reached at

**Website:** [www.sourabhmishranotes.com](http://www.sourabhmishranotes.com)

**YouTube:** sourabhmishranotes

**Twitter:** sourabh\_mishra1

**Facebook:** facebook.com/sourabhmishranotes

**Instagram:** sourabhmishranotes

**Email:** sourabh\_mishra1@hotmail.com

*Investment in knowledge is the biggest investment  
and sharing knowledge is the biggest service to the society.*

—Vinay Bharti Jain

# About the Technical Reviewer

**Kenneth Fukizi** is a software engineer, architect, and consultant with experience in coding on different platforms internationally. Prior to dedicated software development, he worked as a lecturer for a year and was then head of IT at different organizations. He has domain experience working with technology for companies in a wide variety of sectors. When he's not working, he likes reading up on emerging technologies and strives to be an active member of the software community.

# Acknowledgments

*Practical Highcharts with Angular* has been a very special project, brought together through the efforts of very special people in my life. I am deeply thankful to the Apress team and to all those whose enthusiasm and energy transformed my vision to bring this book into reality, especially my family. The commitment and sense of mission moves me to the next level.

I express my special thanks to

- My wonderful parents, Shailendra Mishra and Saroj Mishra, who have supported me in every phase of my life and guided me from day one and gave me chance to work on computers in my childhood. My parents taught me to take challenges in life and come out successfully.
- My lovely and wise sister, Surbhee Mishra, a great content writer, for her encouragement and support at every step.
- My uncle, Vinay Bharti Jain, who has always stood with me like a shield and guided me very well in critical decision making from the start of my career. My aunt, Poonam Jain, who has always believed that one day I will change the world through my knowledge.
- Naveen Verma, a great software architect and my teacher, who taught me how to write good code and how to use the right weapon at the right place in the world of software development.

## ACKNOWLEDGMENTS

- John Ebenezer, a wonderful human being and a true leader, who knows the art of people management. I learned from him how to deal with business people and get the best from the team.
- Welmoed Spahr and the entire Apress team for immediately evaluating the potential of this book and for believe in me and for making this book a reality. I sincerely value her guidance.
- Louise Corrigan, Nancy Chen, and James Markham for having faith in me, for bringing out this book, and for giving support, help, and guidance at every step during the entire journey of writing this book.
- My millions of readers across the globe who have encouraged me to write technical blogs and have given their love and affection, and also the people who are reading this book right now.
- Last but not least, with the deepest gratitude I wish to thank every person who has come into my life and has inspired, touched, and illuminated me through their presence.

Happy Reading!

# Introduction

First of all, thank you for picking up this book. Whether you are standing in a bookshop or reading this at your office or at home, I assume that you probably have a strong interest in developing stunning and interactive dashboards for your web product.

Highcharts is a new age tool for developing an interactive dashboard for your web products. You can easily define and use your data collection and get stunning graphs based on your requirements. Nowadays, charting is used in finance, education, entertainment, sports, and real estate sectors to analyze data. Highcharts is built on top of modern JavaScript frameworks like jQuery and Angular. Highcharts enables developers to easily construct charts that will work in all modern browsers with pure knowledge of HTML, CSS, and JavaScript.

## Who Should Read This Book

*Practical Highcharts with Angular* is a book mainly for developers. In this book, developers will learn step by step how to create client-side and server-side applications with the use of Angular with Highcharts and a REST-based API.

## Organization of This Book

Each chapter in this book has been developed to highlight all the good features of Highcharts. The following is brief summary of each chapter:

- **Chapter 1** gives you an introduction to Highcharts. If you are new to Highcharts, this is the place to start. You will learn the basics of charting, how to set up and install Highcharts for your application, and how easily you can construct your first chart.
- **Chapter 2** talks about Scalable Vector Graphics and how to choose the right chart based on your requirements, because choosing the right chart and setting the layout and legends is an art.
- **Chapter 3** is the base for the rest of the chapters because here you will learn to develop an Angular app from the beginning. You will also learn the basics of Angular and how easily you can develop interactive charts using Highcharts.
- **Chapter 4** shows how to develop some advanced charts with Angular and Highcharts, such as drilldowns, histograms, heatmaps, gauges, stacked bars, and more.
- **Chapter 5** shows how to get real-time data from the server side using REST-based services and how easily you can develop a client-side app using Angular and Highcharts.

- **Chapter 6** teaches you how to apply themes and layouts to a chart so it looks stunning and interactive. You also learn some advanced concept of Highcharts like 3D, exporting in different formats, Pareto charts, combined charts, and more.
- **Chapter 7** shows project-based learning. In this chapter, you will develop a learning project, which shows how to develop a stunning and interactive dashboard with multiple charts. Here, you get live historical data from the stock market using a REST API and then you develop a dashboard based on a portfolio.

## How to Contact the Author

The author can be contacted as follows:

- Website: [www.sourabhmishranotes.com](http://www.sourabhmishranotes.com)
- YouTube: sourabhmishranotes
- Twitter: sourabh\_mishra1
- Facebook: facebook.com/sourabhmishranotes
- Instagram: sourabhmishranotes
- Email: sourabh\_mishra1@hotmail.com



## CHAPTER 1

# Getting Started with Highcharts

Highcharts is a JavaScript-based library. With the use of it, you can develop professional, high-quality, animated, web-based charting with minimal coding. Highcharts provides very simple built-in options that are easy to learn and easy to use; you just have to input data based on your data collection and it will give you charts based on your requirements.

Highcharts provides fast rendering and quick-to-deliver products. You can think out of the box and develop your charting very easily. Highcharts lets you call your services and use it with all modern JavaScript frameworks like Angular and jQuery. You can export your charts into images, CSV files, or Excel files very easily. These built-in options are available at the time of development.

In this chapter, you are going to learn how to configure Highcharts into your web application. In the next part, you will learn how to implement charts very quickly.

Highcharts is built in such a way that all you have to do is input a collection of data and Highcharts will professionally render a chart for you.

## Benefits of Highcharts

Highcharts is an excellent product for building charting for real-time applications. It provides the following rich benefits:

- It's easy to learn and easy to use. All you need is some knowledge of HTML, CSS, and JavaScript and you can develop your charts.
- It works in all modern browsers.
- It works in modern JavaScript libraries like Angular, Vuejs, Reactjs, and jQuery.
- You can export charts in various formats. Highcharts provides different charting types like line, bar, column, map, area, plot, stock, box, heat map, tree map, funnel, and scatter plot.
- It's an excellent tool for developing a real-time informative dashboard for your application.
- **Licenses:** Highcharts provides two types of licenses:
  1. **Non-commercial license:**

This type of licensing is for non-profit purposes and personal use.
  2. **Commercial license:**

This is for commercial purposes, such as an organization building products for commercial-level use.

You can go for a single website, developer license, High-five license.

A single website license is for traditional websites. The developer license is for web apps and SaaS projects, and it comes in single dev, five dev, ten dev, etc.

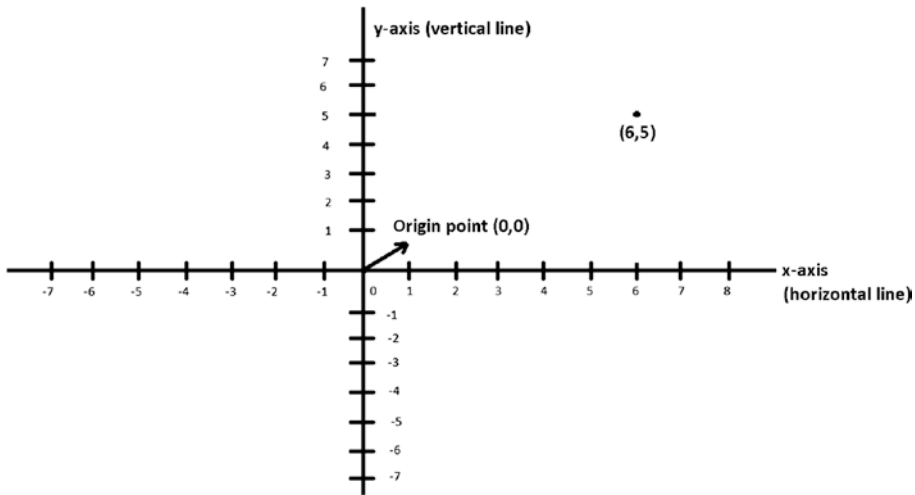
## History of Highcharts

Back in 2003, charting was not an easy option. People were doing charting with the use of an HTML image or Java applet and servlet or Flash-based animated graphics charts. These products ruled the market. In 2009, Highsoft, a Norwegian-based company founded by Torstein Hønsi, developed and introduced a JavaScript-based framework to easily plug into enterprise products to generate world-class, stunning graphs based on your requirements.

## Basics of Charting

A graph is a way to represent relationships between two or more related data. Every graph has two lines, vertical and horizontal (Figure 1-1).

A horizontal graph line is called the *x-axis*, and the vertical line is called *y-axis*. The point where these lines intersect each other is called the *origin point*. In the origin point, in the x-axis, the right side of the origin uses positive numbers and the left side uses negative numbers. The same thing happens with the y-axis: the value on the top side of origin is positive and the downside is negative. The origin point value is always 0.



**Figure 1-1.** A simple graph presentation for the x-axis and y-axis with data

Figure 1-1 clearly shows the x-axis (horizontal) and y-axis (vertical). The right-hand side of the axis is positive; the left-hand side is negative. You can see the same for the y-axis.

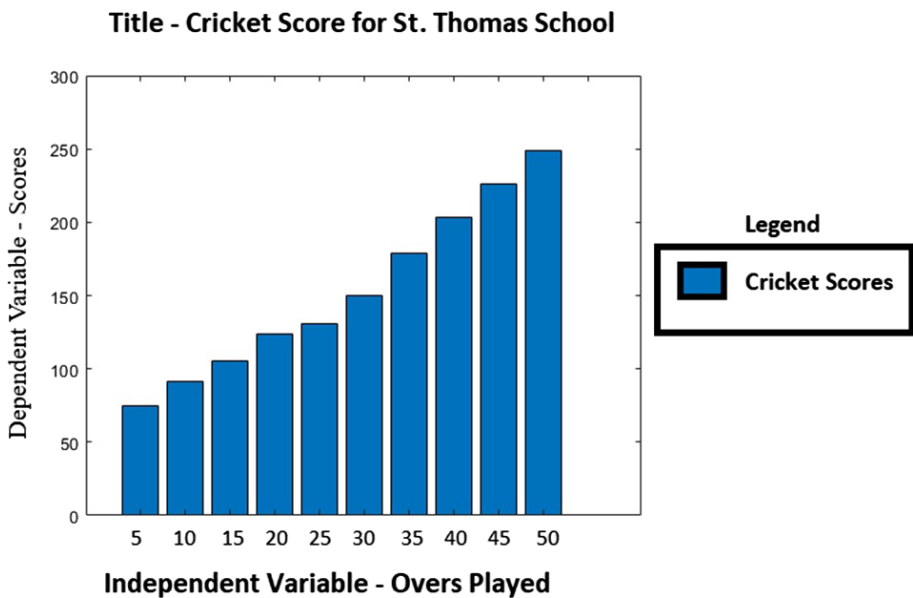
Whenever you want to connect values of the x-axis and the y-axis, this point is called a *coordinate point*. In Figure 1-1, the value of x is 6 and y is 5, so the x and y coordinate is (6,5).

The following are the essential parts of every graph:

- **Title:** This describes what the graph is about.
- **Independent variable:** This part is defined by the x-axis. It usually indicates things like subject name, cricket overs, temperatures, etc.
- **Dependent variable:** This part is defined by the y-axis. This part is connected with an independent variable, and it will show you the result because of the value of the independent variable, such as marks in an examination, cricket runs, etc.

- **Scales:** This part decides where to plot points, which represent data. The scale always starts with 0 and increases with intervals, such as 3, 6, 9, 12, 15. It depends on data values.
- **Legends:** This is a short description of the graph's data.

Figure 1-2 shows a score by the St. Thomas School cricket team, including the title, independent variable, dependent variable, and legends.



**Figure 1-2.** A simple bar graph showing the essential parts of a chart

It's now time to configure Highcharts into your web application and then quickly implement it.

## Setup and Configuration

The installation of Highcharts into a web application is straightforward. You can configure and install Highcharts in three ways into your web application.

1. **CDN (content delivery network):** If you want to implement Highcharts with jQuery, you can use the CDN.

**Example:**

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js">
</script>
<script src="https://code.highcharts.com/highcharts.js">
</script>
```

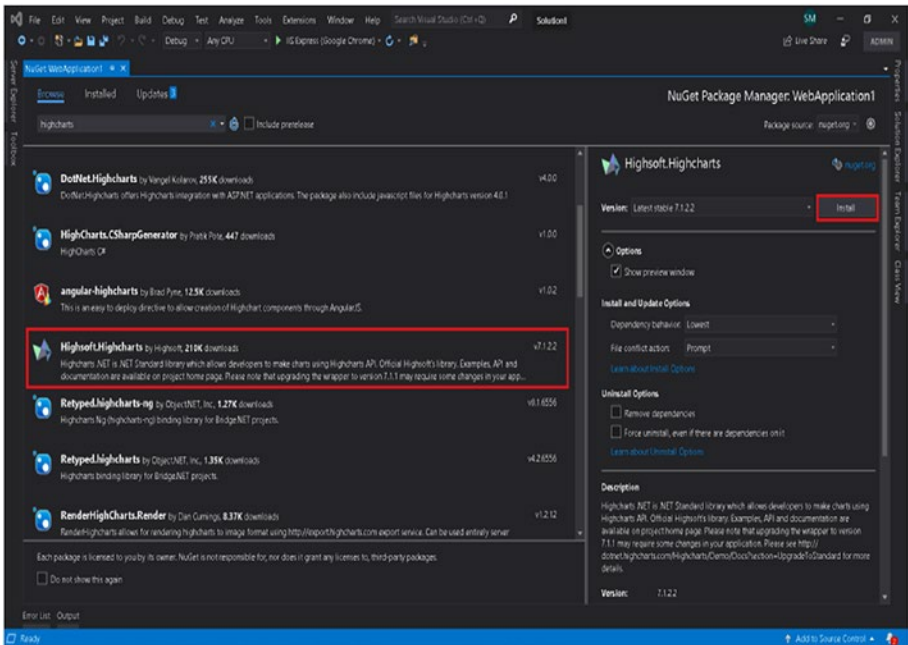
2. **Download the Highcharts.js file:** For this method, go to [www.highcharts.com](http://www.highcharts.com), open the download section, and download the latest zip file for Highcharts. Unzip and add a folder into your project file system.

**Example:** Add the following code based on your path to a file:

```
<script src="code/highcharts.js"></script>
```

The benefit with this method is that, without the Internet, you can run your project in a localhost environment.

3. **With a NuGet package:** If you are developing your project in Visual Studio or Visual Studio code, you can download the NuGet package. Here are the steps: right-click the project and select “Manage NuGet package” from the menu. You will get a dialog box. Click the Browse tab, type “Highcharts,” and press Enter. See Figure 1-3.



**Figure 1-3.** Installing Highcharts from the NuGet package

As seen in Figure 1-3, you get a list in the Browse tab. In this list, select Highsoft. Highchart and click the Install button.

Click the OK button and select the I Accept button (Figure 1-4).



**Figure 1-4.** *Accepting a license for Highcharts from NuGet*

Now click Reference from Solution Explorer and you will get Highcharts added into a reference part.

Now you know the three ways to add Highcharts into your project.

## Creating Your First Chart

Now it's time to do some hands-on with Highcharts. All Highcharts graphs mostly use the same configuration. Here are some significant properties that are always required in order to build Highcharts:

- **chart:** This property applies to the top-level setting.  
Example: The type of graph, where to render it in the page, layout of the chart, animations, etc.



- `title/subtitle`: For the chart title and subtitle
- `xAxis/yAxis`: For properties like category, title, CSS style, interval, etc. for an axis
- `series`: For configuring the data collection to show in the graph, where you can set single or multiple series data

For the first example, you will learn how to generate a column graph. Listing 1-1 is an HTML file that you are going to add jQuery and Highcharts CDN, and then render a column-type graph into an HTML `<div>`.

**Listing 1-1.** Index.html

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html;
   charset=utf-8">
5. <meta name="viewport" content="width=device-width,
   initial-scale=1">
6. <title>Highcharts Example</title>
7. <script src="https://code.jquery.com/jquery-3.4.1.min.js">
   </script>
8. <script src="https://code.highcharts.com/highcharts.js">
   </script>
9. </head>
10. <body>
11. <div id="container" style="min-width: 310px; height: 400px;
    margin: 0 auto"></div>
12. <script type="text/javascript">
13. var charts = new Highcharts.Chart({
14. chart: {

```

## CHAPTER 1 GETTING STARTED WITH HIGHCHARTS

```
15. renderTo: 'container',
16. type: 'column'
17. },
18. title: {
19. text: 'Monthly Sales Chart Department Wise'
20. },
21. subtitle: {
22. text: 'Year 2018'
23. },
24. xAxis: {
25. categories: [
26. 'Jan',
27. 'Feb',
28. 'Mar',
29. 'Apr',
30. 'May',
31. 'Jun',
32. 'Jul',
33. 'Aug',
34. 'Sep',
35. 'Oct',
36. 'Nov',
37. 'Dec'
38. ],
39. },
40. yAxis: {
41. min: 0,
42. title: {
43. text: 'Sales in Million $'
44. }
45. },
```

```

46. series: [{
47.   name: 'Marketing Department',
48.   data: [49.9, 51.5, 32.0, 82.0, 75.0, 66.0, 32.0, 25.0,
          35.4, 65.1, 58.6, 34.4]
49. },
50. {
51.   name: 'Computer Science Department',
52.   data: [40.5, 34.5, 84.4, 39.2, 23.2, 45.0, 55.6, 18.5,
          26.4, 14.1, 23.6, 84.4]
53. }]
54. });
55. </script>
56. </body>
57. </html>

```

If you run the above code, you will get the output shown in Figure 1-5.



**Figure 1-5.** Demo of your first bar chart

Now, let's take a closer look at the code. This code has three parts. In the first part, you add jQuery and Highcharts CDN into the <head> portion:

```

<script src="https://code.jquery.com/jquery-3.4.1.min.js">
</script>
<script src="https://code.highcharts.com/highcharts.js">
</script>

```

Next, in the `<body>` section, there is a `<div>`. The reason for creating this `<div>` is that Highcharts will render in it:

```
<div id="container" style="min-width: 310px; height: 400px; margin: 0 auto"></div>
```

The next part of the code is Highcharts and JavaScript code. Let's understand it line by line.

The following line creates a new Highcharts object, and this object will define all the required properties, which are helpful to render a graph into the browser:

```
var charts = new Highcharts.Chart
```

In this next line of code there is a property called `renderTo`, which indicates in the HTML page which particular `<div>` id you want to render this chart. The `type` property defines what type of graph you want to see, so in the `type` property, you can set like line bar, spline, etc.

```
var charts = new Highcharts.Chart({
  chart: {
    renderTo: 'container',
    type: 'column'
  },
```

In the next line, you set the title and subtitle, so once your graph has rendered, your users will see whatever title/subtitle you want to show:

```
title: {
  text: 'Monthly Sales Chart Department Wise'
},
subtitle: {
  text: 'Year 2018'
},
```

The next line is the `categories` property of the `xAxis`. It contains an array of labels for each data point.

```
xAxis: {  
  categories: [  
    'Jan',  
    'Feb',  
    'Mar',  
    'Apr',  
    'May',  
    'Jun',  
    'Jul',  
    'Aug',  
    'Sep',  
    'Oct',  
    'Nov',  
    'Dec'  
  ],  
}
```

Next is the `yAxis`. Here the `min` property is related to setting a minimum value for a chart, so if you set as 0, Highcharts will never set for negative numbers. So in the future, if any negative values comes into the series collection, the chart will not show negative chart data points. If you want to work with negatives values, you can set `min` as -50 (or whatever highest min value you have) or you can remove it. In the `yAxis` area you can see negative data points.

The `title` property is used to set the title for the `yAxis`.

```
yAxis: {  
  min: 0,  
  title: {
```

```
text: 'Sales in Million $'
  }
}
```

The next property is `series`, one of the most essential properties of Highcharts. First is `name` and it defines what type of data collection you are setting. This is also helpful for tooltips; when users hover their mouse pointer in a graph, it shows that this data point is related to a name. This is also helpful to set legends about the graph.

The `data` property refers to the collection of data in the form of an array, a series you can set as single or multiple. Later chapters will show you how to pass real-time data into a series section and render the chart.

```
series: [{
name: 'Marketing Department',
data: [49.9, 51.5, 32.0, 82.0, 75.0, 66.0, 32.0, 25.0, 35.4,
65.1, 58.6, 34.4]
},
{
name: 'Computer Science Department',
data: [40.5, 34.5, 84.4, 39.2, 23.2, 45.0, 55.6, 18.5, 26.4,
14.1, 23.6, 84.4]
}]
```

## Summary

Highcharts is a new age tool for developing an interactive dashboard for your web products. You can easily define your data collection and get stunning graphs based on your requirements. In this chapter, you saw the basics of Highcharts and how easy it is to set up Highcharts and create your first column-type chart.

## CHAPTER 2

# Concept of Highcharts

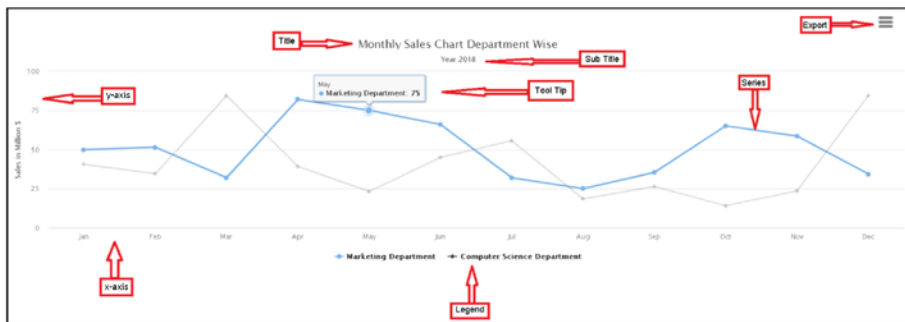
In this chapter, you are going to learn the basic concepts of Highcharts and how Highcharts works. To use Highcharts, you must understand what type of chart to use based on your requirements. Then I will discuss some essential properties of Highcharts. So, let's get into the second chapter.

## Scalable Vector Graphics

Scalable Vector Graphics (SVG) is an XML-based vector image format. Here *scalable* means that it can be resized up or down in any dimension; the user will not lose any quality. SVG is designed for two-dimensional graphics and supports interactive graphics and animation. The behavior of Scalable Vector Graphics is defined in an XML text file. The benefit with this is that Scalable Vector Graphics can be searched, scripted, indexed, and compressed very efficiently. SVG is supported in all modern browsers like Internet Explorer, Internet Edge, Google Chrome, Safari, Opera, and Firefox.

In 1999, the World Wide Consortium (W3C) developed SVG as the language of vector graphics. With the use of SVG, you can create shapes, example paths, and outlines consisting of lines and curves, text, and bitmap images. In SVG, you can apply CSS for styling and JavaScript for scripting.

For text, you can apply internationalization and localization for more accessibility. Highcharts is also SVG driven, and you can generate high-quality charts with interactive graphics and animations (Figure 2-1).



**Figure 2-1.** SVG-based line chart presentation using Highcharts

In Figure 2-1, the tooltip hovers and the first option you see is x-axis. In the x-axis, you draw months; the y-axis is where you populate values based on department revenue collection.

These are pretty basic things. Then you have the title and subtitle for the graph, and then you plot a different data series based on revenue generation of departments. So in the chart each one of the lines is a separate series. Then you have legends based on departments. There’s a tooltip, and you can change your tooltip style based on the requirements. You can export into an image, CSV, or PDF.

## Choosing the Right Chart Type Based on Requirements

Now let’s talk about how to choose the right chart type. It’s essential to understand the purpose of each kind of chart so that you can select the correct chart. Highcharts provides mainly four types of charts: bar charts, line charts, scatter plots, and maps. In later chapters, I will talk more about different charting types.



## Bar Charts

Bar charts are a chart type that represents categorical data with rectangular bars in a proportion of height and length. These bars can plot horizontally or vertically. In bar charts, one axis may represent the specific categories being compared and the other axis may represent measured values.

Bar chart can be arranged in any order. In a bar chart, you can represent multiple data. When you want to represent values from highest to lowest incidents, these types of charts called Pareto charts. These charts provide a visual/graphical representation of categorical data. You can define categories like the age group of students, year, month, animals, shoe size, etc.

In column bar charts, these categories come in the x-axis horizontal form, and the height of the graph will generate based on values defined vertically on the y-axis.

## When to Choose a Bar Chart

Bar charts are good for when you want to compare data based on categories, such as sales in a specific region, quarterly growth of a company, etc. (Figure 2-2).



**Figure 2-2.** Bar chart

## Line Charts

The line chart is also known as a line plot, line graph, or curve chart. It's a type of diagram that displays information in a series of data points. These data points are called markers, and these markers are connected by straight lines. The line chart is one of the standard charts used in many fields. The line chart is useful when you want to show a trend in data over an interval of time, or a time series; here the lines are drawn chronologically.

### When to Choose a Line Chart

Line charts are suitable for a time series when you want to represent data in the form of a graph over time. Here you can define trend lines, and there are lots of ways to represent data over time and to make it meaningful so people can understand where things are going (Figure 2-3).



**Figure 2-3.** Line chart

## Scatter Plots

A scatter plot, also known as a scatter graph, scattergram, or scatter chart, is a type of plot that uses Cartesian coordinate to display typically two variables for a set of data. Here points are coded and define in color shape/size. Data is presented in the collection of points, and each point has one value.

Scatter plots come in a position of x-axis and y-axis, respectively. These types of graphs can show distributions very interestingly. A scatter plot designs for various kinds of correlations between variables with a specific confidence interval. For example, for weight and height, the weight would be on the y-axis and height would be on the x-axis.

Suppose a university researcher is studying the capacity of lungs in a human body, specifically how long people can hold their breath. So lung capacity is the first variable and time is the second variable. Then the researcher can plot data into a scatter plot, assigning lung capacity to the horizontal axis and length of time to the vertical axis.

## Maps

Map charts allow you to represent your data on a geographical map. Here you can define the chart in two ways:

1. Graphical points
2. Geographical area

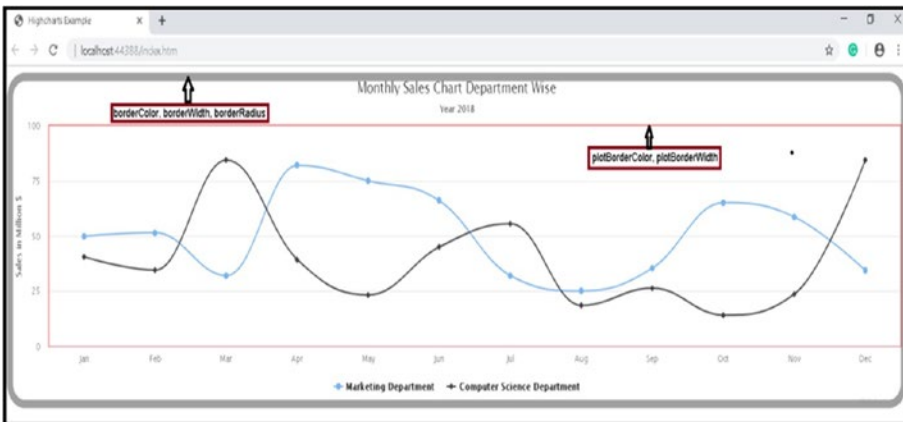
With geographic points, you can set your marks over geographical coordinates; these markers use color, shape, and size. The geographical area defines the colored area on the map. For example, an area could be a country, state, or city.

For example, in the world map, we can see the United States, India, and Africa in different colors; each country indicates values. This type of map chart where we color geographical areas is known as a choropleth. In Highcharts, you can define detailing over maps, so when you click on the country you can see the states of a nation and once you click on states, you can see the cities. In later chapters, I will discuss maps in detail.

## Setting Layouts

To set up the layout in Highcharts, the first step is to set a border around the plot area. For this you have five properties: `plotBorderWidth`, `plotBorderColor`, `borderColor`, `borderWidth`, and `border-radius` in the chart section (Figure 2-4).

```
chart: {
  renderTo: 'container',
  type: 'spline',
  plotBorderColor: 'red',
  plotBorderWidth: 1,
  borderColor: 'grey',
  borderWidth: 10,
  borderRadius: 25
},
```



**Figure 2-4.** Spline chart with a border layout

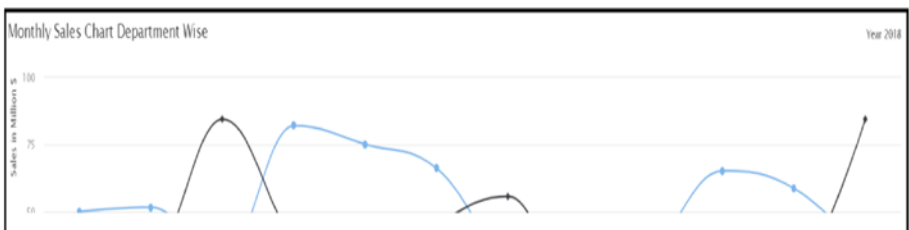
## Alignment

In Highcharts, you can set the alignment for labels such as title, subtitle, xAxis.title, yAxis.title, and credits. For alignment into Axis, you can set as high, middle, and low. For horizontal labels, you can set keywords as left, center, and right.

In the world of charting, x is defined for horizontal and y is defined for vertical. You can set alignment through x and y positioning for the title and subtitle. For example, if you want the title and subtitle on one line, you can use the following code:

```
title: {
text: 'Monthly Sales Chart Department Wise',
align: 'left',
},
subtitle: {
text: 'Year 2018',
align: 'right',
y: 15,
}
```

In the above code, you set left align for the title and right for the subtitle. The positioning of y is 15, so by default the title position is 15; that's the reason both are on the same line. You can set the x value for the subtitle. If you set any value for x to 15, it will move to right more (Figure 2-5).



**Figure 2-5.** Setting a Highcharts title and subtitle alignment

The `verticalAlign` property is used for setting the title and subtitle in the mode of the top, middle, and bottom.

```
title: {  
  text: 'Monthly Sales Chart Department Wise',  
},  
subtitle: {  
  text: 'Year 2018',  
  verticalAlign: 'middle',  
}
```

## Setting Up Chart Margins

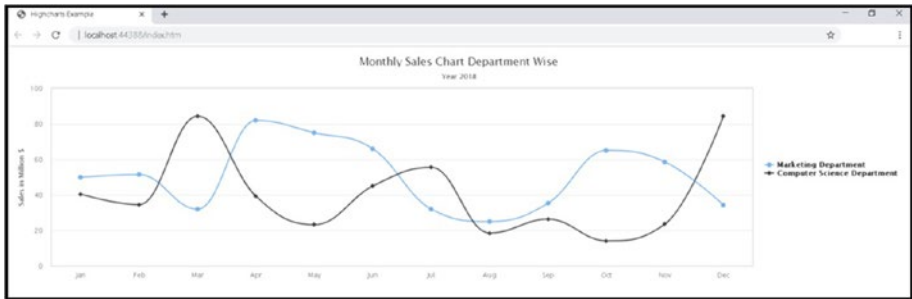
You can set chart margins with four properties: `marginTop`, `marginBottom`, `marginRight`, and `marginLeft`. This will affect the overall layout of your chart. By default these properties are not fixed, so you must set them. Once you set the margin properties, this will affect the plot area. The spacing effects are `spacingTop`, `spacingBottom`, `spacingLeft`, and `spacingRight`. Here `marginTop` sets the plot area top border, and this will also fix labels like the title and subtitle of the plot area. `spacingLeft` and `spacingRight` set the spacing areas.

## Legends

You can set the alignment of legends in Highcharts very easily. There are three properties: `align`, `verticalAlign`, and `layout`.

```
legend: {  
  align: 'right',  
  verticalAlign: 'middle' ,  
  layout: 'vertical',  
},
```

In this code, the layout property is set as vertical, so the values of the legend are displayed vertically. Here plot area automatically resizes for legends. verticalAlign is set to middle and align on the right-hand side (Figure 2-6).



**Figure 2-6.** Setting up the Highcharts legend alignment

## Setting Up Plot Lines

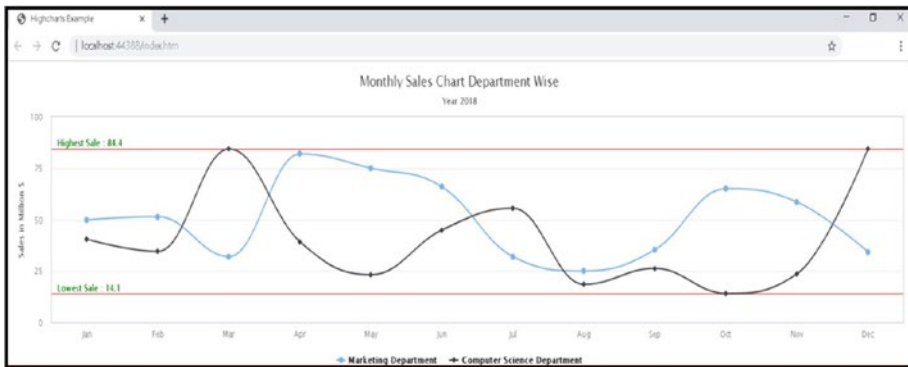
Every chart has a highest and lowest value. Suppose you must mark the highest and lowest index values. For this, plot lines are useful. Plot lines hold an array of objects configuration for each plotline. For example, see this code and Figure 2-7:

```
plotLines: [{
  width: 1,
  value: 84.4,
  color: 'red',
  label: {
    text: 'Highest Sale : 84.4',
    style: {
      color: 'green'
    }
  }
},
```

```

        {
width: 1,
value: 14.1,
color: 'red',
label: {
text: 'Lowest Sale : 14.1',
style: {
color: 'green'
}
}
}
}]

```



**Figure 2-7.** Setting Highcharts plot lines

As you can see, the plotLines properties have an array of configurations, and here you can provide index values. In the first index plot, you set your highest sale. You can set your labels with style, and then you have another index for the lowest value.



## Setting Credits

By default, the credit property is set as HighCharts.com. If you want to change this label, you can use the credit property. The credit object supports align and verticalAlign. The following is an example:

```
credits: {  
  position: {  
    align: 'center'  
  },  
  text: 'Sourabh Mishra Notes',  
    href: 'http://www.apress.com/'  
}
```

## Summary

Highcharts supports Scalable Vector Graphics internally so you can resize your charts very quickly. With the use of Highcharts, you can design and configure your charts based on your requirements. When you start developing a chart, you should know what chart type will be best for your needs. The next chapter will be very interesting because you will learn how to set up your application for Highcharts with Angular.

## CHAPTER 3

# Integrating Highcharts with Angular

In this chapter, you will learn the basics of Angular, and how you can configure and integrate Highcharts with Angular. Angular with Highcharts is a great combination.

## What Is Angular?

Angular is designed to build single page applications. Angular makes your HTML more powerful and fast. HTML is known for its tags and static web development, but with Angular, you can apply local variables, loops, and if-else conditions. Angular provides two-way model data binding. Angular is a product of Google and is top rated by millions of web developers.

Angular provides validations, routing, and binding, which makes developer life more comfortable, so you can build your apps faster. You can easily display your data fields from the data model, track your changes, and process updates from the user. Angular provides a modular approach by its design.

Every web app has a set of building blocks, and every app connects with different modules. Angular makes content easy to develop and you can create reusable code through components. Angular easily connects with back-end web services; with the use of this feature, Angular apps easily connect with HTTP get and HTTP post data to execute server-side business logic.

Angular is born for speed and for improving your web apps. It provides faster initial loads, improved page render times, and quick change detection. Angular is a modern framework with rich features and the latest JavaScript standards. Angular supports all modern browsers. Angular is a simple and rich JavaScript framework, and it provides built-in directives and two-way data binding. It is easy to learn and easy to use, and this improves your productivity in your day-to-day work.

You will get productivity improvement when you interact with Highcharts projects in later chapters.

## Configuring Angular

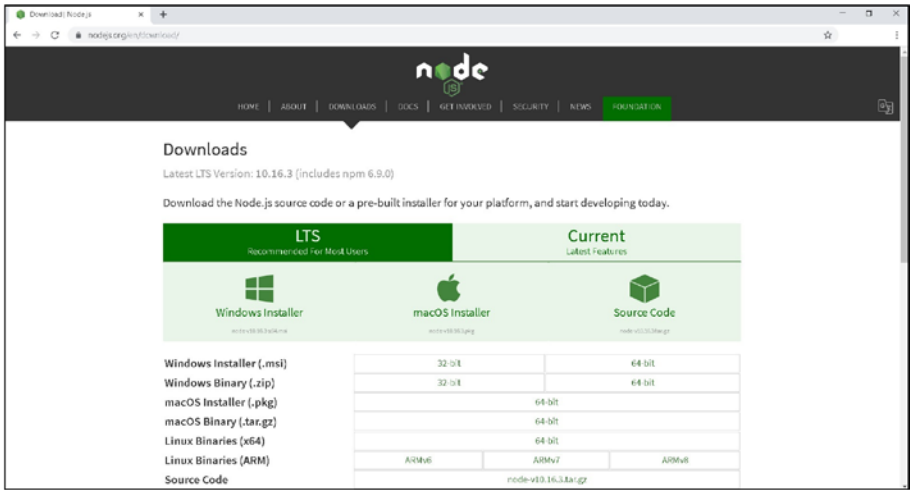
To configure Angular into your system, you have to set up a development environment with the following applications:

- Node.js
- Code editor
- Angular CLI

## Setting Up Node.js

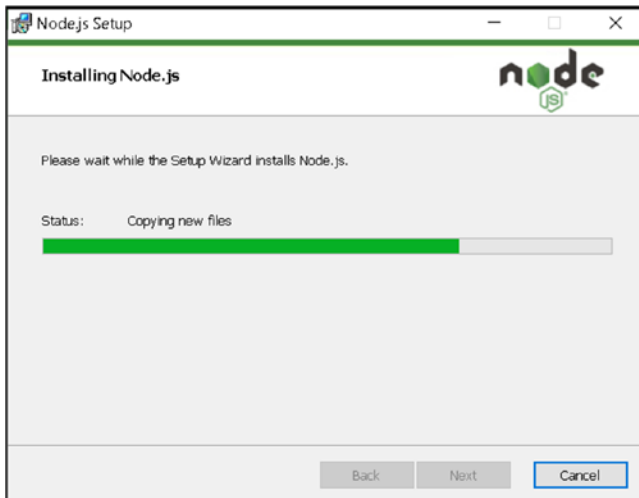
Node.js is a free, open source server environment that provides cross-platform features so you can use Windows, Linux, and macOS. Node.js uses JavaScript so it's straightforward for the developer to build services using it. Node.js provides an extensive ecosystem for open source libraries. Developers prefer Node.js because they can quickly scale up their development in any direction. Node.js is useful for developing real-time, complex, single-page applications.

To install Node.js on your system, download it from <https://nodejs.org/en/download/>. Once you open this site, you'll see the screen shown in Figure 3-1.

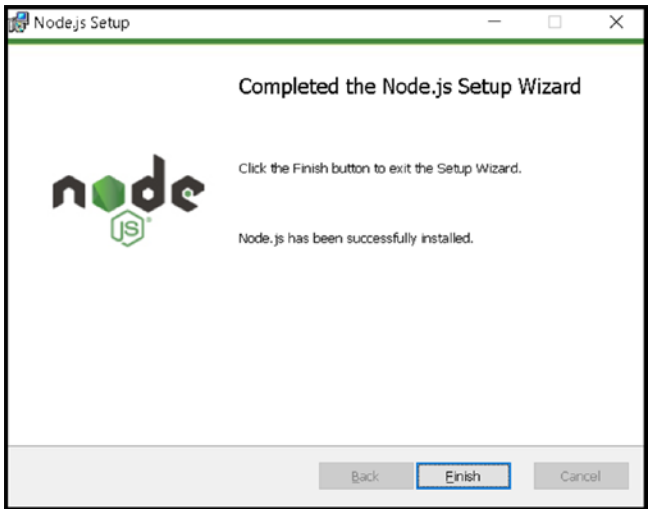


**Figure 3-1.** Download screen from the Node.js website

Figure 3-1 shows how to download Node.js based on your operating system. Once your download is complete, install the .exe file on your system (Figures 3-2 and 3-3).



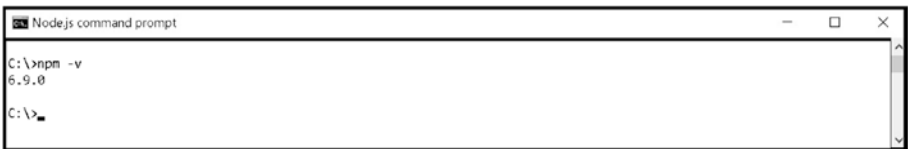
**Figure 3-2.** Installing Node.js



**Figure 3-3.** *The Node.js installation is complete*

After your installation is done, you can see the Node.js command prompt in your system. Click Start ► Programs ► Node.js command prompt.

Open the Node.js command prompt with administrator rights, and type the command `npm -v` (Figure 3-4).



**Figure 3-4.** *Node.js command prompt screen*

As you can see, after running `npm-v`, you'll see the current version of your npm.

## Code Editor

The code editor is designed to simplify and speed up the writing of source code via syntax, indentation, auto-complete, and brace matching functionality. Code editors are responsible for debugging, building, and compiling your code. These editors also provide code extensions for different programming languages.

In this book, I will work mostly in Visual Studio code. Visual Studio code is developed by Microsoft. Visual Studio code provides cross-platform features so it can run easily on the Windows, macOS, and Linux operating systems. You can think beyond syntax highlighting and autocomplete with IntelliSense, which provides smart completion based on function definitions, variable types, and imported modules. Visual Studio code provides a rich debugging feature form editor. You can insert a break point, attach processes very easy for debugging, and understand the system. You can deploy your code over the cloud very quickly. Visual Studio code also supports Git and other SCM providers.

It's a free code editor. You can download it from <https://code.visualstudio.com/download>.

Here is a list of other code editors you can download based on your preference:

- **Microsoft Visual Studio IDE:** Visual Studio Integrated Development Environment (IDE) is developed by Microsoft. It is designed for developing web applications, console-based applications, mobile applications, GUIs, services, etc. It supports many programming languages like C#, Java, C++, VB, Python, JavaScript, TypeScript, and many more.
- **Angular IDE:** Angular IDE is designed to develop Angular-based applications. It supports JavaScript and TypeScript.

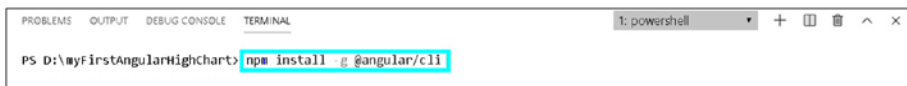
- **WebStorm:** WebStorm is a powerful tool for developing JavaScript-based products. WebStorm fully supports HTML, CSS, JavaScript, TypeScript, and Angular.
- **Bluefish:** Bluefish is a code editor for programmers and web developers. Bluefish is a very lightweight code editor, and it supports all modern programming languages. It's fast and supports work on multiple projects. Bluefish offers auto recovery of code, an inline spell checker, a character map for Unicode characters, and site upload features.

## Setting Up Angular CLI

The Angular command-line interface (CLI) is developed for automating operations for Angular projects. It saves developers time and effort. With the use of the Angular CLI, you can configure and set up your development environment. The Angular CLI is helpful for building services, components, routing, and projects, and it helps them compile and run faster.

The first step is to set up and install the Angular CLI. To run the following command in Visual Studio code, click the Terminal menu ► New Terminal. Then type the following command (as seen in Figure 3-5):

```
npm install -g @angular/cli
```



**Figure 3-5.** Installing the Angular CLI through the terminal window of VS code

Here `-g` stands for global installation. You are using it so in the future you can call the CLI in Angular projects quickly.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS D:\myFirstAngularHighChart> npm install -g @angular/cli
C:\Users\AppData\Roaming\npm> C:\Users\AppData\Roaming\npm\node_modules\angular\cli\bin\ng
> @angular/cli@3.5 postinstall C:\AppData\Roaming\npm\node_modules\angular\cli
> node ./bin/postinstall/script.js
+ @angular/cli@3.5
updated 6 packages in 42.973s
PS D:\myFirstAngularHighChart>

```

*Figure 3-6. Installation completed for the Angular CLI*

Once the `npm` installation for the Angular CLI is completed (Figure 3-6), it's time to create a new Angular application. To create/generate a new Angular application, type the following command:

**ng new application-name**

In this demo, you are going to create an application named `myFirstAngularHighChart`. So type the following command (Figure 3-7):

**ng new myFirstAngularHighChart**

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS D:\myFirstAngularHighChart> ng new myFirstAngularHighChart

```

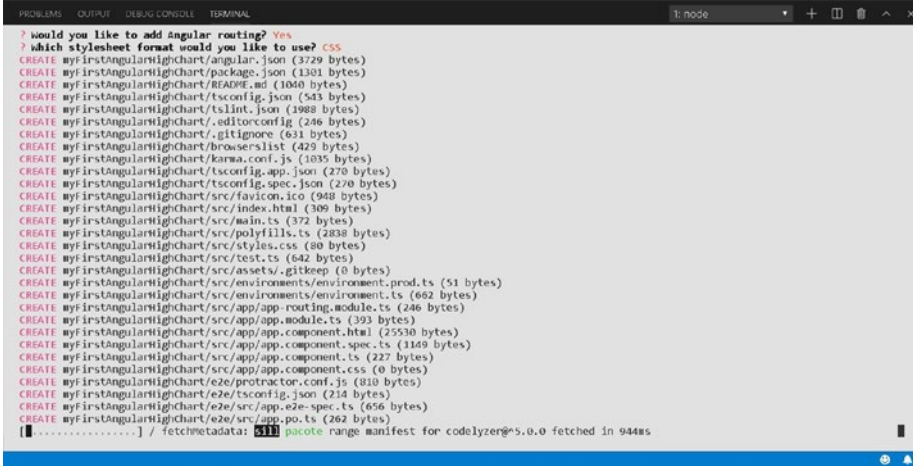
*Figure 3-7. Creating/generating a new Angular application*

Once you press `Enter`, the CLI will ask you some questions. The first one is if you **would like to add Angular routing**. Press **Y** and press `Enter`.

When it comes to picking which stylesheet format you would like to use, here you can set `CSS` or `SCSS` based on your requirements. In this application, you will work with `CSS`. Press `Enter` (Figure 3-8).



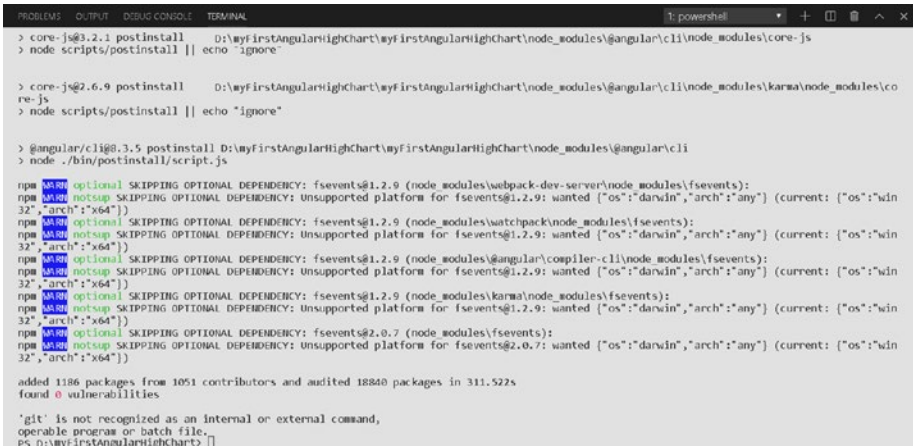
## CHAPTER 3 INTEGRATING HIGHCHARTS WITH ANGULAR



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
? would you like to add angular routing? Yes
? which stylesheet format would you like to use? CSS
CREATE myFirstAngularHighChart/angular.json (3729 bytes)
CREATE myFirstAngularHighChart/package.json (1301 bytes)
CREATE myFirstAngularHighChart/README.md (1040 bytes)
CREATE myFirstAngularHighChart/tsconfig.json (543 bytes)
CREATE myFirstAngularHighChart/tsLint.json (1988 bytes)
CREATE myFirstAngularHighChart/.editorconfig (246 bytes)
CREATE myFirstAngularHighChart/.gitignore (631 bytes)
CREATE myFirstAngularHighChart/karma.conf.js (1035 bytes)
CREATE myFirstAngularHighChart/tsconfig.app.json (270 bytes)
CREATE myFirstAngularHighChart/tsconfig.spec.json (270 bytes)
CREATE myFirstAngularHighChart/src/favicon.ico (940 bytes)
CREATE myFirstAngularHighChart/src/index.html (308 bytes)
CREATE myFirstAngularHighChart/src/main.ts (372 bytes)
CREATE myFirstAngularHighChart/src/polyfills.ts (2838 bytes)
CREATE myFirstAngularHighChart/src/styles.css (80 bytes)
CREATE myFirstAngularHighChart/src/test.ts (642 bytes)
CREATE myFirstAngularHighChart/src/assets/.gitkeep (0 bytes)
CREATE myFirstAngularHighChart/src/environments/environment.prod.ts (51 bytes)
CREATE myFirstAngularHighChart/src/app/app-routing.module.ts (246 bytes)
CREATE myFirstAngularHighChart/src/app/app.module.ts (393 bytes)
CREATE myFirstAngularHighChart/src/app/app.component.html (25530 bytes)
CREATE myFirstAngularHighChart/src/app/app.component.spec.ts (1149 bytes)
CREATE myFirstAngularHighChart/src/app/app.component.ts (227 bytes)
CREATE myFirstAngularHighChart/src/app/app.component.css (0 bytes)
CREATE myFirstAngularHighChart/e2e/protractor.conf.js (810 bytes)
CREATE myFirstAngularHighChart/e2e/tsconfig.json (214 bytes)
CREATE myFirstAngularHighChart/e2e/src/app.e2e-spec.ts (656 bytes)
CREATE myFirstAngularHighChart/e2e/src/app.po.ts (262 bytes)
[.....] / fetchmetadata: 511 package range manifest for codacy@5.0.0 fetched in 944ms
```

Figure 3-8. Creating a new Angular application

Now a process will start. It will take a few minutes to install your Angular application (Figure 3-9).



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> core-js@2.1.1 postinstall D:\myFirstAngularHighChart\myFirstAngularHighChart\node_modules\angular\cli\node_modules\core-js
> node scripts/postinstall || echo "ignore"

> core-js@2.6.9 postinstall D:\myFirstAngularHighChart\myFirstAngularHighChart\node_modules\angular\cli\node_modules\karma\node_modules\co
re-js
> node scripts/postinstall || echo "ignore"

> @angular/cli@0.3.5 postinstall D:\myFirstAngularHighChart\myFirstAngularHighChart\node_modules\angular\cli
> node -/bin/postinstall/script.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win
32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\watchpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win
32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.2.9 (node_modules\angular\compiler-cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win
32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\karma\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win
32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.0.7 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.0.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win
32","arch":"x64"})

added 1186 packages from 1051 contributors and audited 18840 packages in 311.5225
found 0 vulnerabilities

'git' is not recognized as an internal or external command,
operable program or batch file.
PS D:\myFirstAngularHighChart>
```

Figure 3-9. Angular application generation completed screen

If you see the screen shown in Figure 3-9, the Angular creation process is done. Go to File ► Open Folder, select Folder, and click Open. On the left-hand side, you can see that CLI generated your application folder structure. Now it's time to understand the Angular application structure (Table 3-1).

**Table 3-1.** *Application Structure of an Angular App*

Folders/Config Files	Use
e2e	End-to-end test files. The e2e folder contains a source file for testing.
node_modules	This folder contains npm packages for an entire application. Project dependencies also reside here.
src	In this folder, you will get application-level source code like modules, HTML, components, application-level environment config files, icon files, the main page index file, etc.
.editorconfig	Configuration for code editors
.gitignore	This configuration file will intentionally untracked that Git should ignore.
angular.json	Here you can set the default configuration for a build, serve, testing, index page, styles, tsconfig, etc.
package.json	Here you will get all required project dependencies with their versioning. Whenever you add a new dependency for your project, you will get entries here.
tsconfig.json	TypeScript configuration file for the entire app
tslint.json	Default configuration file for tslint

Now I will talk about some essential things that are required to work in Angular with Highcharts.

## TypeScript

TypeScript is an open-source programming language developed by Anders Hejlsberg at Microsoft. After compilation internally, TypeScript is converted to JavaScript. TypeScript is a pure strongly typed and object-oriented language; here you can create classes and interface like in C# and Java. Angular with TypeScript is a great combination.

Now it's time to configure Highcharts with the Angular application. For this, open a project and go to the Visual Studio code terminal window and type the following command:

```
npm install highcharts -save
```

This command will add Highcharts dependencies into your project.

## Highcharts Angular Wrapper

The Highcharts angular wrapper is open source. It provides vibrant and dynamic feature visualization for Highcharts within an Angular application. This wrapper offers browser compatibility for all modern browsers like IE, Chrome, Safari, and Mozilla. A Highcharts wrapper API is available and supports TypeScript.

For installing this wrapper, run the following command in the terminal window:

```
npm install highcharts-angular -save
```

Now open the `package.json` file. You can see in the package list new entries for Highcharts. This shows that your Highcharts dependencies are correctly installed, and now you can use them into your application.

In the Angular application, if you go to the `src` ➤ `app` folder, you will get the following files:

- `app-routing.module.ts`: This file is responsible for routing.
- `app.module.ts`: Here, the root module is defined; in this file, all related components and dependencies will be added for a module (Listing 3-1).
- `app.component.ts`: This is a component. Here you will write business logic and set a selector for HTML template and models for binding (Listing 3-2).
- `app.component.html`: This is an HTML page for components. Here you can call your logic methods and bind models (Listing 3-3).
- `app.component.css`: Here you can add your base CSS style sheet for root code.
- `app.component.spec.ts`: Here you can set a unit test for root component.

Start by adding some code into `app.module.ts`, as shown in Listing 3-1.

**Listing 3-1.** `app.module.ts`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HighchartsChartComponent } from 'highcharts-angular';

@NgModule({
  declarations: [
    AppComponent,
```

```
    HighchartsChartComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Let's try to understand the above code line by line:

- `import`: Angular modules/components are written in the TypeScript language using the `export` keyword, so in order to refer to these components/modules, you must refer to the `import` statement. The syntax to import is `import {module/component} from 'path of the file system.'`
- `BrowserModule`: This exports all required infrastructure for an Angular app.
- `@angular/platform-browser`: This executes Angular apps to all supported browsers.
- `NgModule`: `NgModule` is a class that contains the `@NgModule` decorator. It's responsible for adding dependent components.
- `@angular/core`: This is responsible for implementing Angular core functionality, utilities, and low-level services.
- `AppRoutingModule`: This belongs to `app-routing.module.ts` and is responsible for routing and navigation for the Angular app.

- **AppComponent:** AppComponent is a class declared in `app.component.ts`. You can create a class with a different name.
- **HighchartsChartComponent:** This is for Highcharts. This is a class, and here you are adding this dependency into your Angular app.
- **@NgModule:** The `ngmodule` configures the module and injects related dependencies. `ngmodule` is a decorator that declares `AppComponent` and `HighchartsChartComponent` and imports modules; bootstrapping the main component. Here *bootstrap* is a term for kick-starting your app, so whenever your application is going to run, bootstrap will initialize a component, and that component will run its related HTML into the browser.
- **export:** In Angular, whenever you define a class, and you want to use this class into different modules, you have to define it as `export`; it's the same as the `public` keyword in Java and C#.

Now copy the code in Listing 3-2 into `app.component.ts`.

**Listing 3-2.** `app.components.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
```

## CHAPTER 3 INTEGRATING HIGHCHARTS WITH ANGULAR

```
chartOptions = {
  chart: {
    type: "column"
  },
  title: {
    text: "Monthly Sales Chart Department Wise"
  },
  subtitle: {
    text: "Year 2018"
  },
  xAxis:{
    categories:["Jan", "Feb", "Mar", "Apr", "May", "Jun",
               "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
  },
  yAxis: {
    title:{
      text:"Sales in Million $"
    }
  },
  series: [{
    name: 'Marketing Department',
    data: [49.9, 51.5, 32.0, 82.0, 75.0, 66.0, 32.0, 25.0,
          35.4, 65.1, 58.6, 34.4]
  },
  {
    name: 'Computer Science Department',
    data: [40.5, 34.5, 84.4, 39.2, 23.2, 45.0, 55.6, 18.5,
          26.4, 14.1, 23.6, 84.4]
  }
  ]
};
}
```

In the above code, there is a `@Component` decorator that helps create a fundamental building block for the UI. With this decorator, your class becomes a component. The component of Angular is a subset of directives. In the next line, you use three metadata properties.

- `selector`: `selector` is used to create tags to call in HTML. In this code, you use `app-root` so in HTML you will call this as  

```
<app-root></app-root>
```
- `templateUrl`: The template URL is the relative UI path for the HTML template file.
- `styleUrls`: Here you can define a component CSS path. This is an array type. If you have multiple CSS for this component, you can define them here.

The next line creates a class named `AppComponent`. And then you call the Highcharts JavaScript code. In the next step, you move the existing code into the `app.component.html` file and copy the code from Listing 3-3 into the `app.component.html` file.

**Listing 3-3.** `app.component.html`

```
<div class="content" role="main">
  <highcharts-chart [Highcharts]="highcharts"
    [options]="chartOptions"
    style="width: 100%; height: 400px; display: block;">
  </highcharts-chart>
</div>
<router-outlet></router-outlet>
```



Let's try to understand the Listing 3-3 code. As you can see in the `app.component.html` code, you call a `highcharts-chart` directive in a `<div>`.

```
<highcharts-chart [Highcharts]="highcharts" [options]=
"chartOptions"
  style="width: 100%; height: 400px; display: block;">
</highcharts-chart>
```

Then there are two models, `[Highcharts]` and `[options]`, so in `app.component.ts`, you define `highcharts` and `chartOptions` as variables into the `AppComponent` class and define their values. In this HTML, you just bind those models.

So `<highcharts-chart></highcharts-chart>` is a calling directive, and you have two models to bind, `[Highcharts]` and `[options]`.

Listing 3-4 is the main `index.html` for your project. In it is one directive tag called `<app-root></app-root>`. In `app.component.ts` in the selector metadata property, you define this as `selector: 'app-root'`.

**Listing 3-4.** `index.html`

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyHighChartsApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

So whenever this Angular app compiles and runs in the browser, it will check the `<app-root>` from the component decorator, and from there it will take template URL of component `.html`, and then your component page will render.

To run this Angular app, open a new terminal from VS code, and type `ng serve`. Press Enter. By default it gives the URL as `localhost:4200`. Now go to a browser and run this URL. Your app will run (Figure 3-10).



**Figure 3-10.** Running an Angular app in a browser through `ng serve`

If you want to change the chart type, go to `app.component.ts` and change its type (Listing 3-5).

**Listing 3-5.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

## CHAPTER 3 INTEGRATING HIGHCHARTS WITH ANGULAR

```
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: "area"
    },
    title: {
      text: "Monthly Sales Chart Department Wise"
    },
    subtitle: {
      text: "Year 2018"
    },
    xAxis:{
      categories:["Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
    },
    yAxis: {
      title:{
        text:"Sales in Million $"
      }
    },
    series: [{
      name: 'Marketing Department',
      data: [49.9, 51.5, 32.0, 82.0, 75.0, 66.0, 32.0, 25.0,
        35.4, 65.1, 58.6, 34.4]
    },
    {
      name: 'Computer Science Department',
```

```

data: [40.5, 34.5, 84.4, 39.2, 23.2, 45.0, 55.6, 18.5,
        26.4, 14.1, 23.6, 84.4]
    ]
  };
}

```

Now run the `ng serve` command, and you will get the output shown in Figure 3-11.



**Figure 3-11.** *The Angular app with an area chart*

## Summary

Angular is the superset of JavaScript. Creating, building, compiling, and running an application through Angular is very easy. Angular provides faster execution, faster building, and agile development. Using Angular with Highcharts is a great combination. With Highcharts' Angular dependencies, you can develop stunning, beautiful charts very quickly. For large professional projects, Angular is very popular. In this chapter, you saw some basic building blocks, which are required to create an Angular app with Highcharts. In the next chapter, you will see different charting types, and how you can utilize more of Highcharts with Angular and jQuery.

## CHAPTER 4

# Different Charting Types

In this chapter, you will learn about the different charting types you can develop with the use of Highcharts. This chapter will cover the different types of charts in detail and how you can apply them to your web application using Angular.

You will explore the following charts in this chapter:

- Pie chart
- Donut chart
- Drilldown chart
- Line chart
- Area chart
- Scatter chart
- Histogram chart
- Heatmap series chart
- Stacked bar chart
- Column pyramid chart
- Gauge chart

## Pie Charts

In a pie chart, each slice of the pie describes how much data exists for it. Pie charts are mostly used in business, construction, media, and market research. For business, a pie chart may help to show business success or failure based on each product. You can also figure out the diet of a person with a pie chart. A benefit of the pie chart is there is no axis to configure the data; only data with categories are required.

Let's start by creating a simple pie chart with Angular and Highcharts. In earlier chapters, you created the basic Angular configuration and an application. In this chapter, you will work on the component level only, so most of the work will be done in `app.component.ts`. For the creation of this component, you can refer to Chapter 3.

In this example, you are going to create a pie chart that describes various programming languages used by developers worldwide. See Listings 4-1 and 4-2.

### **Listing 4-1.** `app.component.ts`

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10. highcharts = Highcharts;
11. chartOptions = {
12.   chart: {
13.     type: 'pie'
```

```
14. },
15. title: {
16. text: 'Programming Languages used by developers worldwide'
17. },
18. plotOptions: {
19. pie: {
20. allowPointSelect: true,
21. cursor: 'pointer',
22. dataLabels: {
23. enabled: true,
24. format: '<b>{point.name}</b>: {point.percentage:.1f} %'
25. }
26. }
27. },
28. tooltip: {
29. pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
30. },
31. series: [{
32. name: 'Uses',
33. colorByPoint: true,
34. data: [{
35. name: 'C#',
36. y: 55,
37. sliced: true,
38. selected: true
39. }, {
40. name: 'VB',
41. y: 25
42. }, {
43. name: 'J#',
44. y: 10
```

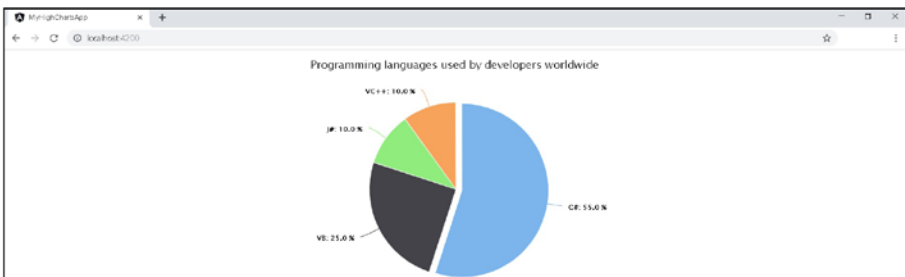
```
45. }, {  
46. name: 'VC++',  
47. y: 10  
48. }]  
49. }]  
50. };  
51. }
```

**Listing 4-2.** app.component.html

```
1. <div class="content" role="main">  
2. <highcharts-chart [Highcharts]="highcharts"  
   [options]="chartOptions"  
3. style="width: 100%; height: 400px; display: block;">  
4. </highcharts-chart>  
5. </div>  
6. <router-outlet></router-outlet>
```

In this chapter, for all examples, the app.component.html code will be same (Listing 4-2); you only have to change code in app.component.ts (Listing 4-1).

To run this example, type ng serve and press Enter. You will get the output shown in Figure 4-1.



**Figure 4-1.** Simple pie chart



Now let's try to understand the code in the `app.component.ts` file. In the last chapter, I discuss the basics of Angular. Here I will talk about the Highcharts code.

In the `app.component.ts` code, you set the `type` property as `pie` so it creates a pie chart.

Now let's look at `plotOptions`. `plotOptions` is a wrapper object for the configuration for each series type:

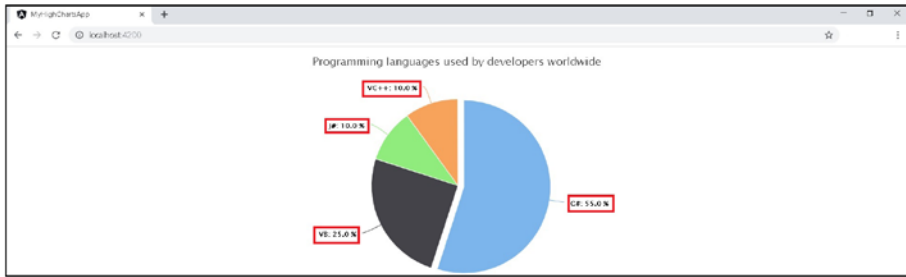
```
plotOptions: {
  pie: {
    allowPointSelect: true,
    cursor: 'pointer',
    dataLabels: {
      enabled: true,
      format: '<b>{point.name}</b>: {point.
percentage:.1f} %'
    }
  }
}
```

Next, `allowPointSelect` is a Boolean type property. Here it's set to `true` so that the user can click over the chart to select and deselect that particular series in the chart.

For example, in this chart, if the user clicks `C#` or `VB`, that specific slice will select and deselect based on the click. If set to `false`, this functionality will not work.

For `cursor`, whenever a mouse pointer hovers into a series, the hand type mouse pointer will appear if you set this as `cursor: 'pointer'`.

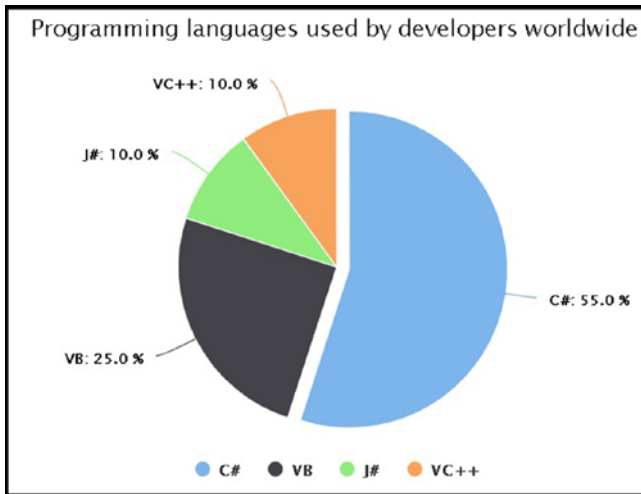
You can see data labels in each series. Here you must enable `dataLabels` to `true` so you can set the format of your data into labels. Figure 4-2 shows the data labels; in this example, red rectangles define the data labels.



**Figure 4-2.** *The dataLabels property in a pie chart*

If you want to set the legend in this particular pie chart, add `showInLegend: true`. You can set this property after the `allowPointSelect` property. The following is the code, and the chart is shown in Figure 4-3:

```
pie: {
    allowPointSelect: true,
    showInLegend: true,
    cursor: 'pointer',
    dataLabels: {
        enabled: true,
        format: '<b>{point.name}</b>: {point.percentage:.1f} %'
    }
}
```



**Figure 4-3.** A pie chart with legends

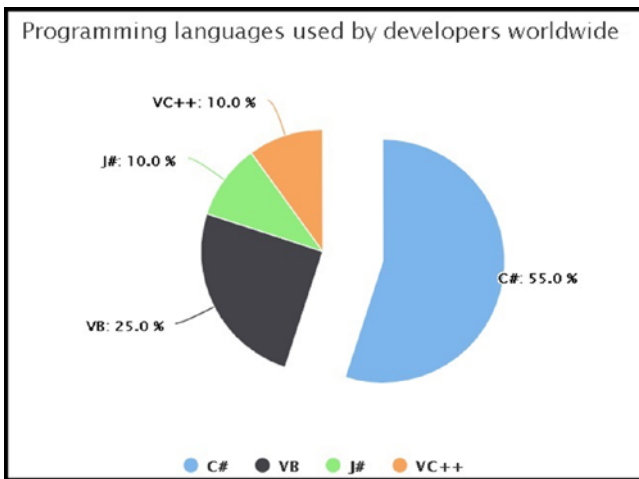
The `slicedOffset` property sets how far you want to move out the particular section of pie from the chart. By default the value is set to 10, but you can increase it.

Next, `sliced` is a Boolean property. If it's set as true, based on the series where you are applying this property, it will slice it off from the pie chart (plus any offset you set) that much distance. Consider the following code:

```
pie: {
    allowPointSelect: true,
    showInLegend: true,
    slicedOffset: 50,
    cursor: 'pointer',
    dataLabels: {
        enabled: true,
        format: '<b>{point.name}</b>: {point.
percentage:.1f} %'
    }
}
```

```
series: [{  
  name: 'Uses',  
  data: [{  
    name: 'C#',  
    y: 55,  
    sliced: true,  
    selected: true  
  }  
}]
```

Run this code to display the chart shown in Figure 4-4.



*Figure 4-4. A pie chart with the sliced and slicedOffset properties*

## Donut Chart

A donut chart is another type of pie chart. It's useful when you want detailed information. The center hole in this chart makes it look like a donut shape.

The next example shows how a donut chart is helpful for details. In this example, you will get the details on different JavaScript frameworks used by developers. This demo is just an example; it's not real data.

Here, I wish to show you how to develop a donut chart and make your life easier. Listing 4-3 shows the code.

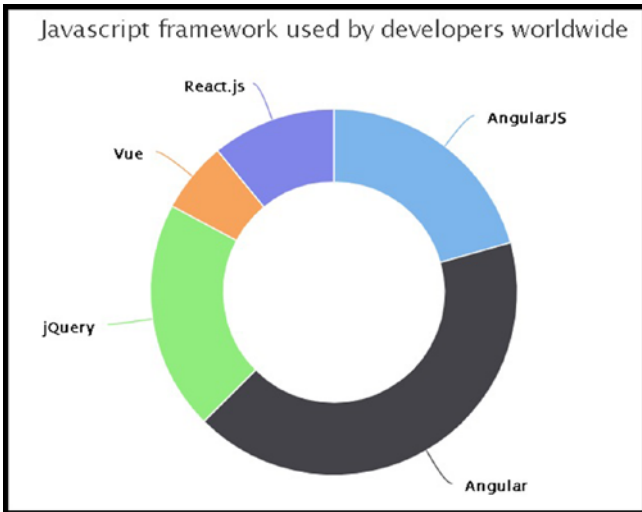
**Listing 4-3.** app.component.ts

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10.  highcharts = Highcharts;
11.  chartOptions = {
12.    chart: {
13.      renderTo: 'container',
14.      type: 'pie'
15.    },
16.    title: {
17.      text: 'Javascript framework used by developers worldwide'
18.    },
19.    plotOptions: {
20.      pie: {
21.        innerSize: '60%'
22.      }
23.    },
24.    series:
25.    [
26.    {
27.      name: 'Uses',
28.      data: [
```

- 29. ['AngularJs', 10.2],
- 30. ['Angular', 20.7],
- 31. ['jQuery', 10],
- 32. ['Vue', 3.1],
- 33. ['ReactJs', 5.4]
- 34. ]
- 35. }
- 36. ]
- 37. }

As you can see in Listing 4-3, one property is `innerSize:60%`; this property makes a hole in the pie chart, which gives it a donut design. You can increase or decrease it as per your requirements (Figure 4-5):

```
plotOptions: {  
  pie: {  
    innerSize: '60%'  
  }  
}
```



**Figure 4-5.** Pie chart with donut feature

## Drilldown Charts

Drilldown charts provide an in-depth and detailed view of your chart. Highcharts provides a drilldown effect on the pie chart so you can get more details into your chart. For adding a drilldown effect into your charts, you must add some dependencies in your code.

## Required Dependencies

### jQuery:

```
<script src="https://code.highcharts.com/highcharts-more.js">
</script>
<script src="https://code.highcharts.com/modules/drilldown.js">
</script>
```

### Angular:

```
import More from 'highcharts/highcharts-more';
More(Highcharts);
import Drilldown from 'highcharts/modules/drilldown';
Drilldown(Highcharts);
```

## Setting Up the Unique Name for a Series

Drilldown charts are basically designed for detailing a chart. In a series, suppose you have four types of information and on each click you want to see the details of that particular type, so you require unique names. These unique names are used to connect with a drilldown event. Listing 4-4 shows the syntax for creating a drilldown series and Listing 4-5 shows how to use the same unique names in the listing and get detailed information about the chart.

**Listing 4-4.** Creating a Series for a Drilldown Chart

```

series: [{
  name: 'Series Name',
  data: [
    {
      name: 'name of series',
      y: 62.12,
      drilldown: 'unique-name'
    },
    ['Data 1', value 1],
    ['Data 2', value 2],
    ['Data 3', value 3]
  ]
}],

```

**Listing 4-5.** Getting the Details on a Click for the Drilldown

```

drilldown: {
series: [{
name: 'name of drill down series',
id: ' unique-name',
data: [
['Detail 1', value 1],
['Detail 2', value 2],
['Detail 3', value 3],
['Detail 4',value 4],
['Detail 5', value 5]
]
}]

```



In the upcoming example, you will draw a drilldown chart, which gives you in-depth details of JavaScript framework versions. If you click one framework, such as Angular, it will take you down one more layer in that particular series. Copy the complete code in Listing 4-6 and paste it into the `app.component.ts` file.

**Listing 4-6.** `app.component.ts`

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. import More from 'highcharts/highcharts-more';
4. More(Highcharts);
5. import Drilldown from 'highcharts/modules/drilldown';
6. Drilldown(Highcharts);
7. @Component({
8.   selector: 'app-root',
9.   templateUrl: './app.component.html',
10.  styleUrls: ['./app.component.css']
11. })
12. export class AppComponent {
13.   title = 'myHighChartsApp';
14.   highcharts = Highcharts;
15.   chartOptions = {
16.     chart: {
17.       type: 'pie',
18.     },
19.     title: {
20.       text: 'Pie Chart with drill down Feature'
21.     },
22.     plotOptions: {
23.       pie: {
24.         innerSize: 100,
```

```
25. }
26. },
27. tooltip: {
28.   headerFormat: '<span style="font-size:10px">{series.
      name}</span><br>',
29.   pointFormat: '<span style="color:{point.color}">{point.
      name}</span>: <b>{point.y:.2f}%</b> of total<br/>'
30. },
31. series: [{
32.   name: 'JavaScript Frameworks',
33.   data: [
34.     {
35.       name: 'Angular',
36.       y: 62.12,
37.       drilldown: 'angular-versions'
38.     },
39.     ['VueJs', 9.35],
40.     ['ReactJs', 15.89],
41.     ['jQuery', 12.64]
42.   ]
43. }],
44. drilldown: {
45.   series: [{
46.     name: 'Angular versions',
47.     id: 'angular-versions',
48.     data: [
49.       ['Angular Js', 17.07],
50.       ['Angular 2', 25],
51.       ['Angular 5', 30],
52.       ['Angular 7', 20.58],
```

```

53. ['Angular 8', 7.35]
54. ]
55. }]
56. }
57. }
58. }

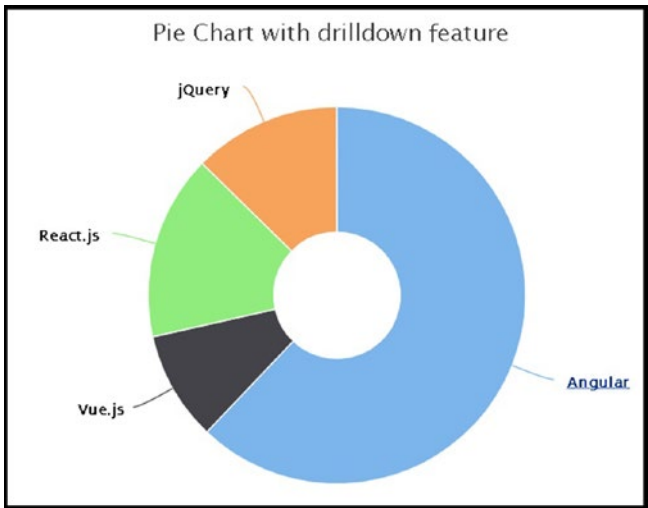
```

Listing 4-6 provides a drilldown feature for the first series array only for Angular; the rest of the frameworks, like React, don't get the drilldown functionality. If you want to provide the drilldown effect, you must set the series as `drilldown: 'uniquename for the drilldown'`. This unique name is required because when you go into detailing this drilldown, the unique property name should match with the drilldown of the series array. You can add this drilldown feature into another series array. So always remember that unique name should be different.

```

series: [{
  name: 'JavaScript Frameworks',
  data: [
    {
      name: 'Angular',
      y: 62.12,
      drilldown: 'angular-versions'
    },
    ['VueJs', 9.35],
    ['ReactJs', 15.89],
    ['jQuery', 12.64]
  ]
}],

```



**Figure 4-6.** *Drilldown with pie feature*

Whenever you run this code, it will first look like Figure 4-6. Now let's go to the next level of the code:

```
drilldown: {  
  series: [{  
    name: 'Angular versions',  
    id: 'angular-versions',  
    data: [  
      ['Angular Js', 17.07],  
      ['Angular 2', 25],  
      ['Angular 5', 30],  
      ['Angular 7', 20.58],  
      ['Angular 8', 7.35]  
    ]  
  }]  
}
```

In this code, the drilldown id ( 'angular-versions' ) matches your series drilldown property value; both ids are the same as the Angular versions. So this way, Highcharts interacts within the features it has to call. In this example, once you click the Angular part, you will get Figure 4-7.



**Figure 4-7.** Pie with drilldown detailed effect after you click the series

In Figure 4-7, there is a button that comes automatically label as <Back to JavaScript Frameworks>. Clicking this button takes you to Figure 4-6. This is how you can implement drilldown effects into pie charts. I hope you enjoy this drilldown feature with Highcharts and pie charts.

## Line Charts

The line chart is also known as a line plot, line graph, or curve chart. It's a type of diagram that displays information in a series of data points, which are called markers. These markers connect by straight lines. Let's create a simple line chart. See Listing 4-7.

**Listing 4-7.** app.component.ts

1. `import { Component } from '@angular/core';`
2. `import * as Highcharts from 'highcharts';`
3. `@Component({`
4. `selector: 'app-root',`

## CHAPTER 4 DIFFERENT CHARTING TYPES

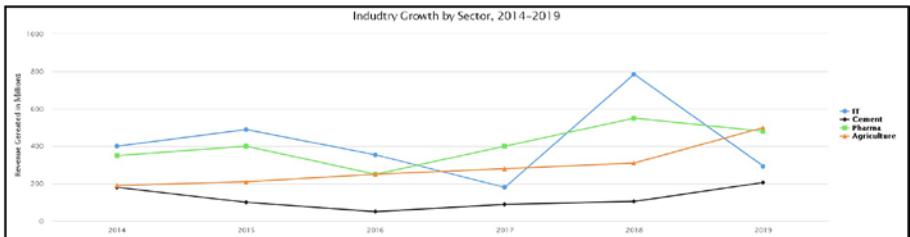
```
5.  templateUrl: './app.component.html',
6.  styleUrls: ['./app.component.css']
7.  })
8.  export class AppComponent {
9.    title = 'myHighChartsApp';
10.  highcharts = Highcharts;
11.  chartOptions = {
12.    chart:{
13.      type:'line'
14.    },
15.    title: {
16.      text: 'Industry Growth by Sector, 2014-2019'
17.    },
18.    xAxis: {
19.      categories: [2014, 2015, 2016, 2017, 2018, 2019],
20.    },
21.    yAxis: {
22.      title: {
23.        text: 'Revenue Generated in million'
24.      }
25.    },
26.    legend: {
27.      layout: 'vertical',
28.      align: 'right',
29.      verticalAlign: 'middle'
30.    },
31.    series: [{
32.      name: 'IT',
33.      data: [400, 489, 354, 180, 785, 293]
34.    }, {
35.      name: 'Cement',
```

```

36. data: [180, 100, 50, 89, 105,206]
37. }, {
38. name: 'Pharmacy',
39. data: [350, 400, 250, 400, 550,480]
40. }, {
41. name: 'Agriculture',
42. data: [190, 210, 250, 280, 310,500]
43. }],
44. }
45. }

```

This code is basic line chart code, where you create a chart to see industry growth with multiple lines. By default, in Highcharts, the chart type is set as line. If you do not use chart type, it will draw a line chart for you. See Figure 4-8.



**Figure 4-8.** Basic multiple line chart

## Area Charts

An area chart represents changes that happen over time; it used to display quantitative data. At the time of development, the Highcharts chart type is area. In this chart, the x-axis part is shaded with colors. Let's create your first area chart. Copy Listing 4-8 into the `app.component.ts` file.

**Listing 4-8.** app.component.ts

```
1.  import { Component } from '@angular/core';
2.  import * as Highcharts from 'highcharts';
3.  @Component({
4.    selector: 'app-root',
5.    templateUrl: './app.component.html',
6.    styleUrls: ['./app.component.css']
7.  })
8.  export class AppComponent {
9.    title = 'myHighChartsApp';
10.   highcharts = Highcharts;
11.   chartOptions = {
12.     chart:{
13.       type:'area'
14.     },
15.     title: {
16.       text: 'Average scored by students in Computer Science'
17.     },
18.     xAxis: {
19.       categories: ['Quarterly', 'Six Monthly', 'Final Year'],
20.     },
21.     yAxis: {
22.       title: {
23.         text: 'Average Scores'
24.       }
25.     },
26.     legend: {
27.       layout: 'vertical',
28.       align: 'right',
29.       verticalAlign: 'middle'
30.     },
```

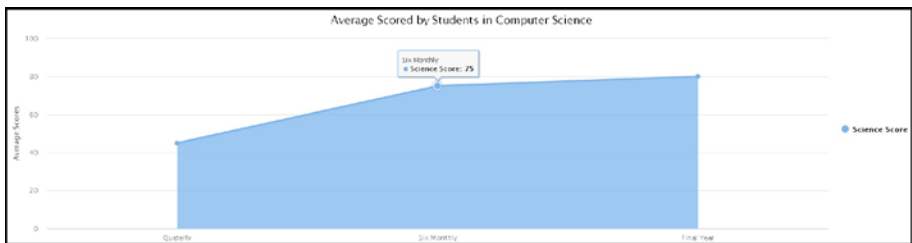


```

31. series: [{
32.   name: 'Science Score',
33.   data: [45, 75, 80]
34. }],
35. }
36. }

```

This is just a simple area chart to help you understand how to develop an area chart with Highcharts. Once you run this code, you will get output like Figure 4-9.



**Figure 4-9.** Basic area chart

Listing 4-9 shows how to develop an area chart with negative values.

**Listing 4-9.** app.component.ts

```

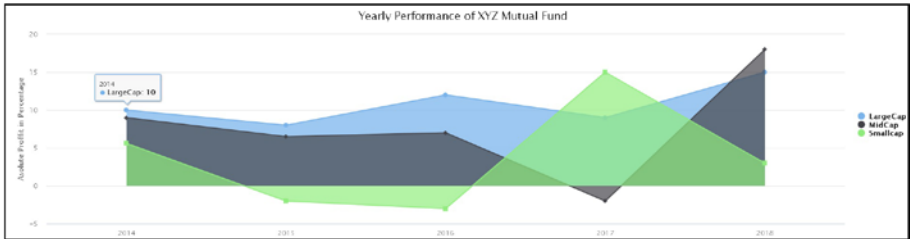
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10.  highcharts = Highcharts;

```

## CHAPTER 4 DIFFERENT CHARTING TYPES

```
11. chartOptions = {
12. chart:{
13. type:'area'
14. },
15. title: {
16. text: 'Yearly Performance of XYZ Mutual Fund'
17. },
18. xAxis: {
19. categories: [2014,2015, 2016, 2017,2018],
20. },
21. yAxis: {
22. title: {
23. text: 'Absolute Profit in percentage'
24. }
25. },
26. legend: {
27. layout: 'vertical',
28. align: 'right',
29. verticalAlign: 'middle'
30. },
31. series: [{
32. name: 'Large Cap',
33. data: [10, 8, 12, 9, 15]
34. }, {
35. name: 'Mid cap',
36. data: [9, 6.5, 7, -2, 18]
37. }, {
38. name: 'Small cap',
39. data: [5.6, -2, -3, 15, 3]
40. }],
41. }
42. }
```

As you can see in Listing 4-9, in different series there are negative values based on the area chart constructed (Figure 4-10).



**Figure 4-10.** Area chart with negative values

In the next example, you will learn about the area-spline chart, which has features of the area and spline charts. Let's take a look. Copy the complete code in Listing 4-10 into the `app.component.ts` file.

**Listing 4-10.** `app.component.ts`

```

1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10.  highcharts = Highcharts;
11.  chartOptions = {
12.    chart: {
13.      type: 'areaspline'
14.    },
15.    title: {

```

## CHAPTER 4 DIFFERENT CHARTING TYPES

```
16. text: 'Number of visitors visited Taj Mahal in a week'
17. },
18. legend: {
19. layout: 'vertical',
20. align: 'left',
21. verticalAlign: 'top',
22. },
23. xAxis: {
24. categories: [
25. 'Monday',
26. 'Tuesday',
27. 'Wednesday',
28. 'Thursday',
29. 'Friday',
30. 'Saturday',
31. 'Sunday'
32. ],
33. plotBands: [{ // Design to visualize the weekend
34. from:5,
35. to: 6,
36. color: 'orange'
37. }]
38. },
39. yAxis: {
40. title: {
41. text: 'Number of visitors'
42. }
43. },
44. tooltip: {
45. valueSuffix: ' people'
46. },
```

```

47. plotOptions: {
48.   areaspline: {
49.     fillOpacity: 0.6
50.   }
51. },
52. series: [{
53.   name: 'Taj Mahal',
54.   data: [5000, 2700, 3200, 3800, 4100, 5600, 6000]
55. }]
56. }
57. }

```

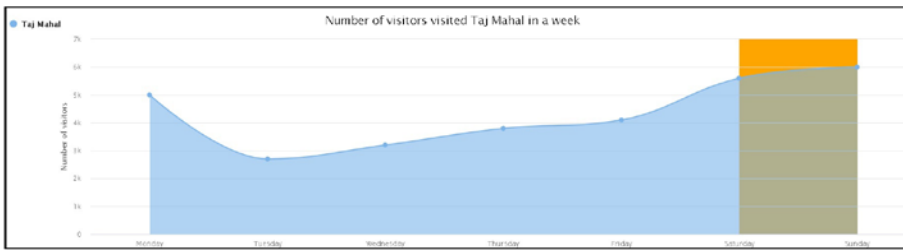
Listing 4-10 is the example of an area-spline chart, which is the combination of area and spline charts. This chart demonstrates how many visitors come to see the Taj Mahal in a week. If you set the code chart type as `areaspline`, you can use the `plotBands` property to highlight the weekend in the chart (Figure 4-11):

```

plotBands: [{ // Design to visualize the weekend
  from:5,
  to: 6,
  color: 'orange'
}]

```

Here you use two properties, `from` and `to`, so `from` demonstrates where to start and `to` describes where to end. The count starts from 0. You want to start on Saturday, so you set 5 as the `from` property and 6 as the `to` property. See Figure 4-11.



**Figure 4-11.** Area spline chart with plotBands

## Scatter Charts

A scatter plot, also known as a scatter graph, scattergram, or scatter chart, is a type of plot that uses Cartesian coordinate to display typically two variables for a set of data. Points are coded and defined using color and shape/size. Data is presented in the collection of points; each point has one value. Copy the code in Listing 4-11 into `app.component.ts`.

**Listing 4-11.** `app.component.ts`

```

1.  import { Component } from '@angular/core';
2.  import * as Highcharts from 'highcharts';
3.  @Component({
4.    selector: 'app-root',
5.    templateUrl: './app.component.html',
6.    styleUrls: ['./app.component.css']
7.  })
8.  export class AppComponent {
9.    title = 'myHighChartsApp';
10.   highcharts = Highcharts;
11.   chartOptions = {
12.     chart: {
13.       type: 'scatter',
14.       zoomType: 'xy'

```

```
15. },
16. title: {
17.   text: 'Height V/s Weight of S.T. Thomas Collage by Gender'
18. },
19. xAxis: {
20.   title: {
21.     enabled: true,
22.     text: 'Height (cm)'
23.   },
24.   startOnTick: true,
25.   endOnTick: true,
26.   showLastLabel: true
27. },
28. yAxis: {
29.   title: {
30.     text: 'Weight (kg)'
31.   }
32. },
33. legend: {
34.   layout: 'vertical',
35.   align: 'left',
36.   verticalAlign: 'top',
37.   x: 150,
38.   y: 40,
39.   floating: true,
40.   borderWidth: 1
41. },
42. plotOptions: {
43.   scatter: {
44.     marker: {
45.       radius: 5,
```

## CHAPTER 4 DIFFERENT CHARTING TYPES

```
46. states: {
    a. hover: {
    b. enabled: true,
    c. lineColor: 'black'
    d. }
47. }
48. },
49. states: {
50. hover: {
    a. marker: {
    b. enabled: false
    c. }
51. }
52. },
53. tooltip: {
54. headerFormat: '<b>{series.name}</b><br>',
55. pointFormat: '{point.x} cm, {point.y} kg'
56. }
57. }
58. },
59. series: [{
60. name: 'Female',
61. color: 'red',
62. data: [[151.2, 53.1], [157.3, 51.0], [169.5, 69.2],
        [147.0, 50.0], [175.8, 83.6],
63. [150.0, 51.0], [151.1, 57.9], [156.0, 79.8], [146.2,
        46.8], [158.1, 74.9],
64. ]
65. }, {
66. name: 'Male',
67. color: 'blue',
```



```

68. data: [[172.0, 63.7], [165.3, 72.7], [183.5, 79.2],
          [176.5, 75.7], [177.2, 85.8],
69. [171.5, 64.8], [181, 82.4], [174.5, 77.4], [177.0, 61.0],
          [174.0, 83.7],
70. ]
71. }]
72. }
73. }

```

Listing 4-11 gives you a chart of the height and weight of students based on their gender. The code chart type is scatter:

```

chart: {
    type: 'scatter',
    zoomType: 'xy'
},

```

Here zoomType is xy, and it means if you drag your mouse on the x-axis or y-axis, your graph will automatically zoom. The zoomType property can be used in any graph in the chart section.

In the plotOptions section, you can set the scatter subproperty as the radius of a circle line color once a mouse hovers.

```

plotOptions: {
    scatter: {
        marker: {
            radius: 5,
            states: {
                hover: {
                    enabled: true,
                    lineColor: 'black'
                }
            }
        }
    },
},

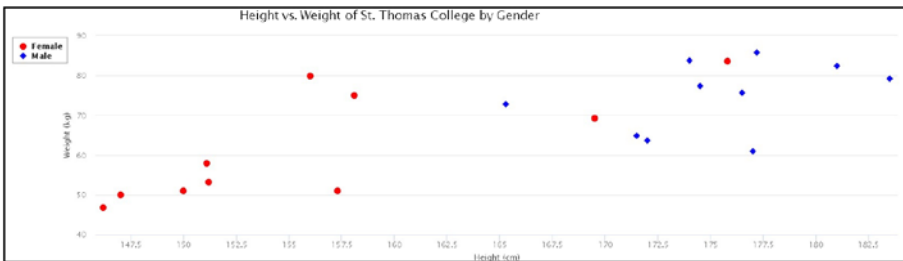
```

```

states: {
  hover: {
    marker: {
      enabled: false
    }
  },
  tooltip: {
    headerFormat: '<b>{series.name}</b><br>',
    pointFormat: '{point.x} cm, {point.y} kg'
  }
}
}

```

Run this code and you will get the output shown in Figure 4-12.



*Figure 4-12. Scatter chart*

## Histogram Charts

A histogram chart is the way to put a group of data into a user-specified range. A histogram looks like a bar chart. This type of chart is used for statistical analysis to illustrate how many kinds of variables are in a specific range, such as data in the form of graph, like census data of a state or how many people are a particular age. See Listing 4-12.

**Listing 4-12.** app.component.ts

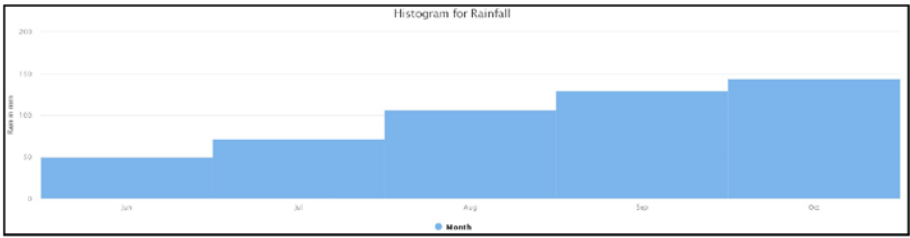
```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10.  highcharts = Highcharts;
11.  chartOptions = {
12.    chart: {
13.      type: 'column'
14.    },
15.    title: {
16.      text: 'Histogram for Rainfall'
17.    },
18.    xAxis: {
19.      categories: [
20.        'Jun',
21.        'Jul',
22.        'Aug',
23.        'Sep',
24.        'Oct',
25.      ],
26.      crosshair: true
27.    },
28.    yAxis: {
29.      title: { text: 'Rain in mm' },
```

```
30. min: 0,  
31. },  
32. plotOptions: {  
33.   column: {  
34.     pointPadding: 0,  
35.     borderWidth: 0,  
36.     groupPadding: 0,  
37.     shadow: false  
38.   }  
39. },  
40. series: [{  
41.   name: 'Month',  
42.   data: [49.9, 71.5, 106.4, 129.2, 144.0]  
43. }]  
44. }  
45. }
```

This histogram chart is developed by setting the chart type to `column`. For this, you set `plotOptions` as

```
plotOptions: {  
  column: {  
    pointPadding: 0,  
    borderWidth: 0,  
    groupPadding: 0,  
    shadow: false  
  }  
},
```

Once you run this chart, you will get the result shown in [Figure 4-13](#).



**Figure 4-13.** Basic histogram chart with chart type as column

## Heat Map Series Charts

A heat map series chart is the way to represent data values in the form of a matrix. The matrix is defined by different colors. If you want to implement a heat map series for your dashboard, you have to configure the following things:

**jQuery:** Add this code to the script section:

```
<script src="https://code.highcharts.com/modules/heatmap.js">
</script>
```

**Angular:** Add this code to the `app.component.ts` file:

```
import Heatmap from 'highcharts/modules/heatmap';
Heatmap(Highcharts);
```

Now see Listing 4-13.

### **Listing 4-13.** `app.component.ts`

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. import Heatmap from 'highcharts/modules/heatmap';
4. Heatmap(Highcharts);
5. @Component({
6.   selector: 'app-root',
```

## CHAPTER 4 DIFFERENT CHARTING TYPES

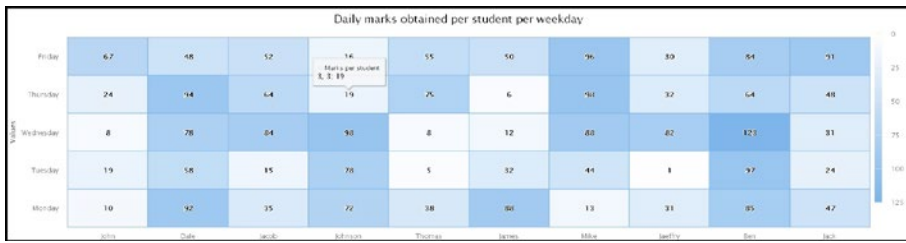
```
7.  templateUrl: './app.component.html',
8.  styleUrls: ['./app.component.css']
9.  })
10. export class AppComponent {
11.  title = 'myHighChartsApp';
12.  highcharts = Highcharts;
13.  chartOptions = {
14.  chart: {
15.  type: 'heatmap',
16.  plotBorderWidth: 1
17.  },
18.  title: {
19.  text: 'Daily marks obtained per student per weekday'
20.  },
21.  xAxis: {
22.  categories: ['John', 'Dale', 'Jacob', 'Johnson', 'Thomas',
23.  'James', 'Mike', 'Jaeffry', 'Ben', 'Jack']
24.  },
25.  yAxis: {
26.  categories: ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
27.  'Friday'],
28.  },
29.  colorAxis: {
30.  min: 0,
31.  minColor: '#FFFFFF',
32.  maxColor: Highcharts.getOptions().colors[0]
33.  },
34.  legend: {
35.  align: 'right',
36.  layout: 'vertical',
37.  margin: 0,
```

```

36. verticalAlign: 'top',
37. y: 25,
38. symbolHeight: 280
39. },
40. series: [{
41. name: 'Marks per student',
42. borderWidth: 1,
43. data: [[0, 0, 10], [0, 1, 19], [0, 2, 8], [0, 3, 24],
          [0, 4, 67], [1, 0, 92], [1, 1, 58], [1, 2, 78], [1, 3, 94],
          [1, 4, 48], [2, 0, 35], [2, 1, 15], [2, 2, 84], [2, 3, 64],
          [2, 4, 52], [3, 0, 72], [3, 1, 78], [3, 2, 98], [3, 3, 19],
          [3, 4, 16], [4, 0, 38], [4, 1, 5], [4, 2, 8], [4, 3, 75],
          [4, 4, 55], [5, 0, 88], [5, 1, 32], [5, 2, 12], [5, 3, 6],
          [5, 4, 50], [6, 0, 13], [6, 1, 44], [6, 2, 88], [6, 3, 98],
          [6, 4, 96], [7, 0, 31], [7, 1, 1], [7, 2, 82], [7, 3, 32],
          [7, 4, 30], [8, 0, 85], [8, 1, 97], [8, 2, 123], [8, 3, 64],
          [8, 4, 84], [9, 0, 47], [9, 1, 24], [9, 2, 31],
          [9, 3, 48], [9, 4, 91]],
44. dataLabels: {
45. enabled: true,
46. color: '#000000'
47. }
48. }]
49. }
50. }

```

In Listing 4-13, the chart type is heatmap. This example shows the marks of different students over the course of a week, in the form of a matrix. The x-axis shows student names. If you run this code, you will get the output in Figure 4-14.



**Figure 4-14.** Heat map series chart

As you can see, the right-hand side legend is a different kind of legend. This kind of legend is provided in a heatmap. See the following code:

```

legend: {
  align: 'right',
  layout: 'vertical',
  margin: 0,
  verticalAlign: 'top',
  y: 25,
  symbolHeight: 280
},
    
```

In the series section, data is written like `data:[[0,0,10],[0,1,19]...]`, so here it's defined in the form of the column, row, and marks. This is the way to define a matrix.

## Stacked Bar Charts

A stacked chart represents different groups on top of each other. The height of the bar represents the combined result of the group. Stacked bars are not suitable when some groups have negative values. Let's take a look. See Listing 4-14.



**Listing 4-14.** app.component.ts

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9.   title = 'myHighChartsApp';
10.  highcharts = Highcharts;
11.  chartOptions = {
12.    chart: {
13.      type: 'column'
14.    },
15.    title: {
16.      text: 'Total hours studies in a week'
17.    },
18.    xAxis: {
19.      categories: ['Maths', 'Science', 'History', 'Social
20.        Science', 'English']
21.    },
22.    yAxis: {
23.      min: 0,
24.      title: {
25.        text: 'Total Hour studied'
26.      },
27.      stackLabels: {
28.        enabled: true,
```

```
28. style: {
    fontWeight: 'bold',
    color: ( // theme
    Highcharts.defaultOptions.title.style &&
    Highcharts.defaultOptions.title.style.color
    ) || 'gray'
29. }
30. }
31. },
32. legend: {
33. align: 'right',
34. x: -30,
35. verticalAlign: 'top',
36. y: 25,
37. floating: true,
38. backgroundColor:
39. Highcharts.defaultOptions.legend.backgroundColor ||
    'white',
40. borderColor: '#CCC',
41. borderWidth: 1,
42. shadow: false
43. },
44. plotOptions: {
45. column: {
46. stacking: 'normal',
47. dataLabels: {
    a. enabled: true
48. }
49. }
50. },
```

```

51. series: [{
52.   name: 'Rocy',
53.   data: [4, 2, 1, 8, 9]},
54. {name: 'Luies',
55.  data: [1, 5, 1, 4, 2]},
56. {name: 'Simon',
57.  data: [7, 2, 3, 1, 4]
58. }]
59. }
60. }

```

This code is a perfect example of a stacked chart. This code calculates the total hours a student spends on a particular subject in a week. The chart type is `column`; in Highcharts, there is no `stackedchart` type, but you can develop one with a bar column, etc. very quickly if you set the `plotOptions` subproperty as `stacking: 'normal'`. This means this property is responsible for stacking each series on top of each other.

```

  plotOptions: {
    column: {
      stacking: 'normal',
      dataLabels: {enabled: true }
    }
  }

```

For better understating, add labels into each series group. For this, see the following code. In y-axis, you use the `stackLabels` property, and then the subproperty, which is

```

yAxis: {
  min: 0,
  title: {
    text: 'Total Hour studied'
  },

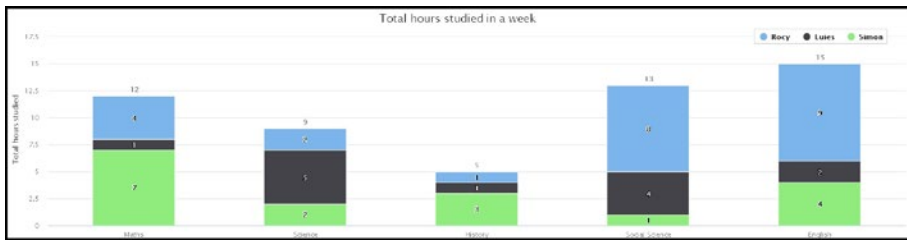
```

```

stackLabels: {
  enabled: true,
  style: {
    fontWeight: 'bold',
    color: ( // theme
      Highcharts.defaultOptions.title.style &&
      Highcharts.defaultOptions.title.style.
      color ) || 'gray'
  }
}
}
}

```

Set `enabled: true` so the labels are in each series. Then add some styling for the text of the labels. Figure 4-15 shows the output of Listing 4-14.



**Figure 4-15.** Stacked bar chart with columns

## Column Pyramid Charts

A column pyramid chart is a kind of column chart, only it looks like a pyramid. Pyramid charts are designed for comparing data with discrete data with values instead of categories.

To develop a simple pyramid chart, these dependencies are required:

**For jQuery users:**

```

<script src="https://code.highcharts.com/highcharts-more.js">
</script>

```

**For Angular users:**

```
import More from 'highcharts/highcharts-more';
More(Highcharts);
```

Now let's create a pyramid chart. See Listing 4-15.

**Listing 4-15.** app.component.ts

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. import More from 'highcharts/highcharts-more';
4. More(Highcharts);
5. @Component({
6.   selector: 'app-root',
7.   templateUrl: './app.component.html',
8.   styleUrls: ['./app.component.css']
9. })
10. export class AppComponent {
11.   title = 'myHighChartsApp';
12.   highcharts = Highcharts;
13.   chartOptions = {
14.     chart: {
15.       type: 'columnpyramid'
16.     },
17.     title: {
18.       text: 'Height of different students in a class'
19.     },
20.     colors: ['red', 'blue', 'green', 'yellow', 'pink'],
21.     xAxis: {
22.       type: 'category',
23.       crosshair: true,
24.       labels: {
```

## CHAPTER 4 DIFFERENT CHARTING TYPES

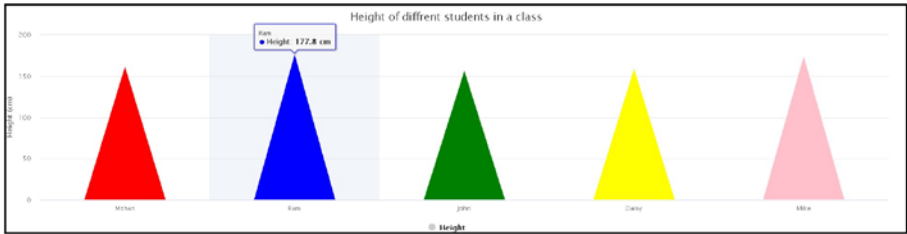
```
25. style: {
26.   fontSize: '10px'
27. }
28. },
29. },
30. yAxis: {
31.   min: 0,
32.   title: {
33.     text: 'Height (cm)'
34.   }
35. },
36. tooltip: {
37.   valueSuffix: ' cm'
38. },
39. series: [{
40.   name: 'Height',
41.   colorByPoint: true,
42.   showInLegend: true,
43.   data: [
44.     ['Mohan', 162.56],
45.     ['Ram', 177.8],
46.     ['John', 157.48],
47.     ['Daisy', 160],
48.     ['Mike', 175.5]
49.   ],
50. }]
51. };
52. }
```

In this code, the chart type is `columnpyramid`. This code creates a series of students' heights. In this code, as you can see, all pyramids are in a different color.

For this, in the series section, a subproperty is

```
colorByPoint: true
```

If you set this property as false, all series pyramids will be the same color. Now type `ng serve` and you will get output shown in Figure 4-16.



**Figure 4-16.** Simple column pyramid chart

You can also develop a stacked column pyramid chart very quickly using Highcharts. Listing 4-16 calculates the total hours spent by a student in a week on a particular subject.

**Listing 4-16.** `app.component.ts`

```
1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. import More from 'highcharts/highcharts-more';
4. More(Highcharts);
5. @Component({
6.   selector: 'app-root',
7.   templateUrl: './app.component.html',
8.   styleUrls: ['./app.component.css']
9. })
10. export class AppComponent {
11.   title = 'myHighChartsApp';
12.   highcharts = Highcharts;
```

## CHAPTER 4 DIFFERENT CHARTING TYPES

```
13. chartOptions = {
14. chart: {
15. type: 'columnpyramid'
16. },
17. title: {
18. text: 'Stacked columnpyramid chart'
19. },
20. xAxis: {
21. categories: ['Maths', 'Science', 'History', 'Social
    Science', 'English']
22. },
23. yAxis: {
24. min: 0,
25. title: {
26. text: 'Total Hour studied'
27. },
28. stackLabels: {
29. enabled: true,
30. style: {
31. fontWeight: 'bold',
32. color: 'gray'
33. }
34. }
35. },
36. legend: {
37. align: 'right',
38. x: -30,
39. verticalAlign: 'top',
40. y: 25,
41. floating: true,
42. backgroundColor: 'white',
```



```
43. borderColor: '#CCC',
44. borderWidth: 1,
45. shadow: false
46. },
47. tooltip: {
48. headerFormat: '<b>{point.x}</b><br/>',
49. pointFormat: '{series.name}: {point.y}<br/>Total: {point.
    stackTotal}'
50. },
51. plotOptions: {
52. columnpyramid: {
53. stacking: 'normal',
54. dataLabels: {
55. enabled: true,
56. color: 'white'
57. }
58. }
59. },
60. series: [{
61. name: 'Rocy',
62. data: [4, 2, 1, 8, 9]
63. }, { name: 'Luies', data: [1, 5, 1, 4, 2] },
64. {
65. name: 'Simon', data: [7, 2, 3, 1, 4]
66. }]
67. }
68. }
```

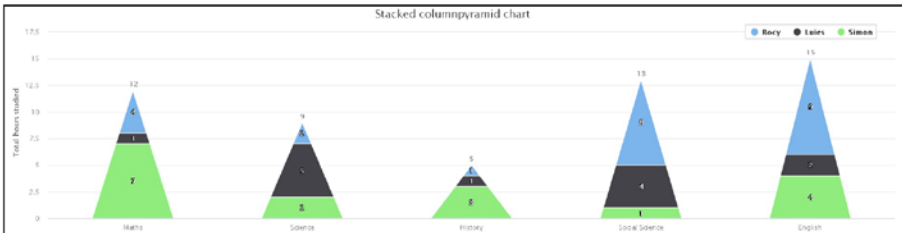
In Listing 4-16, the chart type is `columnpyramid`, but in `plotOptions`, note that `stacking: 'normal'`. This property helps create a stacked based chart in Highcharts.

```

plotOptions: {
  columnpyramid: {
    stacking: 'normal',
    dataLabels: {
      enabled: true,
      color: 'white'
    }
  }
},
},

```

If you run the code, you will get Figure 4-17.



**Figure 4-17.** Stacked column pyramid chart

## Gauge Charts

A gauge chart is also known as a dial chart or a speedometer. These types of charts read the needle on the dial. These types of charts provide great visualization for a dashboard. Gauge charts are mostly used by aircraft pilots. Let’s take a look at how you can implement a gauge chart using Highcharts and Angular. See Listing 4-17.

**Listing 4-17.** app.component.ts

```

1. import { Component } from '@angular/core';
2. import * as Highcharts from 'highcharts';
3. import More from 'highcharts/highcharts-more';

```

```
4. import solidGauge from "highcharts//modules/solid-gauge.js";
5. More(Highcharts);
6. solidGauge(Highcharts);

7. @Component({
8.   selector: 'app-root',
9.   templateUrl: './app.component.html',
10.  styleUrls: ['./app.component.css']
11. })
12. export class AppComponent {
13.   title = 'myHighChartsApp';
14.   highcharts = Highcharts;
15.   chartOptions = {
16.     chart: {
17.       type: 'gauge',
18.       plotBorderWidth: 0,
19.       plotShadow: false
20.     },
21.     title: {
22.       text: 'Speedometer'
23.     },
24.     pane: {
25.       startAngle: -150,
26.       endAngle: 150,
27.       background: [{
28.         backgroundColor: {
29.           linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
30.           stops: [
31.             [0, '#FFF'],
32.             [1, '#333']
33.           ]
34.         },
```

## CHAPTER 4 DIFFERENT CHARTING TYPES

```
35. borderWidth: 0,
36. outerRadius: '109%'
37. }, {
38. backgroundColor: {
39.   linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
40.   stops: [
41.     [0, '#333'],
42.     [1, '#FFF']
43.   ]
44. },
45. borderWidth: 1,
46. outerRadius: '107%'
47. }, {
48.   // default background
49. }, {
50.   backgroundColor: '#DDD',
51.   borderWidth: 0,
52.   outerRadius: '105%',
53.   innerRadius: '103%'
54. }]
55. },
56. yAxis: {
57.   min: 0,
58.   max: 200,
59.   minorTickInterval: 'auto',
60.   minorTickWidth: 1,
61.   minorTickLength: 10,
62.   minorTickPosition: 'inside',
63.   minorTickColor: '#666',
64.   tickPixelInterval: 30,
65.   tickWidth: 2,
```

```
66. tickPosition: 'inside',
67. tickLength: 10,
68. tickColor: '#666',
69. labels: {
70.   step: 2,
71.   rotation: 'auto'
72. },
73. title: {
74.   text: 'km/h'
75. },
76. plotBands: [{
77.   from: 0,
78.   to: 120,
79.   color: '#55BF3B' // green
80. }, {
81.   from: 120,
82.   to: 160,
83.   color: '#DDDF0D' // yellow
84. }, {
85.   from: 160,
86.   to: 200,
87.   color: '#DF5353' // red
88. }]
89. },
90. plotOptions: {
91.   solidgauge: {
92.     dataLabels: {
93.       y: 5,
94.       borderWidth: 0,
95.       useHTML: true
96.     }

```

```

97. }
98. },
99. series: [{
100. name: 'Speed',
101. data: [60],
102. tooltip: {
103. valueSuffix: ' km/h'
104. }
105. }]
106. };
107. }

```

Let's understand Listing 4-17. The chart type is gauge. The pane section is

```

pane: {
  startAngle: -150,
  endAngle: 150,
  background: [{
    backgroundColor: {
      linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
      stops: [
        [0, '#FFF'],
        [1, '#333']
      ]
    },
    borderWidth: 0,
    outerRadius: '109%'
  }, {
    backgroundColor: {
      linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1 },
      stops: [

```

```

        [0, '#333'],
        [1, '#FFF']
    ]
},
borderWidth: 1,
outerRadius: '107%'
}, {
    // default background
}, {
    backgroundColor: '#DDD',
    borderWidth: 0,
    outerRadius: '105%',
    innerRadius: '103%'
}]
},

```

In the gauge, the `startAngle` from the start angle of the x-axis is given in degrees where 0 means north.

In the gauge, the `endAngle` of the x-axis is given in degrees where 0 is north.

The y-axis is where you set the minimum and maximum speed for the gauge:

```

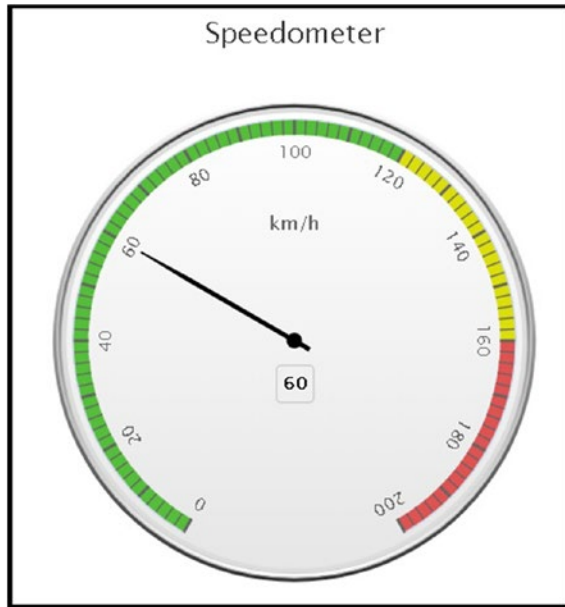
yAxis: {
    min: 0,
    max: 200,
    minorTickInterval: 'auto',
    minorTickWidth: 1,
    minorTickLength: 10,
    minorTickPosition: 'inside',
    minorTickColor: '#666',

```

```
    tickPixelInterval: 30,  
    tickWidth: 2,  
    tickPosition: 'inside',  
    tickLength: 10,  
    tickColor: '#666',  
    labels: {  
      step: 2,  
      rotation: 'auto'  
    },  
    title: {  
      text: 'km/h'  
    },  
    plotBands: [{  
      from: 0,  
      to: 120,  
      color: '#55BF3B' // green  
    }, {  
      from: 120,  
      to: 160,  
      color: '#DDDF0D' // yellow  
    }, {  
      from: 160,  
      to: 200,  
      color: '#DF5353' // red  
    }]  
  },
```

Now you set `plotBands` from where to where based on speed and colors. Then you set your `plotOptions`, and then you define the series. In a later chapter, you will see dynamic gauge charts for your dashboard in detail, but for now, see [Figure 4-18](#).





**Figure 4-18.** Simple gauge chart using Highcharts and Angular

## Summary

In this chapter, you learned how to easily create different types of charts very quickly with the use of Highcharts. These charts are beneficial for your dashboard based on certain conditions. Some charts required extra dependencies such as stack bars, pyramids, heatmaps, and so on. Based on your JavaScript framework, Angular or jQuery, please add those dependencies first into your project, as mentioned. In the upcoming chapters, you will see more charting types, which will make your life more comfortable based on your customer requirements.

## CHAPTER 5

# Working with Real-Time Data

In this chapter, you will learn how to get real-time data from the server side and render it into Highcharts using Angular. To work on real-time data, you must configure and send a request to back-end services (such as a web API, WCF, Web Services, REST Services, etc.), which can fetch data from the server, and the response will provide data for a Highcharts series. This chapter will talk about web APIs, and how you can develop a web API using Visual Studio, which will consume this web API into your single page application for rendering real-time data.

## Web API

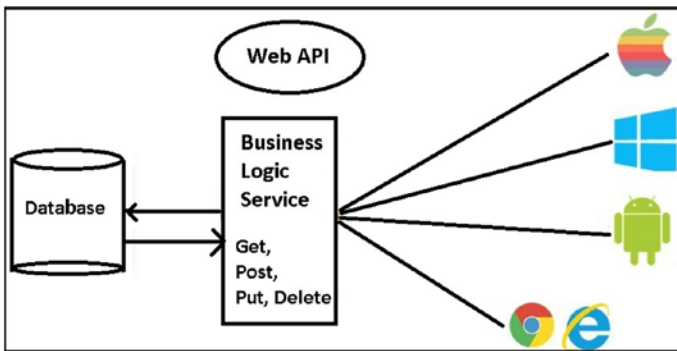
API means *application programming interface*, and a web API is a kind of business logic interface where users can consume and access methods (based on permissions) for specific features. These methods are called *resources*, and these methods are in the layer of HTTP verbs. A web API has four types of common verbs:

- HttpGet
- HttpPost
- HttpPut
- HttpDelete

They correspond to

- Read
- Insert (Create)
- Update
- Delete

Figure 5-1 explains in detail this request-response model in web API services.



**Figure 5-1.** *Web API framework*

For example, you write one method to access and share live market data, and you host your service. Now, if a different application wants to access your service for fetching data, it will consume that particular Web API, the one accessed by the HTTP protocol. As the name suggests, a web API works over the Web. A web API is an excellent framework because it reaches many clients; they can be accessed/consumed through browsers, IoT applications, and mobile devices very quickly.

Figure 5-1 shows three sections:

- Database
- Web API
- Browser and devices

The request comes from the browser or mobile app to the web API services. In a web API, all methods and business logic are written. If a requested resource is permitted to access a method, in the next step it will go to the database and fetch information, and the response will be returned in the form of XML or JSON based on your return type.

## What Is REST?

Rest stands for *representational states transfer*, and it is an architectural pattern. Roy Fielding firstly introduced REST in 2000 in his doctoral presentation. In REST, communication is always stateless. Here *stateless* means if you send one request, after getting a response, the relationship will break, so you must send a new request and get a new response. If you want to maintain your state, you can keep it on the client side. In REST, you can define multiple responses for the same resource method, so you can get a response in the form of JSON, XML, CSV, JPG, PDF, HTML, etc. REST uses the HTTP methods to operate resources in the form of get, put, post, and delete.

The following is an example of JSON format and it comes from a response:

```
{"studentId":1,"studentName":"ram","phone":982641***,"address":  
"delhi"}
```

Here each resource/method has one unique identifier, which will send a request to access a method and get the responses. For example,

- Get: <https://localhost:5001/api/getStudents/>
- Post: <https://localhost:5001/api/AddStudent/>
- Put: <https://localhost:5001/api/UpdateStudent/>
- Delete: <https://localhost:5001/api/DeleteStudent/>

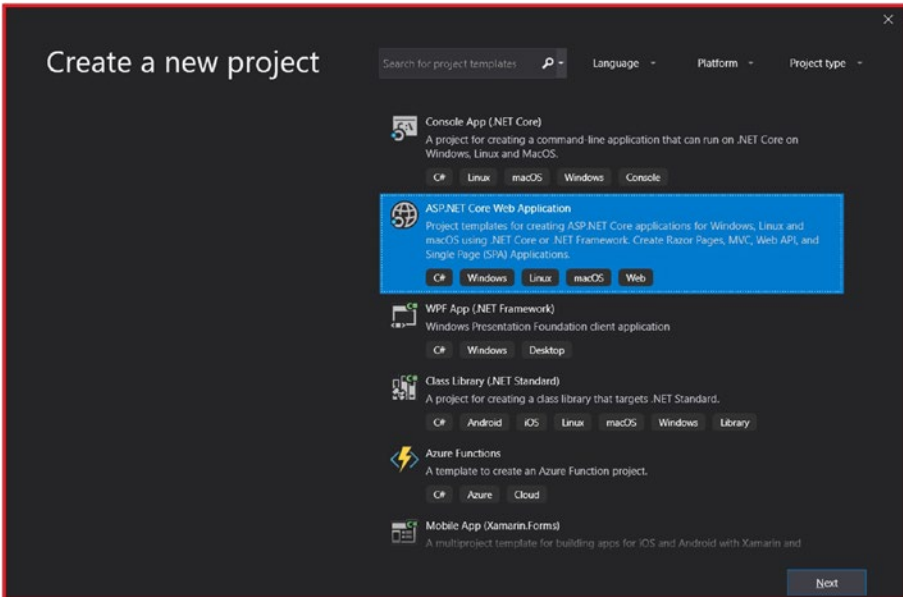
Now let's build a project with Angular and a web API using Highcharts.

## Web API Development Using Visual Studio

This application is pretty simple. You'll create a line chart based on student performance in particular subjects. For this application, you have two different apps.

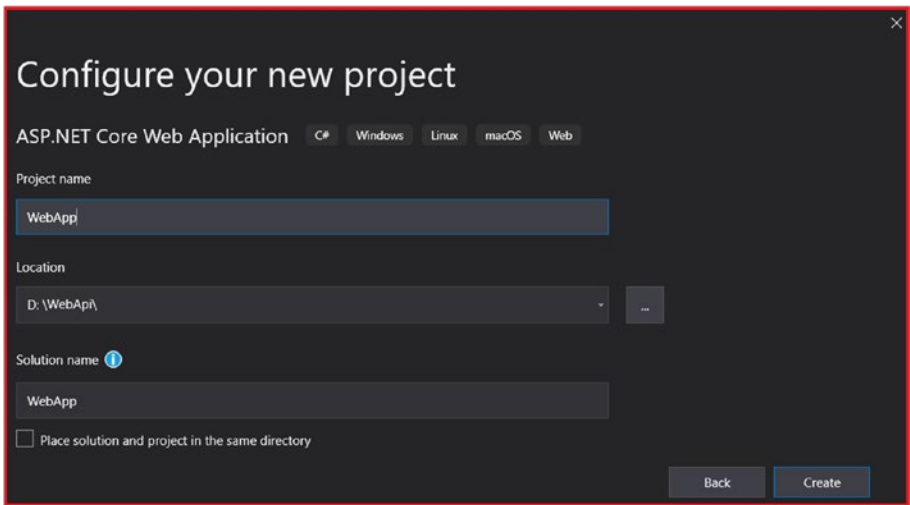
For the server side, you will write one web API service with a SQL Server database using Entity Framework. For the other side, you will use the Angular application you developed in Chapter 3. In this chapter, you will just enhance this application with a web API.

To create the web API application, open Visual Studio, and go to File ► New ► Project ► Select Asp.Net Core Web Application (Figure 5-2).



**Figure 5-2.** Creating a new web API using Visual Studio

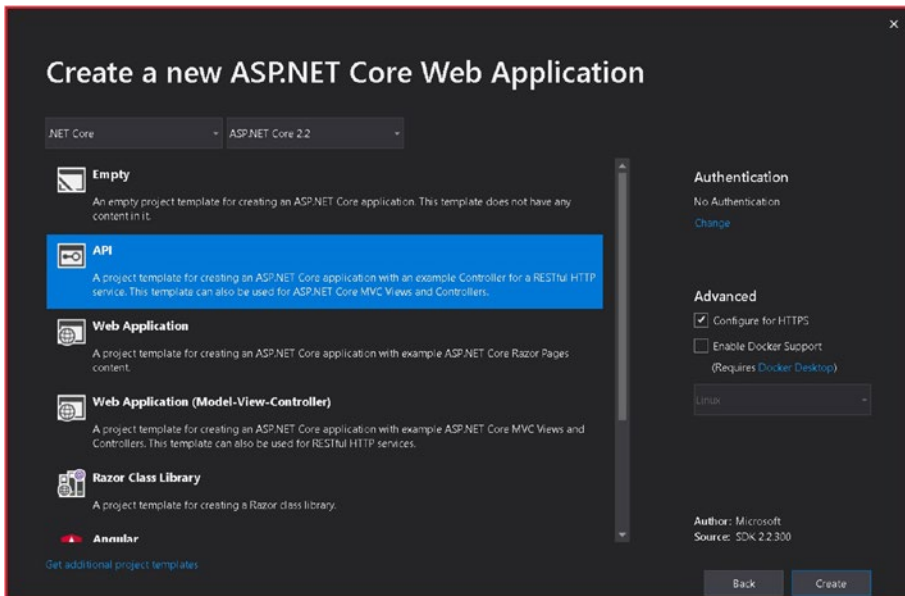
Click the Next button. You will get the screen shown in Figure 5-3.



**Figure 5-3.** *Configuring a new project*

Here you can set the project name and location path (which particular directory you want to save in) of your project. After you fill in these fields, click the Create button.

Next, you'll create a web API project (Figure 5-4), so choose an API and click the Create button. Your new web API will be created. Once your API has been created, you will see all related files and folders into Solution Explorer. So let's try to understand Solution Explorer and what files and folders comes with the new web API project.

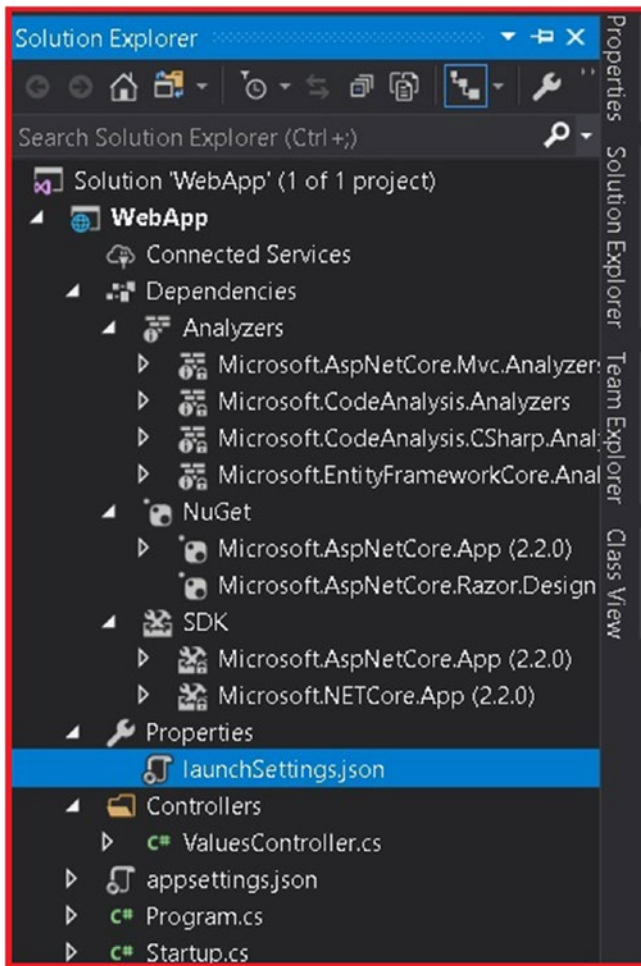


*Figure 5-4. Project template selection screen*

## Solution Explorer

In Visual Studio, Solution Explorer is the place where you see your project file structure. In one solution, you can see more than one project. Here you can add/remove files or projects and include/exclude files for your project.

Figure 5-5 shows Solution Explorer. If you are not able to see this in your Visual Studio, go to **View** ► **Solution Explorer**, and you will get the info shown in Figure 5-5.



**Figure 5-5.** Solution Explorer

Now, let's explore all the required files and folders one by one.

- **Dependencies:** This folder is designed for all project-related dependencies, such as NuGet related packages, project analyzer related, and SDK related. As per your requirements, you can add more dependencies here.



- **Properties:** This folder contains the `launchSettings.json` file. In this file, you can set project launch-related settings, which means when you press F5 to run your project, which particular API you want to run, you can place an environment variable.
- **Controller:** This is the most important folder for a web API because you add all API controllers here and it's where you write your methods and business logic.
- **appSettings.json:** All application-related settings are set here.
- **Program.cs:** All ASP.NET core-based projects are ignited from the console application. `Program.cs` is the execution point; you can see here the `Main` method. This method is connected with `startup.cs` and is required to run the app.
- **Startup.cs:** This file is responsible for all the configuration methods for the project. The following sections describe the required methods.

## ConfigureService()

You can find this method in `Startup.cs`. It's where you set all dependency injection-related kinds of stuff. Note that the .NET core comes with a built-in IOC container concept. Now no third party containers are required.

But what is dependency injection? In an object-oriented programming world, you create classes, and whenever you want to use these classes, you create objects. For example, you have a class called `Maths`:

```

public class Maths
{
public int Add(int a,int b)
{
int c=a+b;
}
}

```

For this Maths class, if you want to access its methods, you must create an object of the Maths class in this way:

```
var obj = new Maths();
```

If you want to access this object method, you must call it like

```
obj.Add(12,3);
```

So it is related to the object creation thing. If you don't want to create objects in this way, you go for IOC. So when you want to invert object creational stuff to someone else, it's called the IOC (inversion of control) pattern, and dependency injection is the way to implement IOC.

With the use of dependency injection, you don't need to create objects all the time. This gets taken care of automatically. One way of implementing dependency injection is to inject your object into the constructor level. In the upcoming chapters, you will explore this concept more.

## Configure()

This is one more method you can find in `Startup.cs`. In this method, you can set your application request pipeline, which means what comes first and what comes next. The .NET core is designed to make your application lightweight so you only call the required dependencies and requests here.

## Routing

In the browser, whenever you want to open a site, you need a URL. Every URL has a path, like [www.apress.com/in/about](http://www.apress.com/in/about). In this URL, `/in/about` is the address of a particular page. This is helpful for SEO (search engine optimization) purposes also. Here you don't require a map to a file, so routing is a concept where you send the request to a particular URL route. It will render a result in the form of a response.

## Attribute Routing

In the ASP.NET web API, you can do attribute routing very easily. This allows you to handle the exact route the user requested. If you open `ValuesController.cs`, you can see code like this:

```
[Route("api/[controller]")]
```

Here the `Route` class is decorating the way for the values controller, which means whenever you want to use the controller in your app, you have to call an API/controller name and then the method name and so on. You can also define this:

```
[Route("api/values")]
```

Now open `ValuesController.cs`, as shown in [Listing 5-1](#).

### **Listing 5-1.** `ValuesController.cs`

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using Microsoft.AspNetCore.Mvc;
```

```
namespace WebApp.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        public ActionResult<string> Get(int id)
        {
            return "value";
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }
    }
}
```

```
// DELETE api/values/5
[HttpDelete("{id}")]
public void Delete(int id)
    {
    }
}
}
```

To understand the `ValuesController.cs` file in more detail, see the following list:

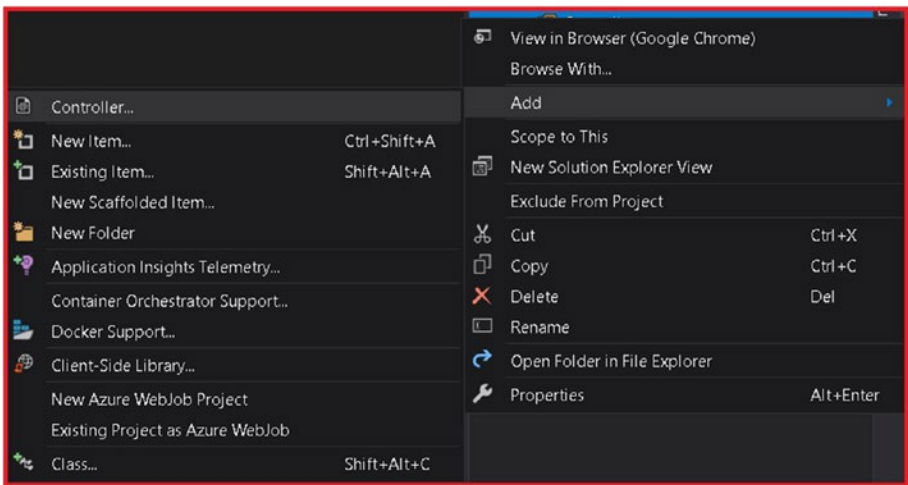
- `ControllerBase`: This is the base class for a controller without view support. `ControllerBase` provides many methods that are very useful for handling HTTP requests.
- `ApiController`: Controllers decorated with this attribute are configured with features and behavior targeted at improving the developer experience for building APIs. When decorated on an assembly, all controllers in the assembly will be treated as controllers with API behavior.
- `ActionResult`: `ActionResult` was introduced in ASP.NET core 2.1; it is the return type of the API controller actions. With the use of `ActionResult<type>`, you can return the kind of value defined in your method.

`ActionResult` works based on HTTP methods. The following are the list of methods used in a web API:

- `HttpGet`: Used to retrieve data. A successful get method returns 200 as status code (OK).

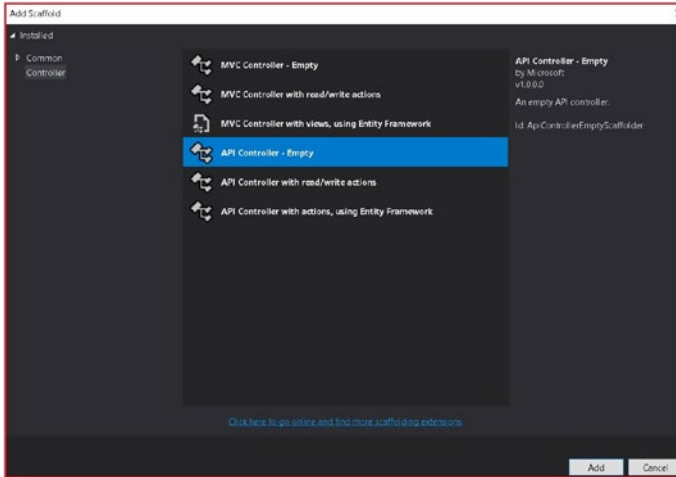
- **HttpPost:** Used for inserting/creating records. The Post method creates new resources and returns a status code of 201. If there is no result, it gives a status code of 204 (No Content). If there are any errors or a client request sends the wrong data, it will return a status code of 400, which is for a bad request.
- **HttpPut:** Used for updating records. It returns 201 as a status code. Here 204 mean no content in the output. 409 status codes are for a conflict.
- **HttpDelete:** Used for deleting the record. If the deletion is successful, it will return a status code of 204. If the resource does not exist, it will return a status code of 404 (Not Found).

Now it's time to create a new web API controller. Open Solution Explorer. Right-click in the Controller folder ➤ Add ➤ Controller (Figure 5-6).

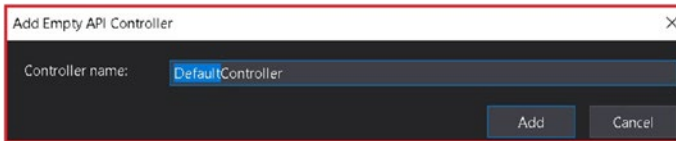


**Figure 5-6.** Adding a new controller

Select API Controller ► Empty and click the Add button (Figure 5-7). You will get the screen in Figure 5-8.



**Figure 5-7.** Adding a new API controller



**Figure 5-8.** Adding an empty API controller

Now it's time to give your controller a name. For this demo, use `StudentController`, and click the Add button (Figure 5-8). Once you click the Add button, you will get the code shown in Listing 5-2.

**Listing 5-2.** `StudentController.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace WebApp.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class StudentController : ControllerBase
    {
    }
}

```

Now, it's time to work on database activity, because you are going to create real-time data, fetch this data, and render it into Highcharts.

## Database Creation

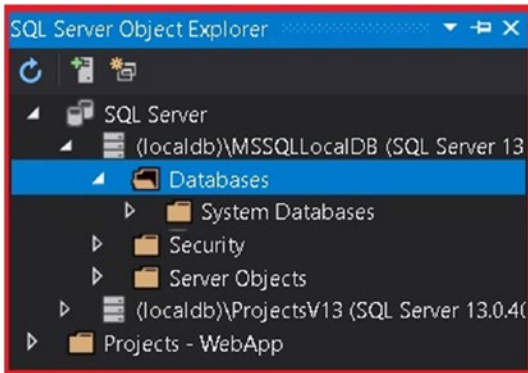
If you already have a database and tables, you can skip this section. For database creation, choose your database program as per your requirements, such as SQL Server, Oracle, MySQL, etc. The only thing you have to remember is the database server name for connecting and the database name.

In this chapter, I am going to use SQL Server, which comes with Visual Studio.

If you want to continue, go to View ► Open SQL Server Object Explorer (Figure 5-9).

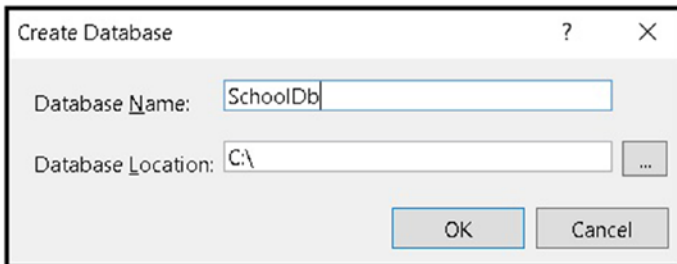
1. Open Local Db.





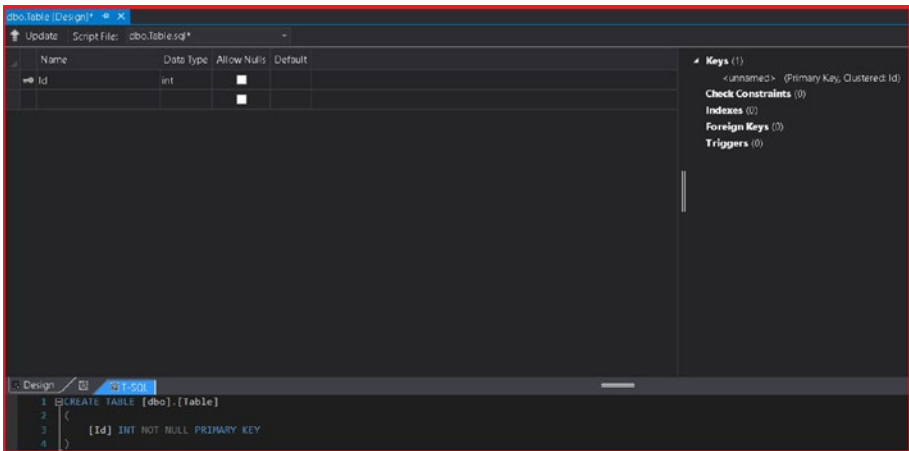
**Figure 5-9.** *SQL Server Object Explorer*

2. Right-click Databases, and select Add New Database.
3. Set the database path and name, and click the OK button (Figure 5-10).



**Figure 5-10.** *The Create Database screen (for setting the database name and path)*

4. You can see your database on the list. Open it. It will be empty, so let's create a new table.
5. To create a new table, right-click in front of the folder of tables. Add new tables (Figure 5-11).



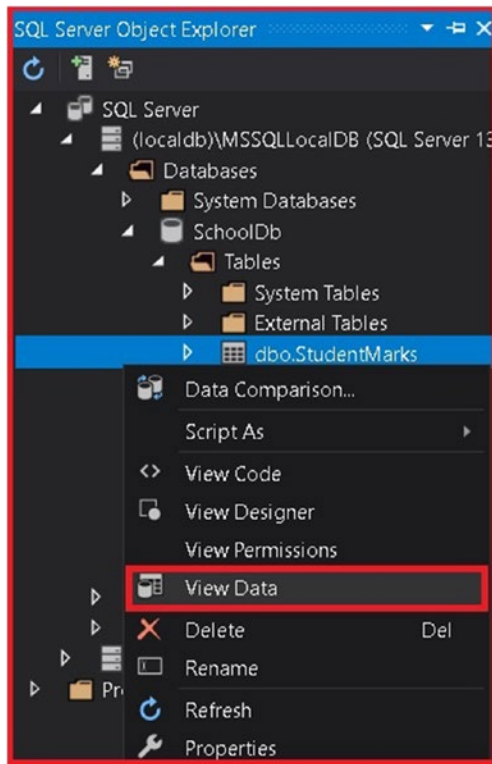
**Figure 5-11.** *Creating a table through code*

- Paste the following code and click the Update button:

```

CREATE TABLE [dbo].[StudentMarks](
    [Id]                INT IDENTITY (1, 1) NOT NULL,
    [Name]              NVARCHAR (50) NULL,
    [English]          INT NULL,
    [Maths]            INT NULL,
    [Science]          INT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
  
```

- Once you click the Update button, you will see a new table called StudentMarks.
- Now it's time to add some data. Right-click in front of the newly created table, and choose View Data (Figure 5-12).



**Figure 5-12.** Adding a record process into a table

9. Now you will get a row- and column-based screen to add records, so add your data in the form of rows and columns. Your table data will be stored. See Figure 5-13.

	Id	Name	English	Maths	Science
>		Ram	48	41	49
	2	James	47	49	41
	3	Jack	42	38	33
	4	Vinni	35	32	42
⊕	NULL	NULL	NULL	NULL	NULL

**Figure 5-13.** Inserting new records into a table

Now it's time to add Entity Framework to your project.

## Adding Entity Framework

Entity Framework is an ORM (object-relational mapping) framework; it gives users an automated mechanism for fetching and storing data into the database. Entity Framework is the extended version of ADO.NET.

In this demo, you already have one database and table so this situation will use a database first approach. This is the approach developers choose when their database tables and stored procedures are already there, and they want to consume them.

So for the database first approach, you must run the Scaffold-DbContext command in the Package Manager console window. Go to Tools ► NuGet Package Manager ► Package Manager Console. Run the following command:

```
Scaffold-DbContext "Server=(localdb)\MSSQLLocalDB;Database=SchoolDb;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Note that you must change the database name, server name, and trusted connection based on your requirements.

The following is a description of the above command:

- `Scaffold-DbContext`: This command produces an Entity Framework model for an existing database.
- `Server`: Database server name
- `Database`: The name of the database at the time of creation, such as `SchoolDb`
- `Trusted_Connection`: This is for using a Windows security trusted connection. If you want to set your database id and password, you have to remove this.
- `-OutputDir`: Name of the folder where you want to add your model classes and `DbContext` files

Once you run this command, it will generate two files for you in the `Models` folder:

- `StudentMarks.cs`
- `StudentDbContext.cs`

`StudentMarks.cs` is the model class for a table. In this demo, you created only one table, so you will get only one model class. If you create more tables based on your requirements, you will get more table files.

`StudentDbContext.cs` is related to table relationship mappings and a complete model of the database.

Now it's time to do some coding in the web API to fetch data from the table. Open `StudentController.cs` and copy the code in Listing 5-3 into it.

**Listing 5-3.** `StudentController.cs`

```
using System;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using WebApp.Models;
```

```

namespace WebApp.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class StudentController : ControllerBase
    {
        SchoolDbContext schoolDbContext = new SchoolDbContext();
        public ActionResult GetStudents()
        {
            var query = schoolDbContext.StudentMarks.ToList();
            return Ok(query);
        }
    }
}

```

If you look at the code for `StudentController.cs`, one object is created for `SchoolDbContext`, which is responsible for connecting with the DB and all tables, views, and stored procedures.

```
SchoolDbContext schoolDbContext = new SchoolDbContext();
```

Next, there's a method called `GetStudents()`. At the time of calling, this method will fetch the data from the db.

```

public ActionResult GetStudents()
{
    var query = schoolDbContext.StudentMarks.ToList();
    return Ok(query);
}

```

So here you create one variable named `query` and fetch the `StudentMarks` list into this variable. After that, there is a `return OK(query)`, which means this will return an OK Negotiated Content result.

Now, it's time to run the web API. If you want to make this student controller method as your default API, it means whenever you run this project, it runs automatically, as do the below changes.

Go to Solution Explorer ► Properties ► Open `launchSettings.json` file and add the changes in Listing 5-4 to the `launchUrl` section.

**Listing 5-4.** `launchSettings.json`

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:54066",
      "sslPort": 44396
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "api/student",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "WebApp": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "api/student",
```

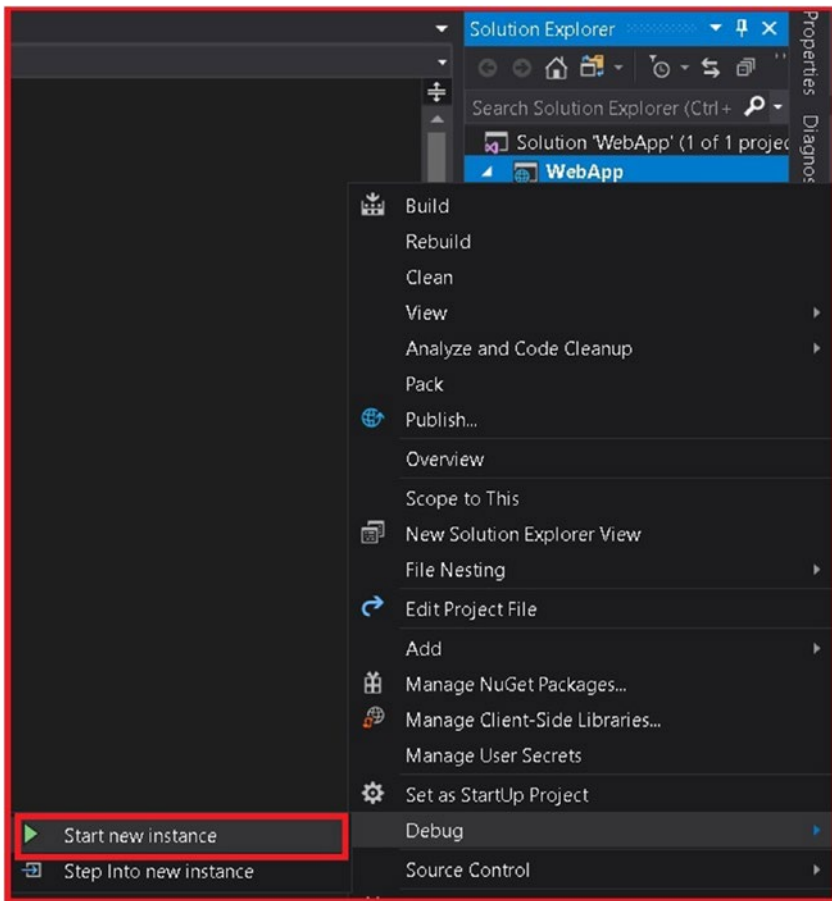
```
"applicationUrl": "https://localhost:5001;http://localhost:5000",  
"environmentVariables": {  
  "ASPNETCORE_ENVIRONMENT": "Development"  
}  
}  
}
```

In the code for `launchSettings.json`, note the change from `launchUrl` to `"api/student"`.

```
"launchUrl": "api/student",
```

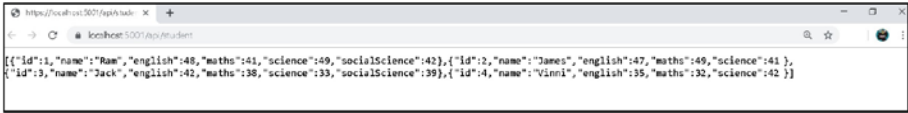
The above step is totally up to you. If you want to skip this step, you can run your web API directly and change it in the browser address bar. To run your web API, go to Solution Explorer and right-click project ► Debug ► Start new instance. See [Figure 5-14](#).





**Figure 5-14.** Running the web API in a Visual Studio project

Once you click Start new instance, it will run your project. Once you run the project, you will get the output in Figure 5-15.



**Figure 5-15.** `api/student/getStudent` returns data in the form of JSON

As you can see in Figure 5-15, you got your response in the JSON format. Your API is running correctly, so you can fetch your data also. Now it's time to develop your Angular app to render the API data into Highcharts.

## Angular-Highcharts UI Application

In Chapter 3, you created one Angular app. Here you'll continue with the same project, but you'll make some changes. So let's start to work on the Angular-Highcharts UI application.

### Services in Angular

Services are code that can be accessed from multiple components. A service is defined for a purpose. If you want to make your code modularize and reusable, services are the best option. Services in Angular can be used as a repository. In the repository pattern, each service class is responsible for one purpose. For example, `StudentService` is responsible for student-related functions like `AddStudent()`, `UpdateStudent()`, `GetStudentDetails()`, etc.

Let's create a new service in Angular. Open the `myFirstAngularHighchart` application you created in Chapter 3. Now run the following command in a Visual Studio code terminal window to create a service:

```
ng generate service service/studentService
```

This command will generate a new service with the name of `studentService` in the service folder (here the service folder is automatically created by this command).

Open `studentservice.service.ts` from the service folder and you will get the code in Listing 5-5.

**Listing 5-5.** `studentservice.service.ts`

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StudentServiceService {
  constructor() { }
}
```

Let's review the code. In the first line, you can see an `Injectable` decorator. For this, we are using `import` for `Injectable`. `import { Injectable } from '@angular/core'`.

You can see in the `@Injectable` decorator I made as a root injector (`providedIn: 'root'`), which means this particular service is accessible from any component or any service level for this application.

Now you will create one model class to bind the data from the web API response. To create a model class, run the following command in Visual Studio:

**ng generate class model/marksModel**

This command will generate one model class for you. Now paste the code in Listing 5-6 into the file.

**Listing 5-6.** marks-model.ts

```
1. export class MarksModel {
2.   english : string;
3.   maths:string;
4.   science:string;
5.   name: string;
6. }
```

In this code, there's one class called `MarksModel` and four properties. You can add more properties based on your requirements.

Now go back to the `Studentservice.service.ts` class. You will send a request to the web API project and get a response in the `MarksModel` class. Copy the complete code in Listing 5-7 into `studentservice.service.ts`.

**Listing 5-7.** studentservice.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { HttpClient } from '@angular/common/http';
3. import { MarksModel } from '../model/marks-model';
4. import { Observable } from 'rxjs';
5. @Injectable({
6.   providedIn: 'root'
7. })
8. export class StudentserviceService {
9.   constructor(private http: HttpClient) {
10.    console.log('StudentserviceService called');
11.   }
12.   Get(url):Observable<MarksModel[]>{
13.    return this.http.get<MarksModel[]>(url);
14.   }
15. }
```

Now let's understand the code. Here you add `HttpClient`. The `HttpClient` module is required for sending the request to the web API, so you add the following line:

```
import {HttpClient} from '@angular/common/http'
```

In the next line, you call `MarksModel`; as you know, you just created one `Model` class. In the next line, you call `Observable` from `RxJs`.

`Observable` helps your application pass messages between publishers and subscribers. Here a method never executes until consumers subscribe to it. `Observable` can handle any type of value like messages, literals, events, etc.

`Observable` is used to retrieve data. `Observable` helps handle asynchronous data, such as data coming from a back-end database or service. Here events are treated as a collection.

`RxJs` stands for Reactive Extensions for JavaScript, and it is used for asynchronous programming using `Observable`. With `RxJs` you can work on the server side with `Node.js` or on the browser side. Here *asynchronous* means you will call your method and register for notifications when results are available, so with this approach, your web page will never become unresponsive.

Now look at the constructor level. You inject `HttpClient`, (this is the best example of dependency injection in Angular) so you can send a get request to the web API.

```
constructor(private http: HttpClient) {  
  }  
}
```

Then you have a `Get` method where you send the `http.get` request. There is a `Get(url)` method with one parameter, `url`. This `url` comes from the caller, which is a component in this application.

Once this method gets its `url`, it will send a `http.get` request to the web API. The method return type is `MarksModel[]` array because from a web API you get data in the form of a collection, which is why you're using

an array here. In this case, the URL will be `https://localhost:5001/api/student`.

**(Note: Your port number may change, so please provide the proper port number. Refer to Figure 5-15.)**

In Figure 5-15, you can see the URL of `https://localhost:5001/api/student`. So if you want to access the student API, you have to call this particular URL.

```
Get(url):Observable<MarksModel[]>{
  return this.http.get<MarksModel[]>(url);
}
```

Now it's time to call `httpClient` into the `app.module.ts` level, so add `HttpClientModule` because it's required to access the web API. Copy the code in Listing 5-8 into the `app.module.ts` file.

**Listing 5-8.** `app.module.ts`

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { HttpClientModule } from '@angular/common/http';
4. import { AppRoutingModule } from './app-routing.module';
5. import { AppComponent } from './app.component';
6. import { HighchartsChartComponent } from 'highcharts-angular';
7. @NgModule({
8.   declarations: [
9.     AppComponent,
10.    HighchartsChartComponent,
11.   ],
12.   imports: [
```

```

13. BrowserModule,
14. HttpClientModule,
15. AppRoutingModule
16. ],
17. //providers: [StudentserviceService],
18. bootstrap: [AppComponent]
19. })
20. export class AppModule { }

```

Now, it's time to work on the component level. Copy the code in Listing 5-9 into the `app.component.ts` file.

**Listing 5-9.** `app.component.ts`

```

1. import { Component } from '@angular/core';
2. import { MarksModel } from '../model/marks-model';
3. import { StudentserviceService } from '../service/
  studentservice.service';
4. import * as Highcharts from 'highcharts';
5. @Component({
6.   selector: 'app-root',
7.   templateUrl: '../app.component.html',
8.   styleUrls: ['../app.component.css']
9. })
10. export class AppComponent {
11.   studentModel: MarksModel[];
12.   url: string = 'https://localhost:5001/api/student';
13.   studentNames: any;
14.   constructor(private studentservice: StudentserviceService) {
15.   }
16.   public options: any = {

```

```
17. chart: {
18.   type: 'line',
19. },
20. title: {
21.   text: 'Real Time Data Example'
22. },
23. credits: {
24.   enabled: false
25. },
26. xAxis: {
27.   categories: ['English', 'Maths', 'Science']
28. },
29. yAxis: {
30.   title: {
31.     text: 'Marks'
32.   },
33. },
34. series: [],
35. }
36. ngOnInit() {
37.   this.getApiResponse(this.url).then(
38.     data => {
39.       const subjectMarks = [];
40.       const names = [];
41.       data.forEach(row => {
42.         const temp_row = [
43.           row.english,
44.           row.maths,
45.           row.science
46.         ];
47.         names.push(row.name);
```



```
48. subjectMarks.push(temp_row);
49. });
50. this.studentModel = subjectMarks;
51. this.studentNames = names;
52. var dataSeries = [];
53. for (var i = 0; i<this.studentModel.length; i++) {
54. dataSeries.push({
55. data: this.studentModel[i],
56. name: this.studentNames[i]
57. });
58. }
59. this.options.series = dataSeries;
60. Highcharts.chart('container', this.options);
61. },
62. error => {
63. console.log('Something went wrong.');
```

```
64. })
65. }
66. getApiResponse(url) {
67. return this.studentservice.Get(this.url)
68. .toPromise().then(res => {
69. return res;
70. });
71. }
72. }
```

Let's try to understand Listing 5-9 line by line. This code is an enhancement of the component you developed in Chapters 3 and 4. You import `MarksModel` and `studentserviceService` for reference.

```
import { MarksModel } from './model/marks-model';
import { StudentService } from './service/
studentService.service';
```

Next are three variables: `studentModel`, `url`, and `studentNames`. `studentModel` is for fetching marks data from service, `url` is for sending the request to the web API, and `studentNames` for collecting names.

```
studentModel: MarksModel[];
url: string = 'https://localhost:5001/api/student';
this.studentNames = names;
```

Then you inject (an example of dependency injection) `studentService` into a constructor.

```
constructor(private studentService: StudentService) {
}
```

Then you write code for Highcharts, where you define all basic properties for the chart in the options.

In the last section of code, you create one method with the name of `getApiResponse(url)`; this method is responsible for sending the request to `studentService Get()` method. See the following code:

```
getApiResponse(url) {
return this.studentService.Get(this.url)
    .toPromise().then(res => {
return res;
    });
}
```

Now let's talk about the `ngOnInit()` function. This function starts to run in the Angular lifecycle when the component load completes, so you can say it's a page load kind of event method. Once `app.component.ts` loads, this method will start to execute.

```
ngOnInit() {
  this.getApiResponse(this.url).then(
    data => {
      const subjectMarks = [];
      const names = [];
      data.forEach(row => {
        const temp_row = [
          row.english,
          row.maths,
          row.science
        ];
        names.push(row.name);
        subjectMarks.push(temp_row);
      });

      this.studentModel = subjectMarks;
      this.studentNames = names;
      var dataSeries = [];
      for (var i = 0; i < this.studentModel.length; i++) {
        dataSeries.push({
          data: this.studentModel[i],
          name: this.studentNames[i]
        });
      }
      this.options.series = dataSeries;
      Highcharts.chart('container', this.options);
    },
    error => {
      console.log('Something went wrong. ');
    }
  )
}
```

In the `ngOnInit()` code, you call the `getApiResponse(url)` method. So this method calls `studentService` and gets a response with the use of the `array.ForEach` loop, storing records one by one into `temp_row` variable.

In the next step, you push this list into the `studentNames` and `studentMarks` array type variables. In the `dataSeries`, which is the most important part for constructing a chart, you define the data and name properties, so the data property will construct in this way. See the following code:

```
[{
name: 'StudentName1',
data: [marks1, marks2, marks3]
}, {
name: 'StudentName2',
data: [marks1, marks2, marks3]
}, {
name: 'StudentName3',
data: [marks1, marks2, marks3]
}, {
name: 'StudentName4',
data: [marks1, marks2, marks3]
}],
```

And then you define `options.series`, and in the next line, call `Highcharts.chart` to construct a 'container `<div>`'.

Now it's time to write code into `app.component.html`. So copy the code in Listing 5-10 into the `app.component.html` file.

**Listing 5-10.** `app.component.html`

1. `<div class="content" id="container" role="main">`
2. `</div>`
3. `<router-outlet></router-outlet>`

Now type `ng serve` and press Enter. Type `localhost:4200` in your browser and press Enter. At this point, you will get no output. Press F12 to troubleshoot this problem. Once you press F12 in your browser, you will get the screen in Figure 5-16.



**Figure 5-16.** Press F12 to troubleshoot CORS issues

Figure 5-16 shows this as an issue of CORS, which stands for *cross-origin resource sharing*. Whenever you send the request in two different resources or projects, you get this problem.

Here two different projects mean one project for Web API and one project for an Angular app. So CORS is a kind of approach that allows restricted resources on a web page.

If you see the browser error, it clearly says Access to XMLHttpRequest at `http://localhost:5001/api/student` (this URL is from the web API project) from origin `http://localhost:4200` (this is from the Angular app) has been blocked by CORS policy because no `access-control-allow-origin` header is present on the requested resource.

So for the requested resource, which is the web API, you have to add a CORS policy. For this, you permit `localhost:4200` because this is your Angular app URL. So stop your web API project and open `Startup.cs` file and change the code.

First, you change into the `ConfigureService(IServiceCollection service)` method. Change the same in your code also.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
    services.AddMvc()
        .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Add the `services.AddCors()` method as the first step. In the next part, change your code to the `Configure()` method.

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
    {
    if (env.IsDevelopment())
        {
        app.UseDeveloperExceptionPage();
        }
    else
        {
        app.UseHsts();
        }

    app.UseCors(
options =>options.WithOrigins("http://localhost:4200").
AllowAnyMethod(
                );

    app.UseHttpsRedirection();
    app.UseMvc();
    }
```

You add `app.UseCors`, and in the lambda expression, you add a URL (you can use any URL based on your requirements).

For your complete reference, Listing 5-11 contains the full code of `Startup.cs`. If you face any issue, paste in the entire code of `Startup.cs`.

**Listing 5-11.** Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace WebApp
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to
        // add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCors();
            services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.
            Version_2_1);
        }

        // This method gets called by the runtime. Use this method to
        // configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app,
            IHostingEnvironment env)
        {
```

```

if (env.IsDevelopment())
    {
app.UseDeveloperExceptionPage();
    }
else
    {
app.UseHsts();
    }
app.UseCors(
options =>options.WithOrigins("http://localhost:4200").
AllowAnyMethod()
    );
app.UseHttpsRedirection();
app.UseMvc();
    }
}
}

```

Now run your Web API project again and run the Angular app. You will get the output shown in Figure 5-17.



**Figure 5-17.** Real-time line chart with a back-end web API



Current data is coming from a back-end service. To call a back-end service, it's not necessary to call the web API. You can send the request to a PHP service or any web service you prefer; all you need is a URL. If your URL is correct and if you know the response type, you can fetch data very easily.

## Events in Highcharts

Events are the essential part of an application. With the use of these events, the app can react to actions like click events, mouse-over events, load events, legendItemClicks, etc. In this section, I will talk about events. An event can be generated in Highcharts. Through an event property, let's see an example. This example continues the last example. Once the user clicks the chart lines series, it will show an alert with the student subject, marks, and name. Copy the code in Listing 5-12 into the `app.component.ts` file.

### *Listing 5-12.* `app.component.ts`

```
import { Component } from '@angular/core';
import { MarksModel } from './model/marks-mode';
import { StudentServiceService } from './services/
studentService.service';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  studentModel: MarksModel[];
```

```

url: string = 'https://localhost:5001/api/student';
studentNames: any;

constructor(private studentservice: StudentService) {
    }

public options: any = {
chart: {
type: 'line',
    },
title: {
text: 'Real Time Data Example'
    },
credits: {
enabled: false
    },
xAxis: {
categories: ['English', 'Maths', 'Science'],
    },
yAxis: {
title: {
text: 'Marks'
    },
    },

plotOptions: {
series:{
    }
    },
series: [],
}

```

```

ngOnInit() {
  this.getApiResponse(this.url).then(
    data => {
      const subjectMarks = [];
      const names = [];
      data.forEach(row => {
        const temp_row = [
          row.english,
          row.maths,
          row.science
        ];
        names.push(row.name);
        subjectMarks.push(temp_row);
      });

      this.studentModel = subjectMarks;
      this.studentNames = names;
      var dataSeries = [];
      for (var i = 0; i<this.studentModel.length; i++) {
        dataSeries.push({
          data: this.studentModel[i],
          name: this.studentNames[i],

          });
      }
      this.options.series = dataSeries;
      this.options.plotOptions.series= {
        point: {
          events: {
            click: function () {
              alert('Name: '+this.series.name+', Subject: ' +
                this.category + ', Marks: ' + this.y);
            }
          }
        }
      }
    }
  );
}

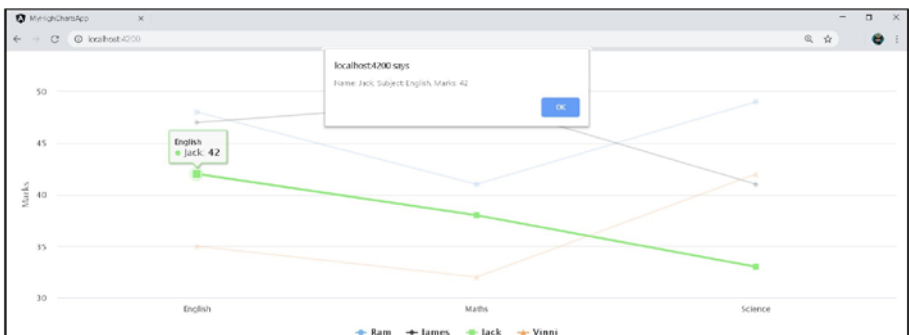
```

```

    }
  }
}
}
Highcharts.chart('container', this.options);
},
error => {
console.log('Something went wrong. ');
})
}
getApiResponse(url) {
return this.studentservice.Get(this.url)
  .toPromise().then(res => {
return res;
});
}
}
}

```

Run Listing 5-12's code with `ng serve`, and you will get the output in Figure 5-18. Click any series and you will get an alert with the name, subject, and marks of a student.



**Figure 5-18.** Chart series with a `plotOptions.series.click` event

Now, let's try to understand Listing 5-12's code. As you know, this is the continuation of the last demo, but here you add a few new things to generate click events. First, you add the following code into options area for creating Highcharts:

```
plotOptions: {
  series:{
    }
  }
}
```

Then in the `ngOnInit()` method area, you add the following code:

```
this.options.plotOptions.series= {
  point: {
    events: {
      click: function () {
        alert('Name: ' + this.series.name + ', Subject: ' + this.
          category + ', Marks: ' + this.y);
      }
    }
  }
}
```

In the plot options series, you add a point, and then you add events, such as a click event. For this click event, you have the method, so once a user clicks a series point, this function will activate.

So after clicking the series, it will display the name of a student, subject, and marks in an alert box (Figure 5-18).

## Drilldown Event

A drilldown event fires when you click the chart, and you will get the detailing of that particular series. You can add this detailing into any chart with the use of the drilldown event. In the upcoming example, I am going to show

you three series of types of software like operating systems, programming languages, and browsers. When the user clicks one series, it will show how much that product is used worldwide. See the code in Listing 5-13 (add this code to `app.component.ts`).

**Listing 5-13.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

import Drilldown from 'highcharts/modules/drilldown';
Drilldown(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: 'column',
      events: {
        drilldown: function (e) {
          if (!e.seriesOptions) {
            var chart = this,
                drilldowns = {
                  ProgrammingLanguage: {
                    name: 'ProgrammingLanguage',
```

```

data: [
    ['C#', 60],
    ['Java', 40]
]
},
OperatingSystem: {
name: 'OperatingSystem',
data: [
    ['Windows', 75],
    ['Dos', 5],
    ['Unix', 20]
]
},
Browser: {
name: 'Browser',
data: [
    ['Chrome', 60],
    ['IE', 10],
    ['FireFox', 30]
]
}
},
series = drilldowns[e.point.name];

// Show the loading label
chart.showLoading('drilldown event called, Loading ...');

setTimeout(function () {
chart.hideLoading();
chart.addSeriesAsDrilldown(e.point, series);
    }, 500);
}

```

```
    }
  }
},
title: {
text: 'Software Products used Worldwide in Percentage'
},
legend: {
enabled: false
},
xAxis: {
type: 'category'
},
plotOptions: {
series: {
borderWidth: 0,
dataLabels: {
enabled: true
}
}
},
series: [{
name: 'Products',
colorByPoint: true,
data: [{
name: 'Browser',
y: 3,
drilldown: true
}, {
name: 'OperatingSystem',
y: 3,
```



```

drilldown: true
    }, {
name: 'ProgrammingLanguage',
y: 2,
drilldown: true
    }]
}],
drilldown: {
series: []
}
};
}

```

Let's go through the code. As you know, to work on the drilldown feature, you must import the Drilldown dependency.

```

import Drilldown from 'highcharts/modules/drilldown';
Drilldown(Highcharts);

```

In the next step, you must call a series.

```

series: [{
name: 'Products',
colorByPoint: true,
data: [{
name: 'Browser',
y: 3,
drilldown: true
    }, {
name: 'OperatingSystem',
y: 3,
drilldown: true
    }, {

```

```

name: 'ProgrammingLanguage',
y: 4,
drilldown: true
  }]
}],

```

Here you use a name property. This name becomes the id for that particular series for the detailing part, because in a drilldown you must show details of that specific chart. Now, you add the events method.

```

events: {
drilldown: function (e) {
if (!e.seriesOptions) {
var chart = this,
drilldowns = {
ProgrammingLanguage: {
name: 'ProgrammingLanguage',
data: [
          ['C#', 60],
          ['Java', 40]
        ]
      },
OperatingSystem: {
name: 'OperatingSystem',
data: [
          ['Windows', 75],
          ['Dos', 5],
          ['Unix', 20]
        ]
      },

```

```

        Browser: {
name: 'Browser',
data: [
            ['Chrome', 80],
            ['IE', 30],
            ['FireFox', 50]
        ]
    }
},
series = drilldowns[e.point.name];

    // Show the loading label
chart.showLoading('drilldown event called Loading ...');

setTimeout(function () {
chart.hideLoading();
chart.addSeriesAsDrilldown(e.point, series);
    }, 500);
}
}
}

```

Note that the name property is the same as the series, because in the series variable, you call `drilldownsmethod()`. This array provides the details of that particular series.

```
series = drilldowns[e.point.name];
```

Now copy the code in Listing 5-14 into the `app.component.html` file.

**Listing 5-14.** `app.component.html`

```

<div class="content" role="main">
<highcharts-chart [Highcharts]="highcharts"
[options]="chartOptions"

```

```

style="width: 100%; height: 400px; display: block;">
</highcharts-chart>

</div>

<router-outlet></router-outlet>

```

Type `ng serve` into a terminal window and your code will start running. See Figure 5-19.



**Figure 5-19.** Bar chart with drilldown event

Now click in the chart. You will get details of the particular series you clicked. In Figure 5-20, there is a button called `<Back to Products>`. Once you click this button, it will redirect to the main chart.



**Figure 5-20.** Drilldown event detail screen

## LegendItem Click Event

The LegendItem click event fires once you click a legend. This action is passed to the method. You set visibility to true or false in a toggle way. See the Listing 5-15 code.

### *Listing 5-15.* app.component.ts

```
plotOptions: {
  series: {
    events: {
      legendItemClick: function () {
        var visibility = this.visible ? 'visible' : 'hidden';
        if (!confirm('The series is currently ' +
          visibility + '. Want to change it ?'))
          {
            return false;
          }
        }
      }
    }
  },
```

As you can see in the Listing 5-15 code, in the `plotOptions.series` you add events as `legendItemClick()`.

Once the user clicks the legend, it will check whether the legend for this particular series is visible or not. The message will generate based on the visibility of a series.

If the user clicks the OK button, the action will occur. If the user clicks the Cancel button, nothing will happen. Now copy the full code in Listing 5-16 into the `app.component.ts` file.

**Listing 5-16.** app.component.ts

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

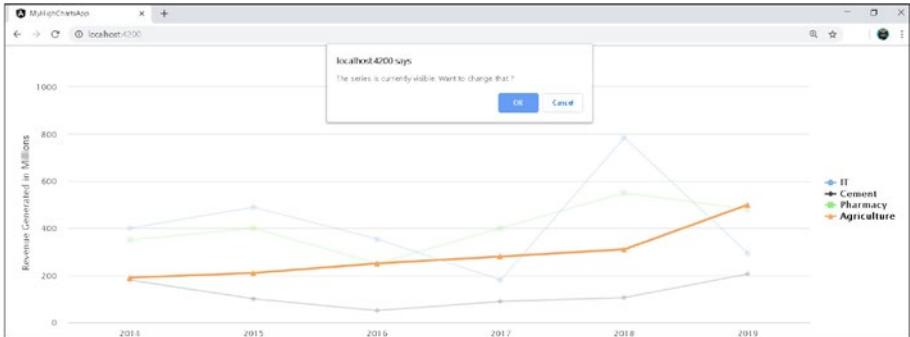
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: 'line'
    },
    title: {
      text: 'Industry Growth by Sector, 2014-2019'
    },
    xAxis: {
      categories: [2014, 2015, 2016, 2017, 2018, 2019],
    },
    yAxis: {
      title: {
        text: 'Revenue Generated in million'
      }
    },
    legend: {
      layout: 'vertical',
      align: 'right',
      verticalAlign: 'middle'
    },
  },
```

```

plotOptions: {
  series: {
    events: {
      legendItemClick: function () {
        var visibility = this.visible ? 'visible' :
          'hidden';
        if (!confirm('The series is currently ' +
          visibility + '. Want to change that ?'))
          {
            return false;
          }
        }
      }
    },
    series: [{
      name: 'IT',
      data: [400, 489, 354, 180, 785, 293]
    }, {
      name: 'Cement',
      data: [180, 100, 50, 89, 105,206]
    }, {
      name: 'Pharmacy',
      data: [350, 400, 250, 400, 550,480]
    }, {
      name: 'Agriculture',
      data: [190, 210, 250, 280, 310,500]
    }
  ]
}

```

Run the Listing 5-16 code with `ng serve` and you will get the output shown in Figure 5-21.



**Figure 5-21.** *LegendItemClick event using HighCharts*

## CheckBoxClick Event

The `CheckBoxClick()` event fires once a user clicks the checkbox visible next to the legend section of a chart. A `CheckBoxclick` event will fire once user Checked or unchecked. You can write your logic based on your requirements. See Listing 5-17.

In this example, once the user clicks the legend checkbox, based on whether it is checked or unchecked, it will display a message in the chart area about the particular series.

**Listing 5-17.** `app.component.ts`

```
plotOptions: {
  series: {
    events: {
      checkBoxClick: function (event) {
        var text;
```



```

if(event.checked==true)
    {
text = 'The checkbox is now checked and Series Label is ' +
this.name;
    }
else
    {
text = 'The checkbox is now unchecked and Series Label is ' +
this.name;
    }
if (!this.chart.lbl) {
this.chart.lbl = this.chart.renderer.label(text, 100, 70)
    .attr({
padding: 10,
r: 5,
fill: Highcharts.getOptions().colors[0],
zIndex: 5
    })
    .css({
color: 'white'
    })
    .add();
    } else {
this.chart.lbl.attr({
text: text
    });
    }
    },
showCheckbox: true
    },

```

In Listing 5-17, there is a property called `showcheckbox: true`. This is the first step to show a checkbox next to a legend.

In the next part, in the event section, you call the `checkboxclick()` method event. In `event.checked`, you get a value of `true` or `false`. If the checkbox is checked, you will get `true`; otherwise `false`. Based on that, I have written the logic.

For the full code, copy Listing 5-18's code into the `app.component.ts` file.

**Listing 5-18.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart:{
      type:'line'
    },
    title: {
      text: 'Industry Growth by Sector, 2014-2019'
    },
    xAxis: {
      categories: [2014, 2015, 2016, 2017, 2018, 2019],
    },
```

```

yAxis: {
  title: {
    text: 'Revenue Generated in million'
  }
},
legend: {
  layout: 'vertical',
  align: 'right',
  verticalAlign: 'middle'
},
plotOptions: {
  series: {
    events: {
      checkboxClick: function (event) {
        var text;
        if(event.checked==true)
          {
            text = 'The checkbox is now checked and Series Label is ' +
            this.name;
          }
        else
          {
            text = 'The checkbox is now unchecked and Series Label is ' +
            this.name;
          }
        if (!this.chart.lbl) {
          this.chart.lbl = this.chart.renderer.label(text, 100, 70)
            .attr({
              padding: 10,
              r: 5,
              fill: Highcharts.getOptions().colors[0],

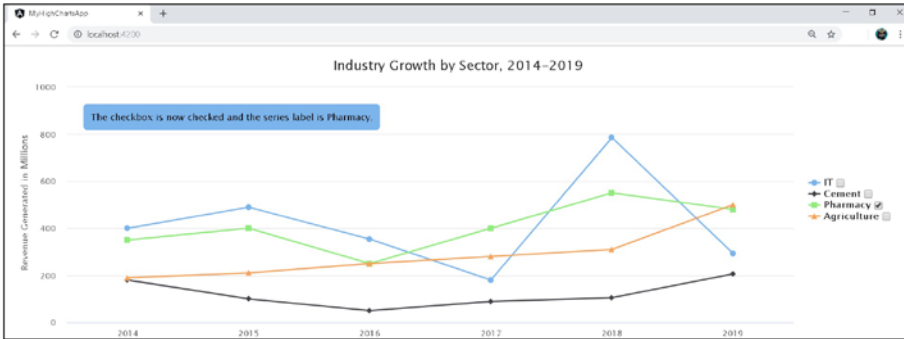
```

```

zIndex: 5
        })
        .css({
color: 'white'
        })
        .add();
    } else {
this.chart.lbl.attr({
text: text
        });
    }
},
showCheckbox: true
},
series: [{
name: 'IT',
data: [400, 489, 354, 180, 785, 293]
}, {
name: 'Cement',
data: [180, 100, 50, 89, 105,206]
}, {
name: 'Pharmacy',
data: [350, 400, 250, 400, 550,480]
}, {
name: 'Agriculture',
data: [190, 210, 250, 280, 310,500]
}],
}
}

```

Once you run the code with `ng serve`, you will get the output seen in Figure 5-22.



**Figure 5-22.** *CheckboxClick event using HighCharts*

## Highcharts Wrapper for .NET

The Highcharts wrapper is an API that is available for almost all significant frameworks like Highcharts IOS, Highcharts Android, Highcharts Angular, Highcharts React, Highcharts React Native, and HighchartsVue (for Vue.js).

In this section, I will talk about the Highcharts wrapper for .NET; this is an API that provides full support for the .NET framework. Now developers can write code without JavaScript. If you are an ASP.NET MVC developer and you want to develop your charting without JavaScript, you can build through the Highcharts API.

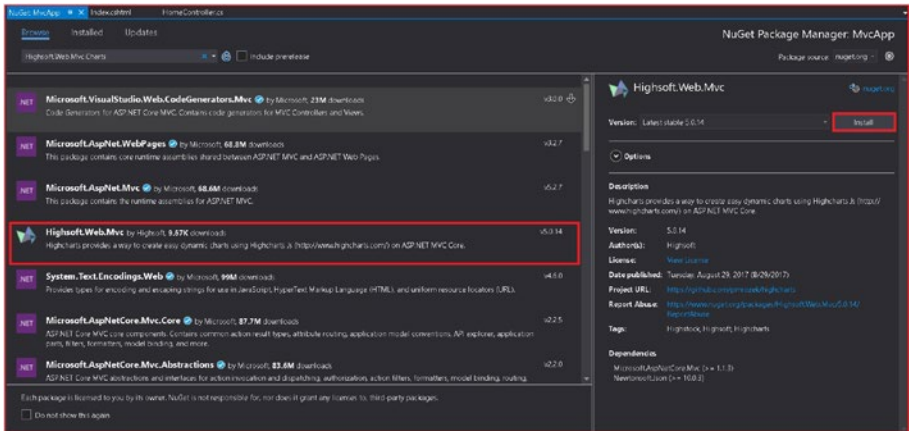
In this section, I will show you two examples: a LineSeries chart and GaugeChart with a Highcharts wrapper.

### LineSeries Chart with a Highcharts Wrapper

Follow these steps:

- Step 1:** Create an ASP.NET MVC application through Visual Studio.

**Step 2:** Go to Solution Explorer. Right-click the MVC project and select Manage NuGet packages from the menu. See Figure 5-23.



**Figure 5-23.** Installing the Highcharts API using NuGet in Visual Studio

**Step 3:** Click Browse, search Highsoft.Web.Mvc. Charts ► Select Highsoft.Web.Mvc, and click the Install button (Figure 5-23).

**Step 4:** Go to Solution Explorer. Open Dependencies ► NuGet section. You can see Highsoft.Web.Mvc.

**Step 5:** Go to the controller, and paste the code in Listing 5-19 into the Index method. For this demo, use `HomeController.cs`.

**Listing 5-19.** HomeController.cs

```

using Microsoft.AspNetCore.Mvc;
using Highsoft.Web.Mvc.Charts;
using System.Collections.Generic;

namespace MvcApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            List<double> marketingDepartmentCollection = new
            List<double>{ 49.9, 51.5, 32.0, 82.0, 75.0, 66.0, 32.0,
            25.0, 35.4, 65.1, 58.6, 34.4 };
            List<double> CsDepartmentCollection = new List<double>{
            40.5, 34.5, 84.4, 39.2, 23.2, 45.0, 55.6, 18.5, 26.4, 14.1,
            23.6, 84.4 };

            List<LineSeriesData> marketingData = new
            List<LineSeriesData>();
            List<LineSeriesData> CsData = new List<LineSeriesData>();

            marketingDepartmentCollection.ForEach(p =>marketingData.
            Add(newLineSeriesData { Y = p }));
            CsDepartmentCollection.ForEach(p =>CsData.Add(newLineSeriesData
            { Y = p }));

            ViewData["marketingData"] = marketingData;
            ViewData["CsData"] = CsData;

            return View();
        }
    }
}

```

In this code, you see lists for the Marketing and Computer Science departments, and you develop a line chart.

So for this demo, you use a `LineSeriesData` class. If you want to create an area chart or pie chart, you must call the related classes like `AreaSeriesData` or `PieSeriesData`.

A later section will talk about series data classes for the Highcharts API. For this `LineSeriesData`, it gets the collection in the form of series data; you send this to the ViewBag collection memory. ViewBag is required when you want to transmit your values from the controller to a view. Now it will work on the UI side, so open View folder ► Home folder ► index.cshtml and paste in the code seen in Listing 5-20.

**Listing 5-20.** index.cshtml

```
<script src="https://code.highcharts.com/highcharts.js">
</script>

@using Highsoft.Web.Mvc.Charts
@using Highsoft.Web.Mvc.Charts.Rendering;

@{var chartOptions = new Highcharts
    {
        Title = new Title
        {
            Text = "Monthly Sales Chart Department Wise",
        },
        Legend = new Legend
        {
            Layout = LegendLayout.Vertical,
            Align = LegendAlign.Right,
            VerticalAlign = LegendVerticalAlign.Middle,
            BorderWidth = 0
        },
```



```

        Subtitle = new Subtitle
        {
            Text = "Year 2018",

        },
XAxis = new List<XAxis>
{
    new XAxis
    {
        Categories = new List<string> { "Jan", "Feb", "Mar", "Apr",
            "May", "Jun","Jul", "Aug", "Sep", "Oct", "Nov", "Dec" },

    }
},
YAxis = new List<YAxis>
{
    new YAxis
        {
            Title = newYAxisTitle
            {
                Text = "Sales in Million $"
            },

PlotLines = new List<YAxisPlotLines>
{
    new YAxisPlotLines
        {
            Value = 0,
            Width = 1,

Color = "red"
        }
    }
},

```

```

        Series = new List<Series>
    {
    new LineSeries
        {
            Name = "Marketing Department",
            Data = @ViewData["marketingData"] as
List<LineSeriesData>
        },
    new LineSeries
        {
            Name = "Computer Science Department",
            Data = @ViewData["CsData"] as
List<LineSeriesData>
        },
    };

    chartOptions.ID = "chart";
var renderer = new HighchartsRenderer(chartOptions);
}

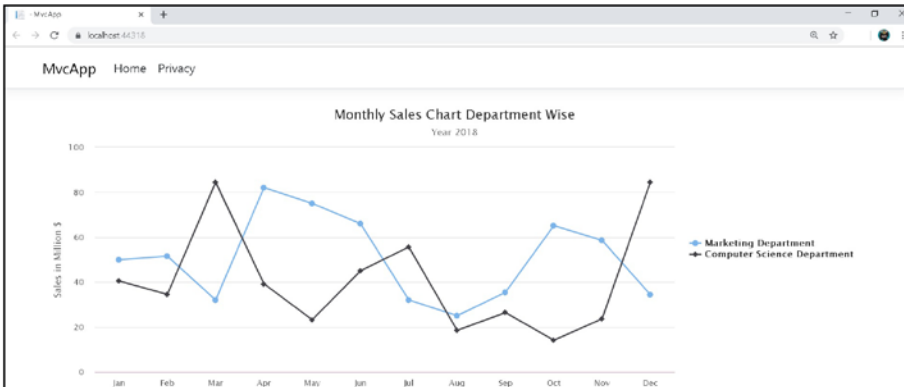
@Html.Raw(renderer.RenderHtml())

```

As you can see in the above code, no JavaScript code is required. You work only with Razor and C# based classes code. Here you have one variable called `chartOptions`, and this variable is called to render Highcharts:

```
var renderer = new HighchartsRenderer(chartOptions);
```

Once you run the above code using Visual Studio, you will get the output seen in [Figure 5-24](#).



**Figure 5-24.** Highcharts API demo with the .NET Framework

## Gauge Series Chart with a Highcharts Wrapper

For developing a gauge, the GaugeSeriesData class is required. For more understanding, copy Listing 5-21 code into the HomeController.cs file.

**Listing 5-21.** HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using Highsoft.Web.Mvc.Charts;
using System.Collections.Generic;

namespace MvcApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            List<GaugeSeriesData> gaugeData = new List<GaugeSeriesData>();
            gaugeData.Add(new GaugeSeriesData { Y = 60 });

            ViewData["gaugeData"] = gaugeData;
        }
    }
}
```

```

return View();
    }
}
}

```

In Listing 5-21, you use the `GaugeSeriesData` class. As you know, in a gauge chart you must set a starting value, so here the value is 60. So whenever you run this code, the starting value will be 60. The next part is the UI. Copy the code in Listing 5-22 into `index.cshtml`.

**Listing 5-22.** `index.cshtml`

```

<script src="https://code.highcharts.com/highcharts.js">
</script>
<script src="https://code.highcharts.com/highcharts-more.js">
</script>

@using Highsoft.Web.Mvc.Charts
@using Highsoft.Web.Mvc.Charts.Rendering

@{var chartOptions = new Highcharts
    {
        Chart = new Highsoft.Web.Mvc.Charts.Chart
        {
            PlotBorderColor = null,
            PlotBackgroundImage = null,
            PlotBorderWidth = 0,
            PlotShadow = new Shadow
                {
                    Enabled = false
                }
        },

```

```

        Title = new Title
        {
            Text = "Speedometer"
        },
        Pane = new Pane
        {
StartAngle = -150,
EndAngle = 150
        },
YAxis = new List<YAxis>
        {
newYAxis
            {
                Min = 0,
                Max = 200,
MinorTickWidth = 1,
MinorTickLength = 10,
MinorTickPosition = YAxisMinorTickPosition.Inside,
MinorTickColor = "#666",

TickPixelInterval = 30,
TickWidth = 2,
TickPosition = YAxisTickPosition.Inside,
TickLength = 10,
TickColor = "#666",

                Labels = new YAxisLabels
                {
                    Step = 2
                },
                Title = new YAxisTitle
                {
                    Text = "km/h"
                },
            }
        }

```

```

PlotBands = new List<YAxisPlotBands>
    {
newYAxisPlotBands
        {
            From = 0,
            To = 120,
Color = "#55BF3B"
        },
new YAxisPlotBands
        {
            From = 120,
            To = 150,
        },
new YAxisPlotBands
        {
            From = 150,
            To = 200,
Color = "#DF5353"
        }
    }
    },
    Series = new List<Series>
{
newGaugeSeries
    {
        Name = "Speed",
        Data = @ViewData["gaugeData"] as
List<GaugeSeriesData>,

```

```

        Tooltip = newGaugeSeriesTooltip
        {
ValueSuffix = " km/h"
        }
    }
}
};
chartOptions.ID = "chart";
var renderer = new HighchartsRenderer(chartOptions);
}

@Html.Raw(renderer.RenderHtml())
<script type="text/javascript">
window.setTimeout(function () {
var chart = Highcharts.charts[0];
if (!chart.renderer.forExport) {
setInterval(function () {
var point = chart.series[0].points[0],
newVal,
inc = Math.round((Math.random() - 0.7) * 30);

newVal = point.y + inc;
if (newVal < 0 || newVal > 200) {
newVal = point.y - inc;
        }

point.update(newVal);

        }, 1000);
    }
}, 1000);
</script>

```

In Listing 5-22, you create a gauge chart using the Highcharts class. After that, you use JavaScript code to generate a gauge event using the `window.setTimeout`, so the value of the gauge chart is changed every second.

```
window.setTimeout(function () {
var chart = Highcharts.charts[0];
if (!chart.renderer.forExport) {
setInterval(function () {
var point = chart.series[0].points[0],
newVal,
inc = Math.round((Math.random() - 0.7) * 30);

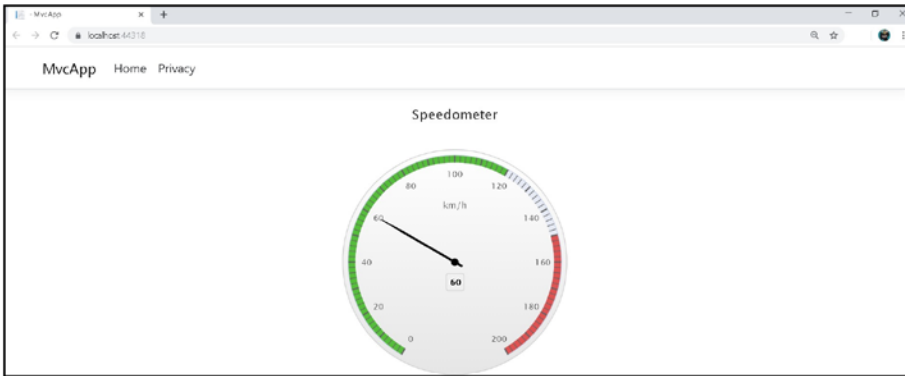
newVal = point.y + inc;
if (newVal < 0 || newVal > 200) {
newVal = point.y - inc;
}

point.update(newVal);

}, 1000);
}, 1000);
```

Run the above code and you will get the output seen in Figure 5-25. The meter changes every second.





*Figure 5-25. Gauge chart with a Highcharts wrapper API for .NET*

## SeriesData Classes

The Highcharts API wrapper is available for different frameworks, and you have to install those dependencies based on your requirements. Refer to [Table 5-1](#).

**Table 5-1.** *Chart Type SeriesData Classes for Working with a Highcharts API in the .NET Framework*

<b>Class Name</b>	<b>Use</b>
LineSeriesData	For drawing line-type charts and generating a series for line components. You already saw an example of this.
AreaSeriesData	For generating an area-type chart with series data for an area chart
SplineSeriesData	For generating a spline-type chart with series data for a spline chart
BarSeriesData	For generating a bar-type chart with series data for a bar chart
ColumnSeriesData	For generating a column-type chart with series data for a column chart
PieSeriesData	For generating a pie-type chart with series data for the pie chart
ScatterSeriesData	For generating a scatter-type chart with series data for a scatter chart

## Summary

In this chapter, you saw how to develop charting with the use of a web API and the Angular service with the REST pattern. You can send the request to any back-end service with an HTTP client; you just need a URL and its return type. You can add events on your charting script very easily; events are executed in the browser once any related action happens, such as a click event, mouse over, mouse up, checked legend click, etc.

The next chapter will talk about themes in Highcharts and additional features provide by Highcharts to make your dashboard more interactive.

## CHAPTER 6

# Themes and Additional Features of Highcharts

In this chapter, you are going to learn about themes and styles and additional features of Highcharts. In this chapter, you will also see how to export your charts in different formats. You'll also learn about 3D charting and Highcharts Gantt. Highcharts Gantt is newly introduced by the Highcharts team. These features will make your dashboard rich and more interactive, so let's begin this chapter.

## Themes in Highcharts

In Computer Science, themes are a set of packages of colors and graphical representations. In Highcharts, you can define your theme. Here's how to set styles and themes for Highcharts:

- **Axis:** You can add styles to the x-axis and the y-axis.
- **alternateGridColor:** You can set this property for the x-axis and the y-axis. This will add a color band alternatively across the chart plot base on the axis. See Listing 6-1.

**Listing 6-1.** app.component.ts

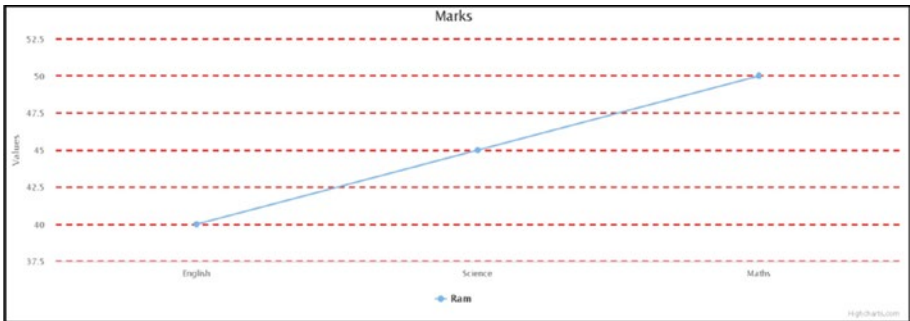
```
yAxis: {  
  alternateGridColor: 'red'  
},  
xAxis: {  
  alternateGridColor: 'green'  
},
```

- `gridLineColor`: This sets a primary grid `lineColor` for an axis.
- `gridLineDashStyle`: With this property, you can establish a line style as a dash into significant network lines.
- `gridLineWidth`: This property is for increasing and decreasing the width for the primary grid line for a chart. See Listing 6-2.

**Listing 6-2.** app.component.ts

```
yAxis: {  
  gridLineDashStyle: 'dash',  
  gridLineWidth: 2,  
  gridLineColor: 'green'  
},
```

If you add the code in Listing 6-2 to the y-axis in a line chart and run it, you will get Figure 6-1.



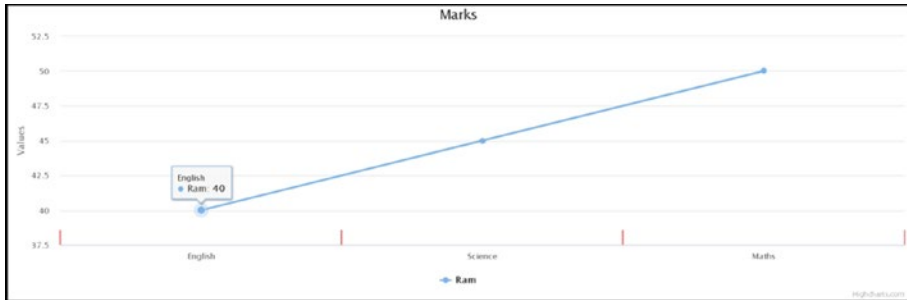
**Figure 6-1.** Demo of `gridLineDashStyle`, `gridLineWidth`, and `gridLineColor`

- `tickWidth`: This property is used to set the width for major ticks. You can set this for both the x-axis and the y-axis.
- `tickPosition`: By default, this property is outside. You can set it as inside, too; also, this will reflect in major ticks.
- `tickColor`: With this property, you can set a major tick color for the axis.
- `tickLength`: This will increase/decrease the length of main ticks in the form of the pixel. See Listing 6-3.

**Listing 6-3.** `app.component.ts`

```
xAxis: {
  categories: ['English', 'Science', 'Maths'],
  tickWidth: 1,
  tickLength: 20,
  tickPosition: 'inside',
  tickColor: 'red',
},
```

If you add the above code to the x-axis, you will get the output shown in Figure 6-2.



**Figure 6-2.** Major tick-related properties for Highcharts

- `lineColor`: This sets the axis line color for the chart.
- `lineWidth`: With this property, you can set the width of the axis line.

## Applying a Dash Style Series to a Line Chart

Highcharts provides a dash style series for line charts. For this, you have to set `series.dashStyle`. You can set the value as `LongDash`, `ShortDot`, `Dot`, `ShortDashDot`, `Dash`, and `DotDash`. Copy the Listing 6-4 code into the `app.component.ts` file.

### **Listing 6-4.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from "highcharts";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

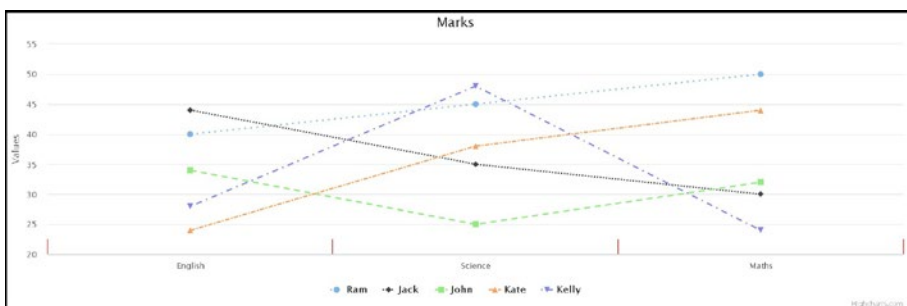
```
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: 'line',
    },
    title: {
      text: 'Marks',
      backgroundColor: '#FCFFC5',
      borderColor: 'black',
      borderRadius: 10,
      borderWidth: 3
    },
    xAxis: {
      categories: ['English', 'Science', 'Maths'],
      tickWidth: 1,
      tickLength: 20,
      tickPosition: 'inside',
      tickColor: 'red',
    },
    series: [{
      name: 'Ram',
      data: [40, 45, 50],
      dashStyle: 'Dot'
    },
    {
      name: 'Jack',
      data: [44, 35, 30],
      dashStyle: 'ShortDot'
    }
  ]
}
```

```

    {
name: 'John',
data: [34, 25, 32],
dashStyle: 'Dash'
    },
    {
name: 'kate',
data: [24, 38, 44],
dashStyle: 'ShortDashDot'
    },
    {
name: 'Kelly',
data: [28, 48, 24],
dashStyle: 'DotDash'
    }
]
};
}

```

Run the Listing 6-4 code and you will get the output seen in Figure 6-3.



**Figure 6-3.** *dashStyle series in a line chart*



## Combinations in Highcharts

Highcharts supports multiple charts in one place. Suppose you want to define a bar and spline in one chart to display a city's temperatures based on rainfall. You can do so very easily. Let's see one example of how you can set a combination using Highcharts and Angular.

In the upcoming example, you will plot a mutual fund's performance for the year against its benchmark index. If you want to perform this kind of task, you have to use a combination of charts. In this example, you'll use a column chart for the mutual fund scheme performance and a spline chart for the benchmark index. Look at Listing 6-5.

### *Listing 6-5.* app.component.ts

```
series: [{
  name: 'Scheme',
  type: 'column',
  yAxis: 1,
  data: [7.43, 8.5, 5.4, 8.2, 8.97, 6.9, 7.6, 8.5, 8.4, 8.9, 9.6,
  10.4],
  tooltip: {
    valueSuffix: ' %'
  }
}, {
  name: 'Benchmark',
  type: 'spline',
  data: [4.12, 3.9, 2.68, 3.5, 4.2, 2.5, 3.2, 6.5, 6.3, 7.3, 7.9,
  7.6],
  tooltip: {
    valueSuffix: '%'
  }
}],
```

In Listing 6-5, in one series array, you use the type property, so here you have two series, one for Benchmark and one for Scheme performance. Remember that you should have a good idea what type of series generation is required for that particular chart. Refer to Chapter 4 where I describe different charting options in Highcharts. Listing 6-6 contains the full code. You can copy this code into the `app.component.ts` file.

**Listing 6-6.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      zoomType: 'xy'
    },
    title: {
      text: 'IECE Digital Bluechip Fund (LargeCap Category)'
    },
    xAxis: [{
      categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
        'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
    }],
```

```

yAxis: [{
  labels: {
    format: '{value}%',
    style: {
      color: Highcharts.getOptions().colors[1]
    }
  },
  title: {
    text: 'Benchmark',
    style: {
      color: Highcharts.getOptions().colors[1]
    }
  }
}, { // Secondary yAxis
  title: {
    text: 'Scheme',
    style: {
      color: Highcharts.getOptions().colors[0]
    }
  },
  labels: {
    format: '{value} %',
    style: {
      color: Highcharts.getOptions().colors[0]
    }
  },
  opposite: true
}],
tooltip: {
  shared: true
},

```

```

legend: {
  layout: 'vertical',
  align: 'left',
  x: 170,
  verticalAlign: 'top',
  y: 70,
  floating: true,
  backgroundColor:
    Highcharts.defaultOptions.legend.backgroundColor
  },
series: [{
  name: 'Scheme',
  type: 'column',
  yAxis: 1,
  data: [7.43, 8.5, 5.4, 8.2, 8.97, 6.9, 7.6, 8.5, 8.4, 8.9, 9.6,
    10.4],
  tooltip: {
    valueSuffix: ' %'
  }
  }, {
  name: 'Benchmark',
  type: 'spline',
  data: [4.12, 3.9, 2.68, 3.5, 4.2, 2.5, 3.2, 6.5, 6.3, 7.3, 7.9,
    7.6],
  tooltip: {
    valueSuffix: '%'
  }
  }],
}
}

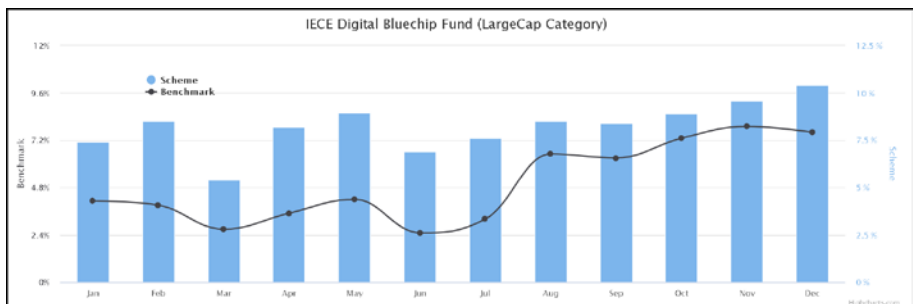
```

Now copy Listing 6-7's code into `app.component.html`.

**Listing 6-7.** `app.component.html`

```
<div class="content" role="main">
<highcharts-chart [Highcharts]="highcharts"
[options]="chartOptions"
style="width: 100%; height: 400px; display: block;">
</highcharts-chart>
</div>
<router-outlet></router-outlet>
```

Now, type `ng serve` into the terminal window of Visual Studio and type your URL in the browser. You will get the output seen in Figure 6-4.



**Figure 6-4.** *Combination column/spline chart*

As you can see, there are two y-axes, one for Scheme and one for Benchmark.

Now let's see another example of a combination chart. This example is a combination of pie, spline, and column charts. In this chart, you will construct a graph for demonstrating the mobile operating systems used by different countries (Listing 6-8).

**Listing 6-8.** app.component.ts

```

series: [{
  type: 'column',
  name: 'India',
  data: [25, 55, 10, 5, 5]
}, {
  type: 'column',
  name: 'UK',
  data: [57, 30, 7, 3, 3]
}, {
  type: 'column',
  name: 'US',
  data: [50, 30, 15, 3, 2]
}, {
  type: 'spline',
  name: 'Average',
  data: [44, 38.3, 10.67, 3.67, 3.34],
  marker: {
    lineWidth: 2,
    lineColor: Highcharts.getOptions().colors[4],
    fillColor: 'white'
  }
}, {
  type: 'pie', //total
  name: 'Total consumption',
  data: [{
    name: 'India',
    y: 100,
    color: Highcharts.getOptions().colors[0]
  }, {
    name: 'UK',

```

```

y: 100,
color: Highcharts.getOptions().colors[1]
    }, {
name: 'US',
y: 100,
color: Highcharts.getOptions().colors[2]
    }],
center: [590, 80],
size: 120,
showInLegend: false,
dataLabels: {
enabled: false
    }
}],

```

In this code, you have a series based on the chart type, so you have to generate the series. This will construct a chart with column, spline, and pie. Listing 6-9 has the complete code, so copy it into the `app.component.ts` file.

**Listing 6-9.** `app.component.ts`

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

```

```

chartOptions = {
  chart: {
    zoomType: 'xy'
  },
  title: {
    text: 'Mobile Operating System used by different Countries in
    Percentage'
  },
  labels: {
    items: [{
      html: 'Total product used',
      style: {
        left: '550px',
        top: '18px',
        color: (
          Highcharts.defaultOptions.title.style&&
          Highcharts.defaultOptions.title.style.color
        ) || 'black'
      }
    }]
  },
  xAxis: {
    categories: ['IOS', 'Android', 'Windows', 'Black Berry', 'Symbian']
  },

  series: [{
    type: 'column',
    name: 'India',
    data: [25, 55, 10, 5, 5]
  }, {
    type: 'column',
    name: 'UK',

```



```

data: [57, 30, 7, 3, 3]
  }, {
type: 'column',
name: 'US',
data: [50, 30, 15, 3, 2]
  }, {
type: 'spline',
name: 'Average',
data: [44, 38.3, 10.67, 3.67, 3.34],
marker: {
lineWidth: 2,
lineColor: Highcharts.getOptions().colors[4],
fillColor: 'white'
  }}, {
type: 'pie', //total
name: 'Total consumption',
data: [{
name: 'India',
y: 100,
color: Highcharts.getOptions().colors[0]
  }, {
name: 'UK',
y: 100,
color: Highcharts.getOptions().colors[1]
  }, {
name: 'US',
y: 100,
color: Highcharts.getOptions().colors[2]
  }],
center: [590, 80],
size: 120,

```

```

showInLegend: false,
dataLabels: {
  enabled: false
}
}],
}
}
}

```

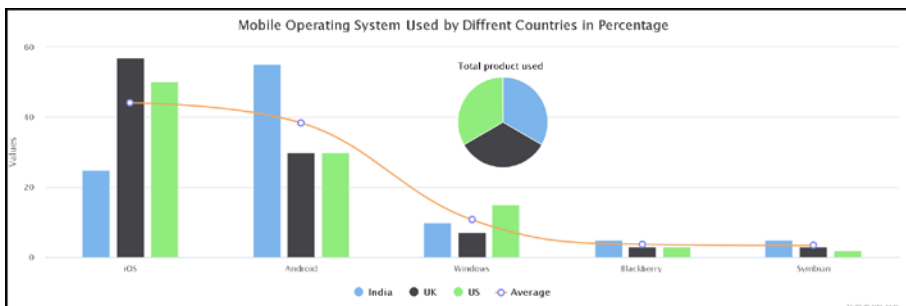
## Zoom Option in Highcharts

Note in Listing 6-9 the appearance of `zoomType: 'xy'`. You can set a `chart.zoomType` property to the x-axis or y-axis, or if necessary you can arrange for the “xy” axis. You can zoom by dragging your mouse pointer based on your setting in the form of x, y, or xy.

Once your zoom is done, on the top right-hand side of your chart area you will automatically get one button as a Zoom Out option.

In `label.item.style`, you set `style` with the `left` and `top` property; this is for changing the position of the “Total product used” label in the chart area. It’s the same as for a pie chart; here the `center` and `size` properties set the position of the pie chart and size of a pie.

Type `ng serve` into the terminal window of Visual Studio and you will get output seen in Figure 6-5.



**Figure 6-5.** Combination column/spline/pie chart

## Setting an Image in a Chart Area

You can set images to make your chart more interactive. In Highcharts, you have an option to arrange an image in the chart area.

For this, you must use the `render.events` method; with the use of the `event.renderer` method, you can set an image. See Listing 6-10. This example is only a simple line chart with one image; here you are showing the yearly temperature of a city in the summer season. On the left-hand side is an image of a sun.

### **Listing 6-10.** `app.component.ts`

```
chart: {
  events: {
    render: function () {
      var chart = this,
          renderer = chart.renderer,
          bg = chart.plotBackground;

      renderer.image('https://www.highcharts.com/samples/graphics/
sun.png', 100, 100, 30, 30).add();
    }
  },
},
```

Here you render the event method. In this method, you set a variable as `renderer = chart.renderer` and then the `renderer.image` method is used to set the path of the image, the image position, and the image size for the chart. So this is how to place an image in a chart. Listing 6-11 is the full code, so copy it into the `app.component.ts` file.

**Listing 6-11.** app.component.ts

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      events: {
        render: function () {
          var chart = this,
              renderer = chart.renderer,
              bg = chart.plotBackground;

          renderer.image('https://www.highcharts.com/samples/graphics/
sun.png', 100, 100, 30, 30).add();
        }
      },
    },
  },

  title: {
    text: 'Yearly Temperature in Summar Season'
  },

  xAxis: {
    name: 'Year',

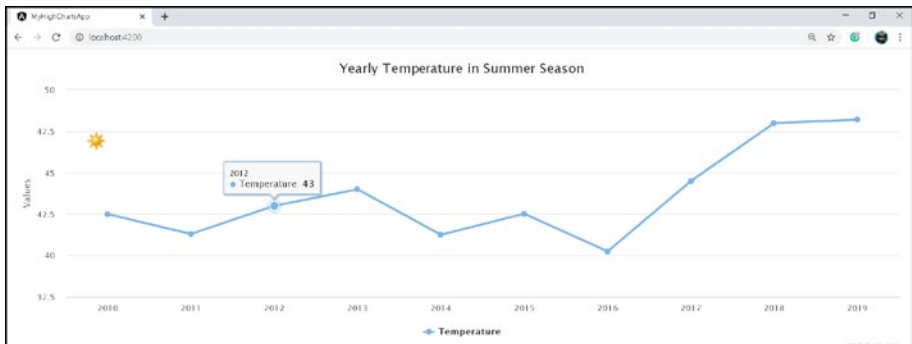
```

```

categories: [2010, 2011, 2012, 2013, 2014, 2015, 2016,
2017, 2018, 2019],
  },
series: [{
name: 'Temprature',
data: [42.5, 41.3, 43.0, 44.0, 41.25, 42.52, 40.25, 44.50,
48.0, 48.2]
}]
}
}

```

Type `ng serve` into the terminal window of Visual Studio. Once you run this code, you will get the output shown in Figure 6-6.



**Figure 6-6.** Adding an image in a Highchart example

## 3D Charts

Highcharts 3D provides 3D support to Highcharts. With this feature, you can develop an interactive chart. To work with a 3D chart, here are the required dependencies:

**jQuery:**

```
<script src="https://code.highcharts.com/highcharts-3d.js">
</script>
```

**Angular:**

```
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
```

To implement Highcharts 3D in code, see Listing [6-12](#).

**Listing 6-12.** app.component.ts

```
options3d:
  {
    enabled: boolean,
    alpha: number,
    beta: number,
    depth: number,
    viewDistance: number,
    axisLabelPosition: "auto",
    fitToPlot: boolean,
    frame: {
      bottom: {
        size: number,
        color: 'color'
      },
      back: {
        size: number,
        color: 'color'
      },
```

```

side: {
size: number,
color: 'color'
    },
}
}

```

You can use these properties based on your requirements:

- `Options3d`: Required to develop a 3D chart; this is the leading property.
- `enabled`: This is a Boolean property. If `true`, you will see a 3D chart. If `false`, nothing will reflect.
- `alpha`: Number type of property; it will rotate in the bottom and top view level.
- `beta`: Number type of property; it will turn right and left.
- `depth`: This is for a total depth of chart.
- `viewDistance`: This defines how far viewers are from the chart.
- `frame`: This is for setting a chart from the bottom, back, and side.

Listing 6-13 is the full code, so copy it into `app.component.ts`.

**Listing 6-13.** `app.component.ts`

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);

```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

  chartOptions = {
    chart: {
      renderTo: 'container',
      type: 'column',

      options3d:
        {
          enabled: true,
          alpha: 10,
          beta: 45,
          depth: 150,
          viewDistance: 50,
          axisLabelPosition: "auto",
          fitToPlot: true,

          frame: {
            bottom: {
              size: 10,
              color: 'orange'
            },
            back: {
              size: 10,
              color: 'orange'
            },
```



```
side: {
  size: 10,
  color: 'orange'
    },
  }
},

title: {
  text: 'Real Time Data Example'
    },

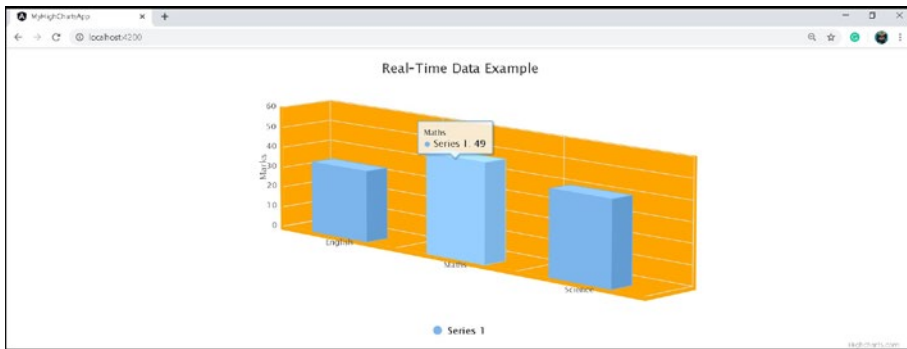
xAxis: {
  categories: ['English', 'Maths', 'Science']
    },

yAxis: {
  title: {
    text: 'Marks'
      },
  },

plotOptions: {
  column: {
    depth: 65
      }
  },

series: [{
  data: [35, 49, 42]
    }]
  }
}
```

Type `ng serve` into a terminal window of Visual Studio and you will get the output shown in Figure 6-7.



*Figure 6-7. Highcharts 3D example for a column type chart*

## Cylinder Chart

A cylinder chart is another type of 3D chart. If you want to implement a cylinder, the following dependencies are required:

### jQuery:

```
<script src="https://code.highcharts.com/highcharts-3d.js">
</script>
<script src="https://code.highcharts.com/modules/cylinder.js">
</script>
```

### Angular:

```
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
import cylinder from 'highcharts/modules/cylinder.src'
cylinder(Highcharts);
```

For implementing cylinder chart, the type should be `cylinder`. You must add the dependency as per your JavaScript framework.

Copy Listing 6-14's code into `app.component.ts` to generate a cylinder chart.

**Listing 6-14.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
import cylinder from 'highcharts/modules/cylinder.src'
cylinder(Highcharts);

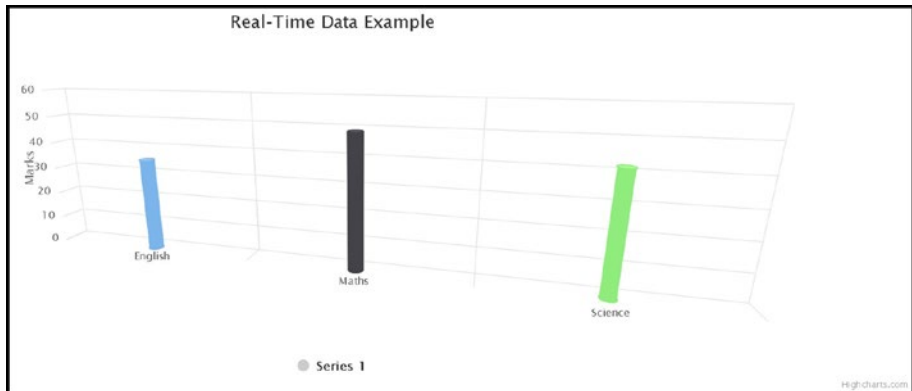
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

  chartOptions = {
    chart: {
      renderTo: 'container',
      type: 'cylinder',
      options3d:
        {
          enabled: true,
          beta: 15,
          alpha: 15,
```

```
viewDistance: 15,  
depth: 50,  
  }  
},  
  
title: {  
text: 'Real Time Data Example'  
  },  
  
xAxis: {  
categories: ['English', 'Maths', 'Science']  
  },  
  
yAxis: {  
title: {  
text: 'Marks'  
  },  
  },  
  
plotOptions: {  
series: {  
depth: 25,  
colorByPoint: true  
  }  
  },  
  
series: [{  
data: [35, 49, 42]  
  }],  
  }  
}
```

Now type `ng serve` and you will get the output shown in Figure 6-8.



**Figure 6-8.** *Cylinder chart example*

## Funnel 3D

The funnel 3D chart type is another type used for 3D charting in Highcharts. A funnel chart is used to display different stages in a business process; the completed process is on the top and the pending process is on the bottom.

To implement the funnel 3D chart, the following dependencies are required:

### jQuery:

```
<script src="https://code.highcharts.com/highcharts-3d.js">
</script>
<script src="https://code.highcharts.com/modules/cylinder.js">
</script>
<script src="https://code.highcharts.com/modules/funnel3d.js">
</script>
```

### Angular:

```
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
```

```
import cylinder from 'highcharts/modules/cylinder.src';
cylinder(Highcharts);
import funnel3d from 'highcharts/modules/funnel3d.src';
funnel3d(Highcharts);
```

To create a funnel 3D chart, the `chart.type` should be `funnel3d`, and the next step should be to define the series. See Listing 6-15.

**Listing 6-15.** `app.component.ts`

```
series: [{
  name: 'Customers',
  data: [
    ['Customer visits Website totally', 8000],
    ['App Downloads', 5150],
    ['Requested price list', 2000],
    ['Proposal sent', 1600],
  ]
}]
```

Listing 6-15 shows labels and values for the data series for the funnel. Next, you require `plotOptions`; see Listing 6-16.

**Listing 6-16.** `app.component.ts`

```
plotOptions: {
  series: {
    dataLabels: {
      enabled: true,
      format: '<b>{point.name}</b> ({point.y:,.0f})',
      allowOverlap: true,
    },
  },
  height: '50%',
  width: '40%',
```

```

neckWidth: '15%',
neckHeight: '15%',
  }
},

```

In `plotOptions`, you can set height, width, and format-related settings for `funnel3d`. Listing 6-17 is the full code to generate `funnel3d`, so copy this code into the `app.component.ts` file.

**Listing 6-17.** `app.component.ts`

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
import cylinder from 'highcharts/modules/cylinder.src';
cylinder(Highcharts);
import funnel3d from 'highcharts/modules/funnel3d.src';
funnel3d(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    title: {
      text: 'Highcharts Funnel3D Chart'
    },

```

```

chart: {
  renderTo: 'container',
  type: 'funnel3d',
  options3d: {
    enabled: true,
    alpha: 10,
    depth: 50,
    viewDistance: 50
  }
},
series: [{
  name: 'Customers',
  data: [
    ['Customer visits Website totally', 8000],
    ['App Downloads', 5150],
    ['Requested price list', 2000],
    ['Proposal sent', 1600],
  ]
}],

plotOptions: {
  series: {
    dataLabels: {
      enabled: true,
      format: '<b>{point.name}</b> ({point.y:,.0f})',
      allowOverlap: true,
    },
    height: '50%',
    width: '40%',
    neckWidth: '15%',
    neckHeight: '15%',
  }
}

```

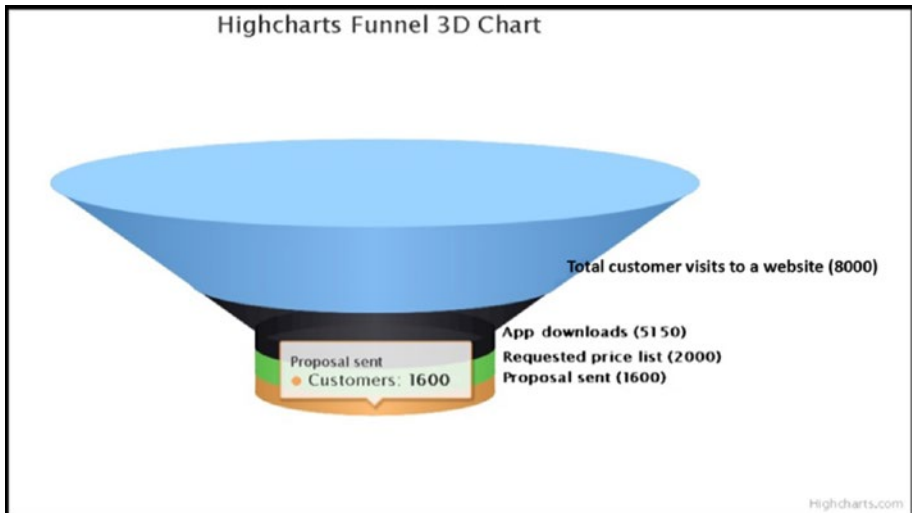


```

    },
  }
}

```

Type `ng serve` and press Enter. You will get the output shown in Figure 6-9.



**Figure 6-9.** Funnel 3D chart

## Pyramid 3D

A pyramid chart is a triangle-based chart. The triangle has sections and these sections work top to bottom. This type of chart is mostly used to show hierarchy, priorities, steps, or processes.

To work on a pyramid 3D chart, the following dependencies are required:

**jQuery:**

```

<script src="https://code.highcharts.com/highcharts.js">
</script>

```

```

<script src="https://code.highcharts.com/highcharts-3d.js">
</script>
<script src="https://code.highcharts.com/modules/cylinder.js">
</script>
<script src="https://code.highcharts.com/modules/funnel3d.js">
</script>
<script src="https://code.highcharts.com/modules/pyramid3d.js">
</script>

```

**Angular:**

```

import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
import cylinder from 'highcharts/modules/cylinder.src';
cylinder(Highcharts);
import funnel3d from 'highcharts/modules/funnel3d.src';
funnel3d(Highcharts);
import pyramid3d from 'highcharts/modules/pyramid3d.src';
pyramid3d(Highcharts);

```

This demo is the same as the one for the funnel 3D. A pyramid 3D works top to bottom. Copy Listing 6-17's code into `app.component.ts`.

**Listing 6-17.** `app.component.ts`

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
import cylinder from 'highcharts/modules/cylinder.src';
cylinder(Highcharts);
import funnel3d from 'highcharts/modules/funnel3d.src';

```

```
funnel3d(Highcharts);
import pyramid3d from 'highcharts/modules/pyramid3d.src';
pyramid3d(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

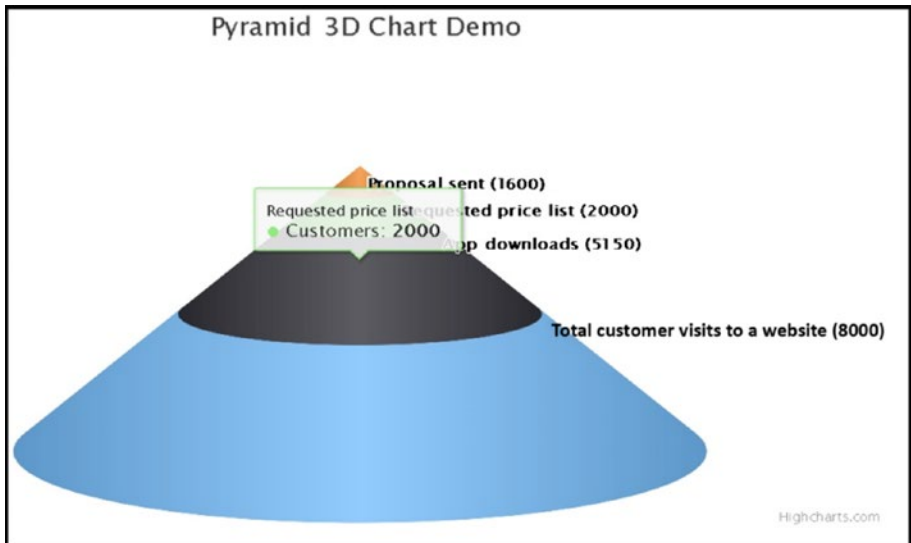
  chartOptions = {
    title: {
      text: 'Pyramid3D Chart Demo'
    },

    chart: {
      renderTo: 'container',
      type: 'pyramid3d',
      options3d: {
        enabled: true,
        alpha: 10,
        depth: 50,
        viewDistance: 50
      }
    },
  },
}
```

```
series: [{
  name: 'Customers',
  data: [
    ['Customer visits Website totally', 8000],
    ['App Downloads', 5150],
    ['Requested price list', 2000],
    ['Proposal sent', 1600],
  ]
}],

plotOptions: {
  series: {
    dataLabels: {
      enabled: true,
      format: '<b>{point.name}</b> ({point.y:,.0f})',
      allowOverlap: true,
    },
    width: '40%',
    height: '60%',
  }
},
}
```

Once you run this code you will get the output shown in [Figure 6-10](#).



**Figure 6-10.** Pyramid 3D chart

## Pie 3D Chart

With the pie 3D chart, you can add 3D features very easily. Here the chart type is pie. And you must add the following dependencies:

### jQuery:

```
<script src="https://code.highcharts.com/highcharts.js">
</script>
<script src="https://code.highcharts.com/highcharts-3d.js">
</script>
```

### Angular:

```
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);
```

Next, you have to set options3d and plotOptions. See Listing 6-18.

**Listing 6-18.** app.component.ts

```

chart: {
  type: 'pie',
    options3d: {
enabled: true,
alpha: 65,
    }
  },

```

Here `options3d.enabled` is `true`. Note the use of the `alpha` property, which is required for rotating angles of a chart. Next, you need `plotOptions` for the depth of the chart for 3D; see Listing 6-19.

**Listing 6-19.** app.component.ts

```

plotOptions: {
  pie: {
allowPointSelect: true,
cursor: 'pointer',
depth: 65,
dataLabels: {
enabled: true,
format: '<b>{point.name}</b>: {point.percentage:.1f} %'
    }
  }
},

```

Paste Listing 6-20's complete code into the `app.component.ts` file.

**Listing 6-20.** app.component.ts

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import highcharts3D from 'highcharts/highcharts-3d.src';
highcharts3D(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: 'pie',
      options3d: {
        enabled: true,
        alpha: 65,
      }
    },
    plotOptions: {
      pie: {
        allowPointSelect: true,
        cursor: 'pointer',
        depth: 65,
        dataLabels: {
          enabled: true,
```

```

format: '<b>{point.name}</b>: {point.percentage:.1f} %'
      }
    }
  },

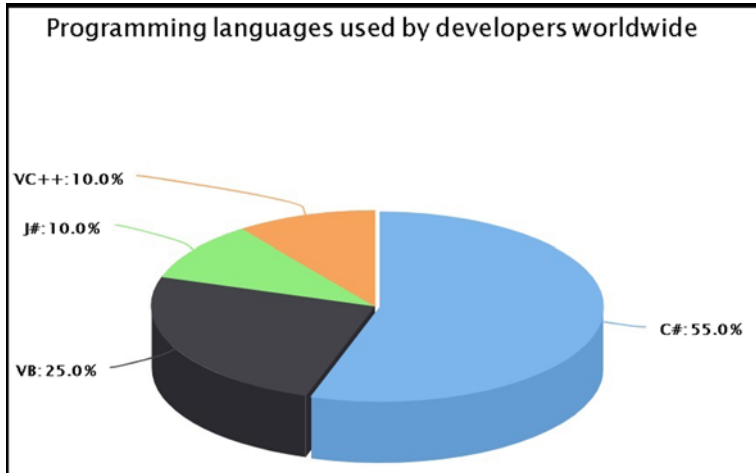
title: {
text: 'Programming Languages used by developers worldwide'
  },
tooltip: {
pointFormat: '{series.name}: <b>{point.percentage:.1f}%</b>'
  },

series: [{
name: 'Uses',
colorByPoint: true,
data: [{
name: 'C#',
y: 55,
sliced: true,
selected: true
  }, {
name: 'VB',
y: 25
  }, {
name: 'J#',
y: 10
  }, {
name: 'VC++',
y: 10
  }
  ]
  }];
}

```



Run the Listing 6-20 code and you will get the output shown in Figure 6-11.



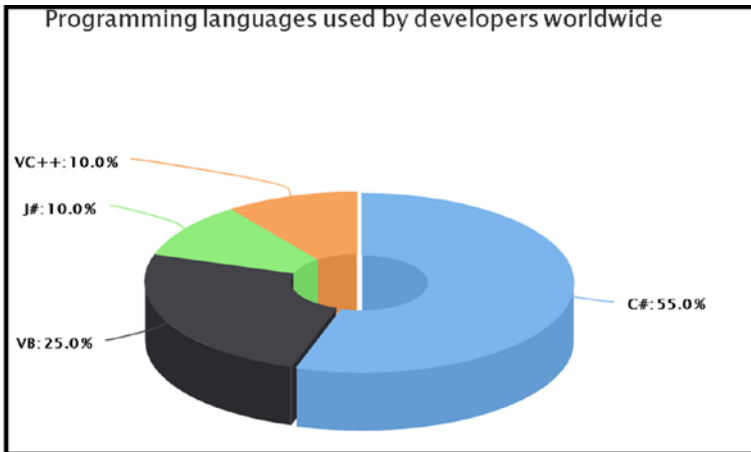
**Figure 6-11.** 3D pie chart

You can convert the same chart into a donut chart with the use of `plotOptions.pie.innersize: number`. See Listing 6-21.

**Listing 6-21.** `app.component.ts`

```
plotOptions: {  
  pie: {  
    innerSize: 100,  
  }  
}
```

If you add the code in Listing 6-21 into `app.component.ts` and run it, you will get the output shown in Figure 6-12.



*Figure 6-12. 3D donut chart demo*

## Exporting and Printing Charts

Highcharts also provide export and print features, so a user can print and download charts easily in the PNG, JPEG, PDF, XLS, CSV, and SVG formats. For exporting and printing, Highcharts provides different dependencies.

- `Exporting.js`: This js dependency provides the following options:
  - View in full screen
  - Print chart
  - Download PNG image
  - Download JPEG
  - Download PDF
  - Download SVG vector

- `Export-data.js`: If you want to add more options related to exporting to CSV and XLS, this dependency is required, but you have to add `exporting.js` with it. `Export-data.js` adds the following options:
  - Download CSV
  - Download XLS
  - View in data table
  - Open in Highcharts cloud

If you want to add export and print functionality, you must add these dependencies based on your JavaScript framework:

**jQuery:**

```
<script src="https://code.highcharts.com/modules/exporting.js">
</script>
<script src="https://code.highcharts.com/modules/export-data.js">
</script>
```

**Angular:**

```
import * as Highcharts from "highcharts";
import HighchartsExporting from "highcharts/modules/exporting";
import HighchartsExportData from "highcharts/modules/export-data";
HighchartsExporting(Highcharts);
HighchartsExportData(Highcharts);
```

Once you add the dependency into your code, you will automatically get the right-hand menu bar. If you click this menu, you will get the export and print features. See Figure 6-13.

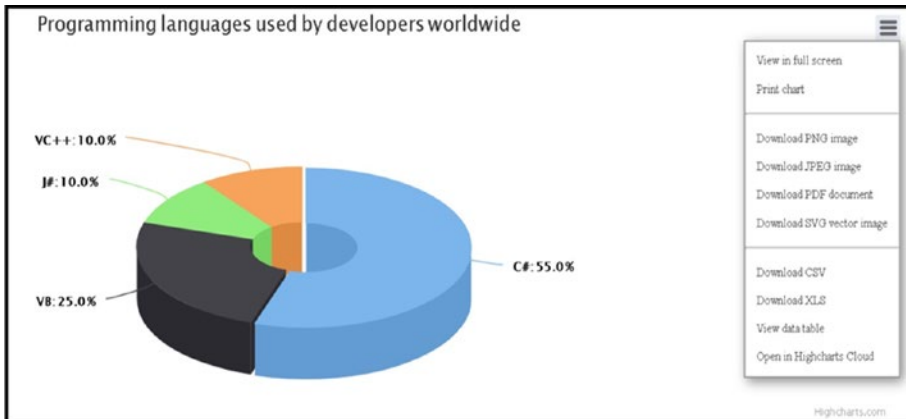


Figure 6-13. Export and print options in Highcharts

## Additional Chart Features

This section covers new charts provided by Highcharts.

### Radar Chart

A radar chart is useful for providing multivariate data, data you get in two dimensional or more quantitative variables. Radar charts are helpful for showing a comparison of data or if you want to show ratings for analyzing a particular product by its features.

In this demo, you will show a comparison of two mobile features so people can analyze which one is best, based on functionality.

If you want to create a radar polar chart using Highcharts, the first step is to set `chart.polar: true`. In the `chart.type` you can set `area`, `line`, `spline`, and `columns`. To develop polar radar charts, the following dependencies are required:

**jQuery:**

```
<script src="http://code.highcharts.com/highcharts.js">
</script>
<script src="http://code.highcharts.com/highcharts-more.js">
</script>
```

**Angular:**

```
import { Component } from '@angular/core';
import * as Highcharts from "highcharts";
import HighChartMore from 'highcharts/highcharts-more.src';
HighChartMore(Highcharts);
```

Now copy Listing 6-22's code into the `app.component.ts` file.

**Listing 6-22.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from "highcharts";
import HighChartMore from 'highcharts/highcharts-more.src';
HighChartMore(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      polar: true,
```

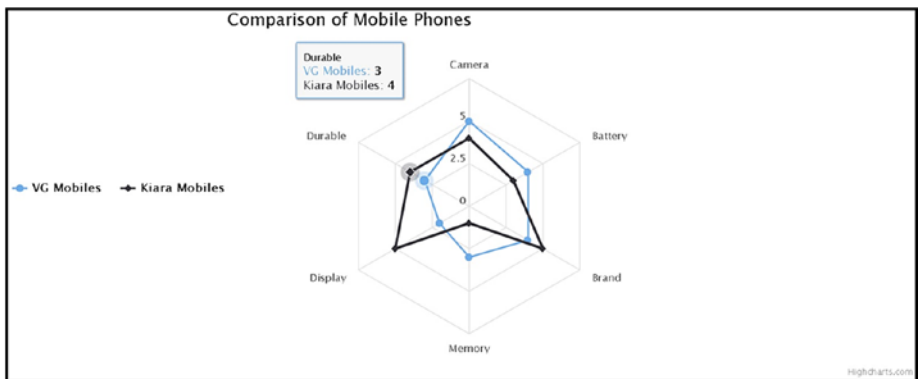
```
type: 'line',
  },
pane: {
  size: '80%'
  },
title: {
  text: 'Comparison of Mobile Phones',
  },
tooltip: {
  shared: true,
  pointFormat: '<span style="color:{series.color}">
{series.name}: <b>{point.y:,.0f}</b><br/>'
  },
xAxis: {
  categories: ['Camera', 'Battery', 'Brand', 'Memory',
    'Display', 'Durable'],
  tickmarkPlacement: 'on',
  lineWidth: 0
  },
yAxis: {
  gridLineInterpolation: 'polygon',
  lineWidth: 0,
  min: 0
  },
legend: {
  align: 'left',
  verticalAlign: 'middle'
  },
```

```

series: [{
  name: 'VG Mobiles',
  data: [5, 4, 4, 3, 2, 3],
  pointPlacement: 'on'
}, {
  name: 'Kiara Mobiles',
  data: [4, 3, 5, 1, 5, 4],
  pointPlacement: 'on'
}],
};
}

```

Run Listing 6-22's code and you will get the output shown in Figure 6-14.



**Figure 6-14.** Radar chart demo

## Pareto Chart

A Pareto chart is a combination of a line and bar graph, where values are defined in the form of descending order through bars, and the cumulative total comes as a line. First of all, you have to understand how Pareto works. Table 6-1 describes hair loss reasons for men.

**Table 6-1.** *Hair Loss Reasons for Men*

Hair Loss Reason	Frequency	Cumulative Frequency	Percentage
Genetics	50	50	57.47%
Cosmetic damage	15	65	74.71%
Stress	11	76	87.35%
Smoking	6	82	94.25%
Vitamin Deficiency	3	85	97.70%
Infections	2	87	100%
Total	87		

Table 6-1 shows hair loss reason and frequency. The cumulative frequency is based on the next value total. For example, genetics + cosmetic damage = cumulative frequency. Based on totality, you create a percentage, and the percentage part comes into the Pareto section.

For this example, you'll convert the information in Table 6-1 into Highcharts with the Pareto chart.

To generate a Pareto chart, the following dependencies are required:

**jQuery:**

```
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/pareto.js">
</script>
```

**Angular:**

```
import { Component } from '@angular/core';
import * as Highcharts from "highcharts";
import Pareto from 'highcharts/modules/pareto.src';
Pareto(Highcharts);
```



Next, you must set the y-axis. See Listing 6-23.

**Listing 6-23.** app.component.ts

```

yAxis: [{
  title: {
    text: ''
  }, {
  title: {
    text: ''
  },

  minPadding: 0,
  maxPadding: 0,
  max: 100,
  min: 0,
  opposite: true,
  labels: {
    format: "{value}%"
  }
}],

```

In Listing 6-23, you have set a `min` and `max` property for Percentage. In the next part, you have a series for both columns and Pareto. See Listing 6-24.

**Listing 6-24.** app.component.ts

```

series: [{
  type: 'pareto',
  name: 'Pareto',
  yAxis: 1, //number of declared yaxis
  baseSeries: 1 //index of column series
},

```

```
{
  name: 'Frequency',
  type: 'column',
  data: [50, 15, 11, 6, 3, 2],
  color: '#FF0000'
}]
```

Listing 6-25 is the complete code for generating the Pareto chart, so copy this code into the `app.component.ts` file.

**Listing 6-25.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from "highcharts";
import Pareto from 'highcharts/modules/pareto.src';
Pareto(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;
  chartOptions = {
    chart: {
      type: 'column'
    },

    title: {
      text: 'Haifall Reasons for Men',
    },
```

```
tooltip: {
  shared: true,
  },

xAxis: {
  categories: [
    'Genetically',
    'Cosmetic damage',
    'Stress',
    'Smoke',
    'Vitamin Defficiency',
    'Infections',
  ],
  },

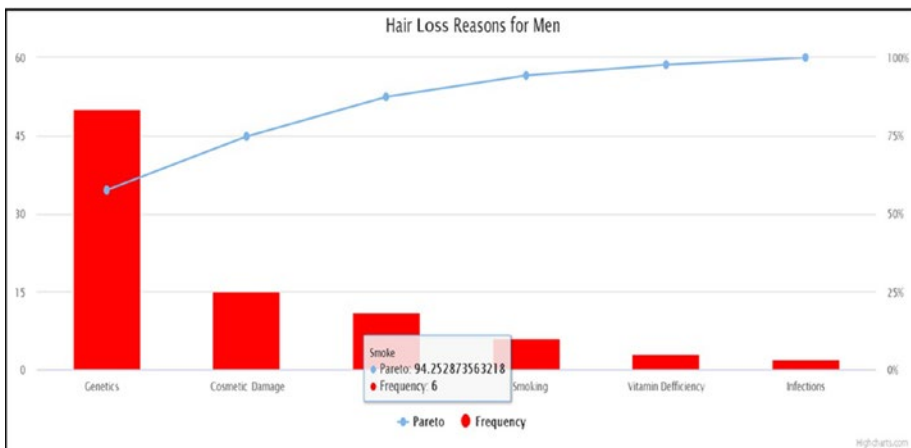
yAxis: [{
  title: {
  text: ''
  }
  }, {
  title: {
  text: ''
  }
  },

  minPadding: 0,
  maxPadding: 0,
  max: 100,
  min: 0,
  opposite: true,
  labels: {
  format: "{value}%"
  }
  }],
```

```

series: [{
  type: 'pareto',
  name: 'Pareto',
  yAxis: 1, //number of declared yAxis
  baseSeries: 1 //index of column series
}, {
  name: 'Frequency',
  type: 'column',
  data: [50, 15, 11, 6, 3, 2],
  color: '#FF0000'
}]
};
}
    
```

Run Listing 6-25's code and you will get the output shown in Figure 6-15.



**Figure 6-15.** Pareto Chart

## Bell Curve Chart

A bell curve chart is used for distribution of variables. Considering a normal distribution with a bell shape line, the highest point in the curve represents the most probable event in the series of data.

To generate a bell curve using Highcharts, the following dependencies are required:

### **jQuery:**

```
<script src="https://code.highcharts.com/highcharts.js">
</script>
<script src="https://code.highcharts.com/modules/histogram-
bellcurve.js"></script>
```

### **Angular:**

```
import * as Highcharts from 'highcharts';
import Bellcurve from 'highcharts/modules/histogram-bellcurve';
Bellcurve(Highcharts);
```

In the next step, the chart type should be `bellcurve`, and the `baseSeries` you can define in the form of `id` or `index`.

Copy Listing 6-26's code into the `app.component.ts` file to generate the bell curve.

### **Listing 6-26.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import Bellcurve from 'highcharts/modules/histogram-bellcurve';
Bellcurve(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
```

```
styleUrls: ['./app.component.css']  
})
```

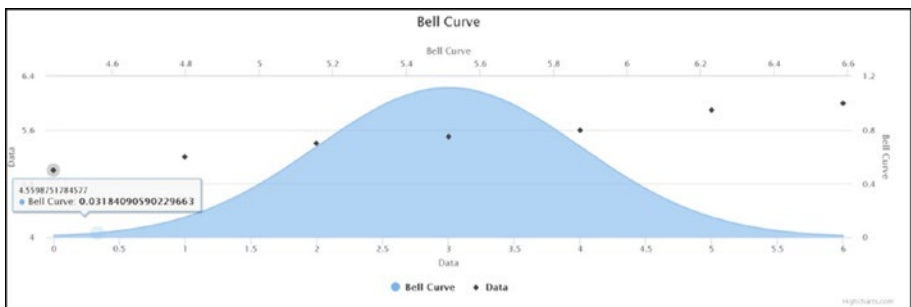
```
export class AppComponent {  
  title = 'myHighChartsApp';  
  highcharts = Highcharts;  
  
  chartOptions = {  
    title: {  
      text: 'Bell Curve'  
    },  
  
    xAxis: [{  
      title: { text: 'Data' },  
    }, {  
      title: { text: 'Bell Curve' },  
      opposite: true  
    }],  
  
    yAxis: [{  
      title: { text: 'Data' }  
    }, {  
      title: { text: 'Bell Curve' },  
      opposite: true  
    }],  
  
    series: [{  
      name: 'Bell Curve',  
      type: 'bellcurve',  
  
      xAxis: 1,  
      yAxis: 1,  
      baseSeries: 'series1',  
      zIndex: -1  
    }]  
  }  
}
```

```

    },
    {
name: 'Data',
type: 'scatter',
data: [5, 5.2, 5.4, 5.5, 5.6, 5.9, 6],
visible: true,
id: 'series1',
    }
];
}

```

If you run Listing 6-26's code, you will get the output shown in Figure 6-16.



**Figure 6-16.** Bell curve demo

## Organization Chart

Organization charts are helpful to show organization structure or hierarchy. You can understand a Reporting To structure very quickly through a chart.

If you want to develop organization charts using Highcharts, the following dependencies are required:

**jQuery:**

```

<script src="https://code.highcharts.com/highcharts.js">
</script>
<script src="https://code.highcharts.com/modules/sankey.js">
</script>
<script src="https://code.highcharts.com/modules/
organization.js"></script>

```

**Angular:**

```

import * as Highcharts from 'highcharts';
import Sankey from 'highcharts/modules/sankey';
import Organisation from 'highcharts/modules/organization';
Sankey(Highcharts);
Organisation(Highcharts);

```

Now copy Listing 6-27's code into the `app.component.ts` file.

**Listing 6-27.** `app.component.ts`

```

import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import Sankey from 'highcharts/modules/sankey';
import Organisation from 'highcharts/modules/organization';
Sankey(Highcharts);
Organisation(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['./app.component.css']
})

```



```
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

  chartOptions = {
    chart: {
      inverted: true,
      height: 400,
    },

    title: {
      text: 'Reporting Structure For IECE Group'
    },

    series: [{
      type: 'organization',
      name: 'IECE',
      keys: ['from', 'to'],

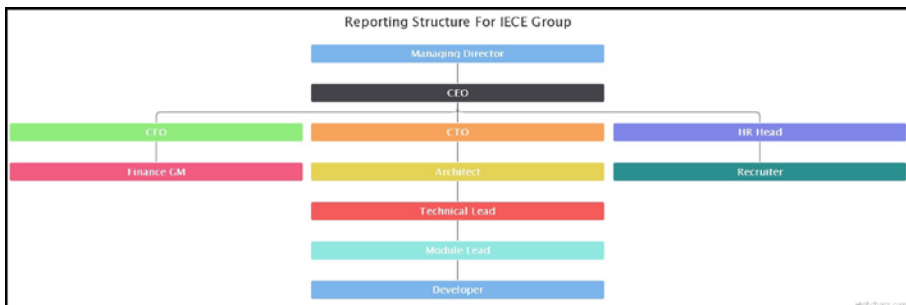
      data: [
        ['Managing Director', 'CEO'],
        ['CEO', 'CFO'],
        ['CEO', 'CTO'],
        ['CEO', 'HR Head'],
        ['CFO', 'Finance GM'],
        ['CTO', 'Architect'],
        ['HR Head', 'Recruiter'],
        ['Architect', 'Technical Lead'],
        ['Technical Lead', 'Module Lead'],
        ['Module Lead', 'Developer']
      ],
    ]
  }
}
```

```

color: 'blue',
dataLabels: {
  color: 'white'
},
borderColor: 'white',
nodeWidth: 25
}],
tooltip: {
  outside: true
},
};
}

```

In the chart `type:organization` and `series` sections, there is new property called `key`. Here you define `[from, to]` so you can define structure. For example, in this demo, you create a structure for the IECE group, where the CEO reports to the Managing Director, and the CFO, HR Head, and the CTO all report to the CEO, and so on. Once you run the above code, you will get the output shown in Figure 6-17.



**Figure 6-17.** Organization chart using Highcharts

## Timeline Chart

The timeline chart is designed to show a journey over time. Here you can define essential events in the form of the vertical or horizontal line. For each event, you can set a description for the event so via a tooltip the user can get details about the event. If you want to perform a timeline chart using Highcharts, the following dependencies are required:

### jQuery:

```
<script src="https://code.highcharts.com/highcharts.js">
</script>
<script src="https://code.highcharts.com/modules/timeline.js">
</script>
```

### Angular:

```
import * as Highcharts from 'highcharts';
import TimeLine from 'highcharts/modules/timeline';
TimeLine(Highcharts);
```

Now copy Listing 6-28's code into `app.component.ts`.

### **Listing 6-28.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import TimeLine from 'highcharts/modules/timeline';
TimeLine(Highcharts);

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['./app.component.css']
})
```

## CHAPTER 6 THEMES AND ADDITIONAL FEATURES OF HIGHCHARTS

```
export class AppComponent {
  title = 'myHighChartsApp';
  highcharts = Highcharts;

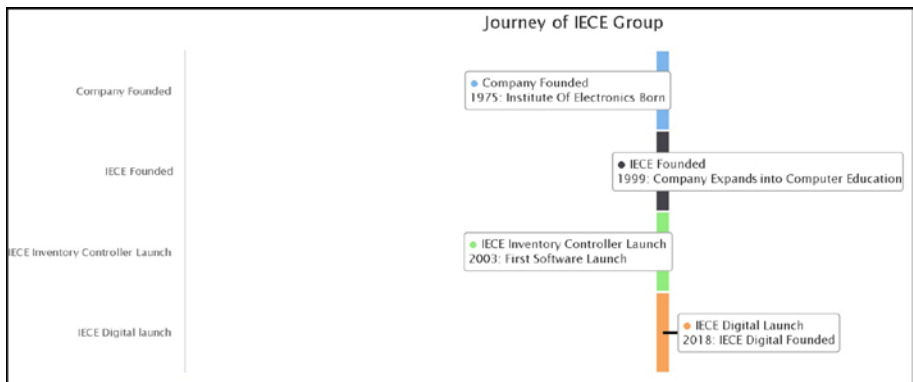
  chartOptions = {
    chart: {
      type: 'timeline',
      inverted: true
    },
    yAxis: {
      visible: false
    },
    title: {
      text: 'Journey of IECE Group'
    },
    series: [{
      dataLabels: {
        connectorColor: 'black',
        connectorWidth: 3
      },
      data: [{
        name: 'Company Founded',
        label: '1975: Institute Of Electronics Born',
        description: 'In the year 1975, Institute of Electronics found
to provide trainings for Electrical/Electronics Engineers'
      }, {
        name: 'IECE founded',
        label: '1999: Company Expend into Computer Education',
        description: 'With new Name IECE, company starts provide
training into Computer Scienece Students as well'
```

```

    }, {
name: 'IECE Inventory Controller Launch',
label: '2003: First Software Launch',
description: '4th December 2003, First Software launch with
name IECE Inventory Controller'
    }, {
name: 'IECE Digital launch',
label: '2018: IECE Digital founded',
description: 'IECE Digital launch, to provide world class
animation and VFX.'
    }]
  ]]
};
}

```

In Listing 6-28, the chart type is timeline. On the next line is a property called `inverted: true`, and this means you can see your timeline in vertical mode. If you want to develop in horizontal mode, make it `false`. Then next in the series section is a property called `dataLabels`; here you can set colors and width for the connector lines for the labels. Once you run the above code, you will get the output shown in Figure 6-18.



**Figure 6-18.** A timeline chart demo using Highcharts

## Gantt Chart

A Gantt chart is used to demonstrate project progress. Henry Gantt introduced the Gantt chart. This type of chart also shows project activities and current schedule status relationships.

A Gantt chart is a type of bar chart where the vertical axis can define the task to perform, and the horizontal axis can represent the time interval and progress.

If you want to develop a Gantt chart, the following dependencies are required:

### **jQuery:**

```
<script src="https://code.highcharts.com/ highcharts.js">
</script>
<script src="https://code.highcharts.com/gantt/highcharts-
gantt.js"></script>
```

### **Angular:**

```
import * as Highcharts from 'highcharts';
import GanttModule from 'highcharts/modules/gantt';
GanttModule(Highcharts);
```

Listing 6-29 had the code to show the progress of a software development project with the use of a Gantt chart. Copy Listing 6-29's code into the `app.component.ts` file.

### **Listing 6-29.** `app.component.ts`

```
import { Component } from '@angular/core';
import * as Highcharts from 'highcharts';
import GanttModule from 'highcharts/modules/gantt';
GanttModule(Highcharts);
```

```

Highcharts.setOptions({
  title: {
  style: {
  color: 'blue'
    }
  },
  legend: {
  enabled: false
  }
});

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  Highcharts: typeof Highcharts = Highcharts;
  chartGantt: Highcharts.Options = {

  xAxis: {
  min: Date.UTC(2019, 11, 1),
  max: Date.UTC(2019, 11, 30)
  },

  title: {
  text: 'IECE Inventory Controller Project Progress'
  },

  series: [{
  name: 'IECE Inventory Controller Project',
  type: 'gantt',

```

```

data: [{
  name: 'Start Project Requirement Analysis',
  start: Date.UTC(2019, 11, 5),
  end: Date.UTC(2019, 11, 15),
  completed: 0.90
},
{
  name: 'Development',
  start: Date.UTC(2019, 11, 11),
  end: Date.UTC(2019, 11, 22),
  completed: {
    amount: 0.35,
    fill: 'green'
  }
},
{
  name: 'Continuous Testing Software',
  start: Date.UTC(2019, 11, 15),
  end: Date.UTC(2019, 11, 29)
}]
];
}

```

In the `xAxis` section, you set the minimum (start date) and maximum date (end date) for the project.

In the series section, three new properties are added:

- `start`: This is the date-time property where you define the start date for the particular task.
- `end`: Here you set the end `DateTime` for the specific task.
- `completed`: Here you can define how much in percentage a particular task is complete.



You define `min` and `max`, where the start date and end date are set; the `completed` property is required to fill the value of how much in percentage this particular part is complete.

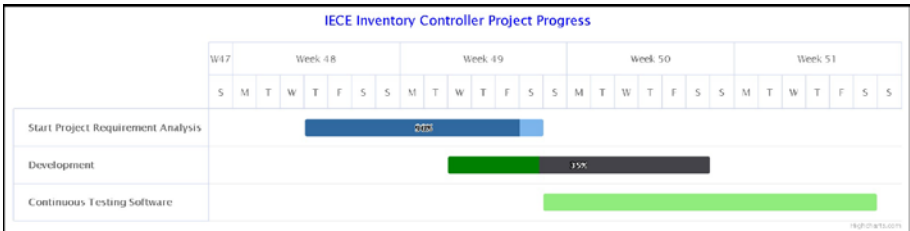
Now copy Listing 6-30's code into `app.component.html` to display the page.

**Listing 6-30.** `app.component.html`

```
<div>
<highcharts-chart [Highcharts]="Highcharts" [constructorType]="
'ganttChart' "
[options]="chartGantt" style="width: 100%; display: block;">
</highcharts-chart>
</div>

<router-outlet></router-outlet>
```

Once you run Listing 6-30's code, you will get the output shown in Figure 6-19.



**Figure 6-19.** Gantt chart demo with Highcharts

## Summary

In this chapter, you saw how easily you can set themes and styles and make your charts more interactive and understandable. You also saw some additional charts like 3D charts, organizational charts, Gantt charts, timelines, and bell curves. Highcharts is a stunning tool, and you can develop any chart based on your requirements. In the next chapter, I will go over how to build a real-time dashboard using Highcharts.

## CHAPTER 7

# Building a Real-Time Dashboard

In this chapter, you will learn how to show multiple real-time charts in a dashboard. To understand better how to easily consume a web API and develop a real-time dashboard, you will create one sample learning application. In this chapter, you will also learn some advanced concepts of Angular routing and the Forms module in order to quickly build your interactive app with the use of Angular and Highcharts.

## Real-Time Dashboard Application

In this sample learning e-portfolio application, you are going to develop features that provide live historical data from the stock market. Users can generate a portfolio of any stock and check their profit/loss. They can also check top loser and gainer stocks so that they can make better decisions about their investments. The idea behind developing this app is so you can understand and learn how easy it is to design interactive apps with the use of Angular and Highcharts. For this, you'll have two sections:

- Market radar section
- Dashboard section

For the market radar, the user can add their favorite stock into a database. Then they can select stock names, dates, and periods such as a daily or monthly basis, and they can see the performance of a particular stock very quickly.

In the dashboard section, users can analyze how much they invested and whether they have a profit or loss.

In Chapter 5, I talked about how you can easily consume a web API with Angular using .NET Core. For this application, you'll use the same Angular project, and if you want, you can develop a new Angular application using the Angular CLI, which I discussed in Chapter 3.

This app is designed with a client-server architecture model where you have two different projects. One is the UI side, which you developed in Angular with Highcharts. For the server side, you'll develop a .NET Framework-based web API.

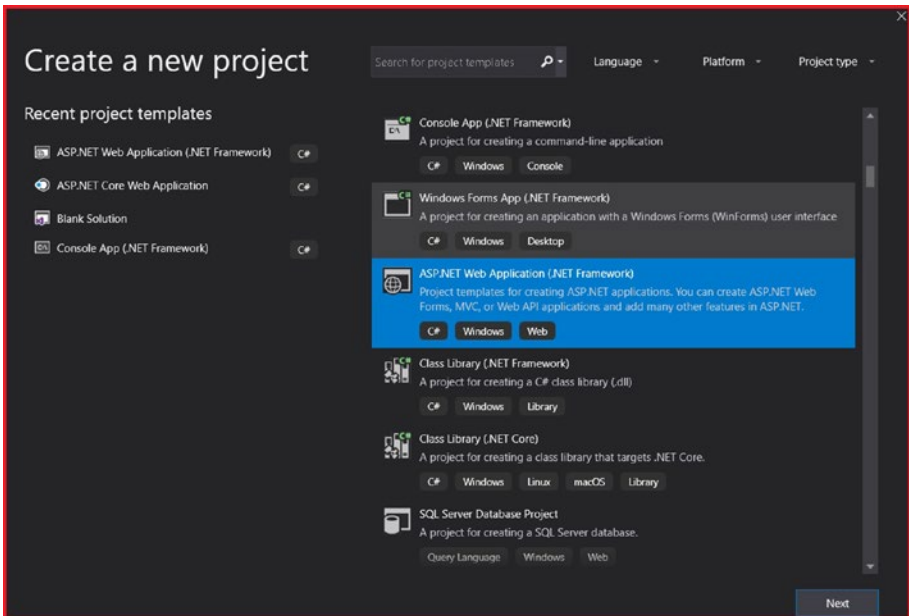
## Features of the App

For this app, the user will get three menus:

- **Add Stock:** Users can add their stocks in a database in the form of buying price, quantity, etc.
- **Market Radar:** The user can select whatever stocks you inserted into the database, from a drop-down list, with parameters of From Date, End Date, and a period type in the form of daily and monthly. Once the user clicks the Search button, the basis of the selected conditions will generate a chart.
- **Dashboard:** In this section, the user can check their list of invested stocks. They can see a profit/loss portfolio chart and a top gainer/loser investment chart. So this app will give them a better idea about their investments using Highcharts.

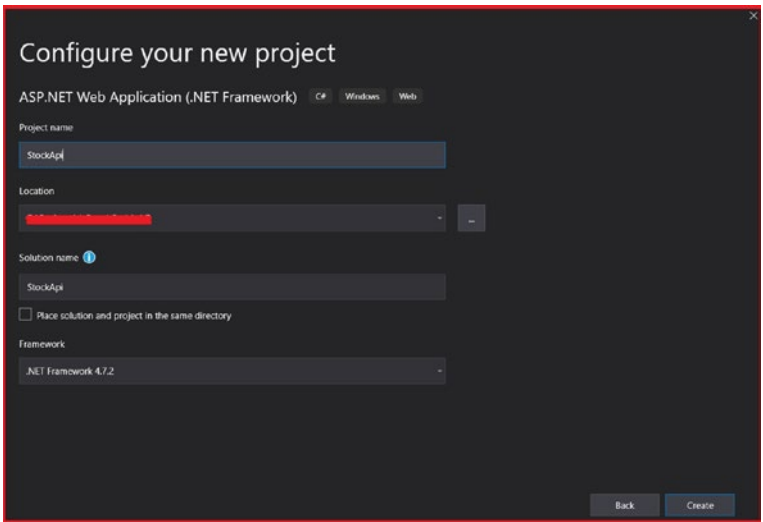
## Creating a Web API

For this application, you'll create a web API with the use of Visual Studio. Go to File ► New Project ► Select ► ASP.NET Web Application (.NET Framework) (Figure 7-1). In Chapter 5, you saw how to develop a web API using .NET core, so you'll use the .NET Framework this time. See Figure 7-1.



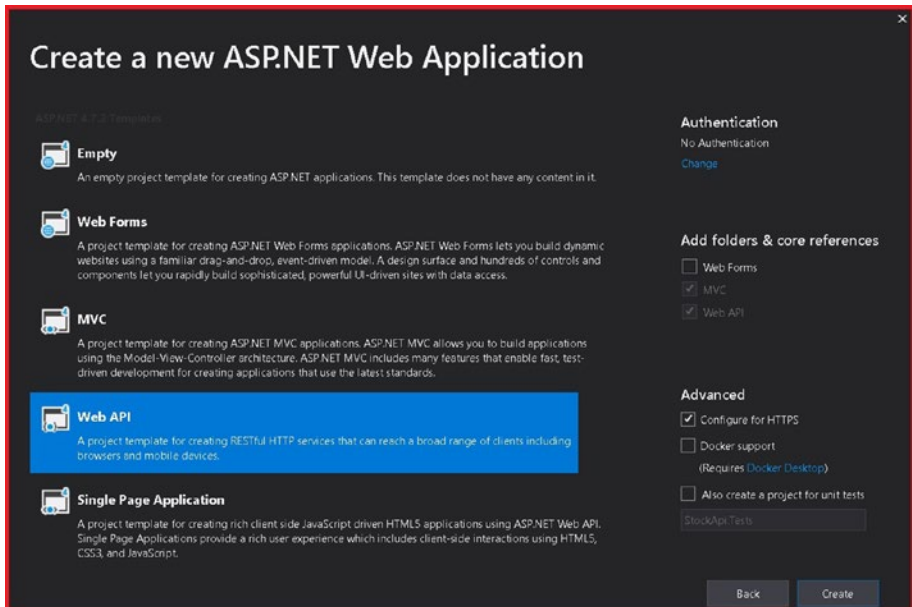
**Figure 7-1.** *Creating a new project*

Click the Next button (Figure 7-2).



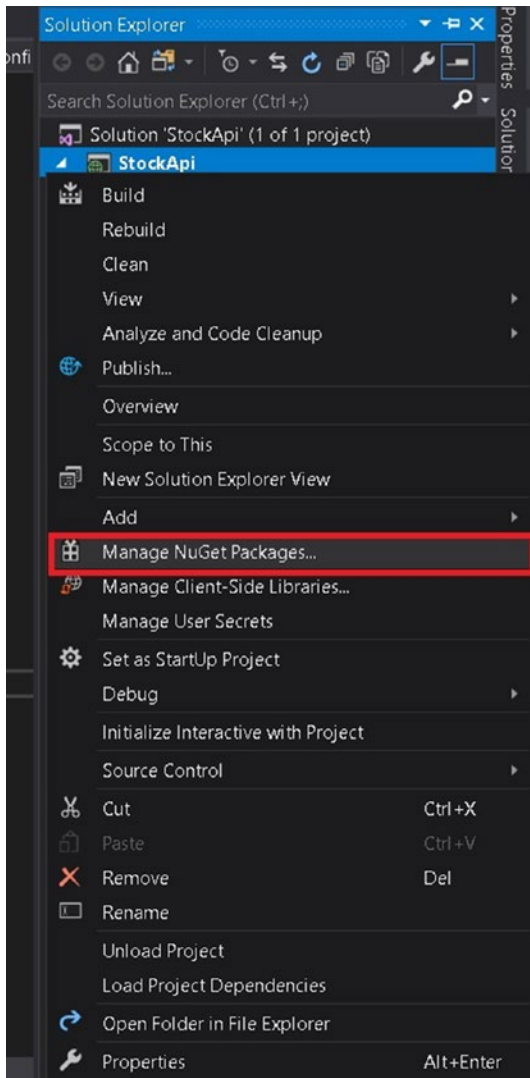
**Figure 7-2.** *Configuring a new project*

Here, you must provide the project name and you can choose the .NET Framework version. Click the Create button to go to the Create a new ASP.NET Web Application screen (Figure 7-3).



**Figure 7-3.** *Creating a new web API application*

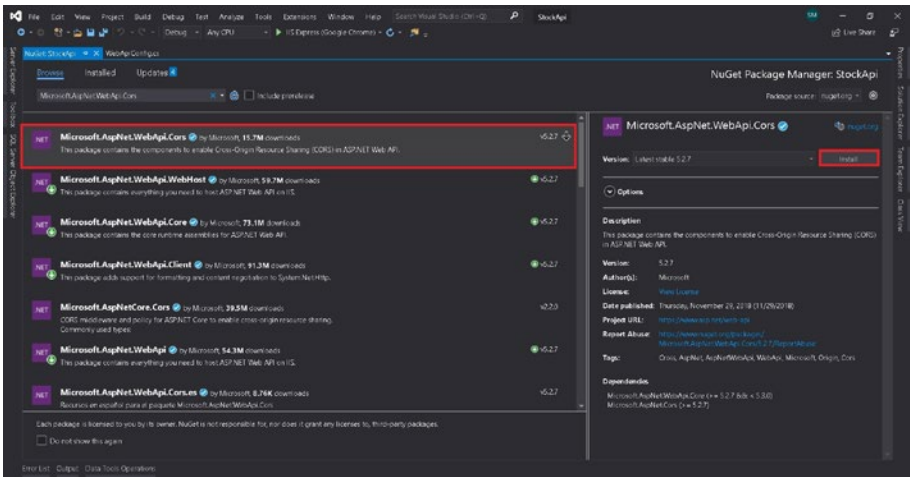
Click the Next button. You can see that your web API creation has completed. In the first step, you will call CORS-related dependencies because this web API will be consumed by the Angular app. So open Solution Explorer and right-click into the project and choose the Manage NuGet Package option (Figure 7-4).



**Figure 7-4.** *Managing the NuGet package for the project*

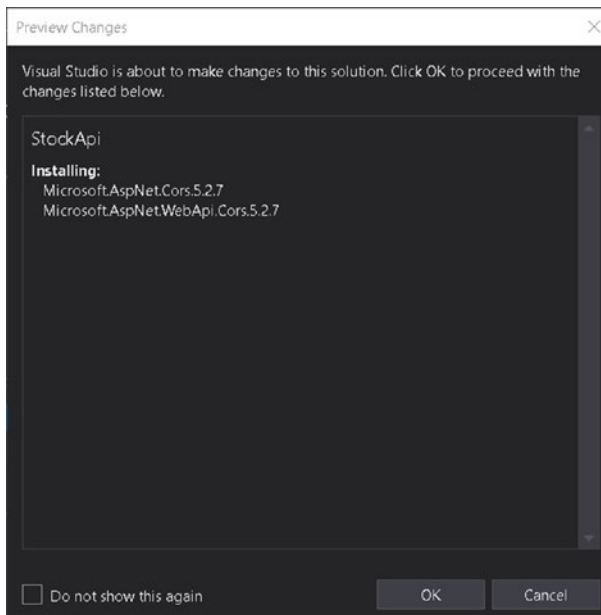
Now search for Microsoft.AspNet.WebApi.Cors. You will get the same list, so choose Microsoft.AspNet.WebApi.Cors and click the Install button (Figure 7-5).





**Figure 7-5.** Installing *Microsoft.AspNet.WebApi.Cors*

Now click the OK button (Figure 7-6).



**Figure 7-6.** Installing the *NugetPacakge* for the project

Now click the I Accept button. After this step, the NuGet package installation will complete.

Now it's time to set CORS, so open Solution Explorer ► App\_Start folder ► Open WebApiConfig.cs, and copy Listing 7-1's code.

**Listing 7-1.** WebApiConfig.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using System.Web.Http.Cors;

namespace StockApi
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services
            var cors = new EnableCorsAttribute
                ("http://localhost:4200", "*", "*");
            config.EnableCors(cors);
            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{action}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

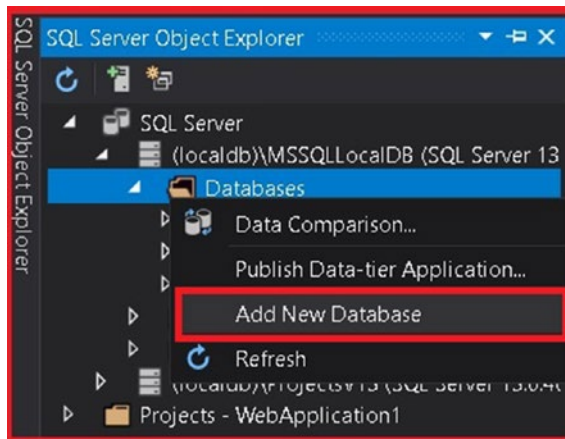
In this code, you enable CORS and CORS attributes so you can send a request from the Angular application to the web API very quickly. For related learning about CORS, please refer to Chapter 5 where I describe CORS in more detail.

## Setting Up a Database

This learning application requires one database because you want to work with live data. In this section, you are going to develop a database called StockDb. StockDb contains one table called StockMaster where you store your stock-related information. For real-time information, you will get data from a NuGet service, which is the Yahoo Finance API.

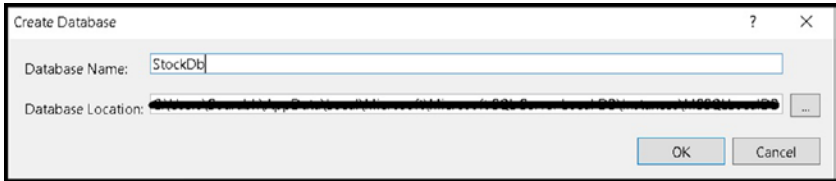
So let's create the database. Here you'll use Visual Studio, but you can also choose a platform per your requirements.

To create a database in Visual Studio, go to View ► SQL Server Object Explorer. Open a connection and right-click into the Database folder and select Add New Database (Figure 7-7).



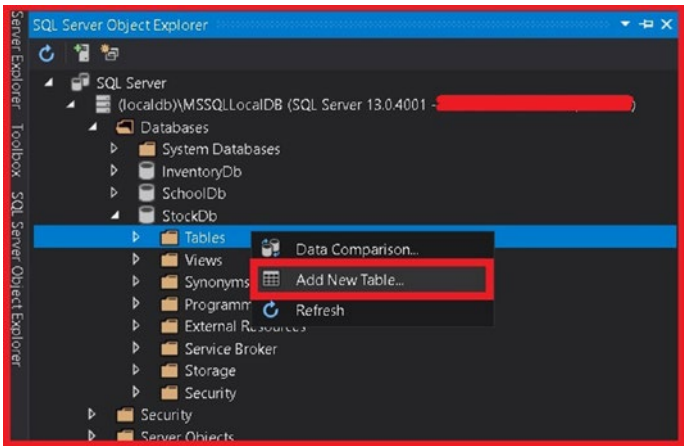
**Figure 7-7.** Adding a new database using Visual Studio

Once you click Add New Database, you will get the screen shown in Figure 7-8. Set the database name and click the OK button.



**Figure 7-8.** Create Database screen

Your database has been created. Now it's time to create the table. Expand StockDb and right-click the Table folder and choose Add New Table (Figure 7-9).



**Figure 7-9.** Add New Table screen

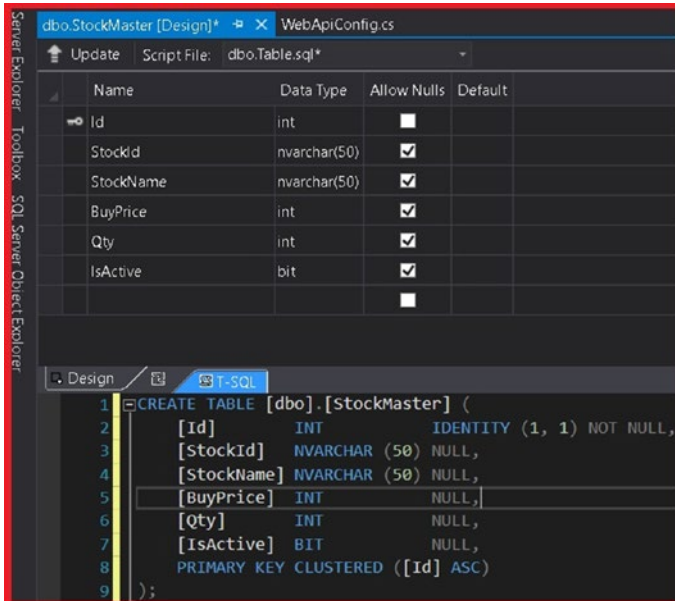
Now add the script in Listing 7-2 into the T-Sql Section (Figure 7-10).

**Listing 7-2.** StockMaster.sql

```

CREATE TABLE [dbo].[StockMaster](
    [Id]          INT IDENTITY (1, 1) NOT NULL,
    [StockId]    NVARCHAR (50) NULL,
    [StockName]  NVARCHAR (50) NULL,
    [BuyPrice]   INT NULL,
    [Qty]        INT NULL,
    [IsActive]   BIT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);

```

**Figure 7-10.** Adding the table script

Click the Update button. After that, you will get one dialog box so choose to Update Database and your table will be updated. Right-click into the StockMaster table and select ViewData. Now you can enter some data (Figure 7-11).

Id	StockId	StockName	BuyPrice	Qty	IsActive
1	MSFT	Microsoft	50	10	True
2	ibm	IBM	105	5	True
3	abb	ABB	NULL	NULL	False
4	GOOGL	google	NULL	NULL	False
5	MT	Arcelor Mittal	NULL	NULL	False
6	infy	Infosys	150	20	True
NULL	NULL	NULL	NULL	NULL	NULL

Figure 7-11. Adding data manually into a table

## Creating a Database First Approach Using Entity Framework

In this section, you will set up a database first approach. For more detail about the database first approach and Entity Framework, refer to Chapter 5, where I talk in detail about this process.

Right-click in Solution Explorer and select Add ► New Item (Figure 7-12).

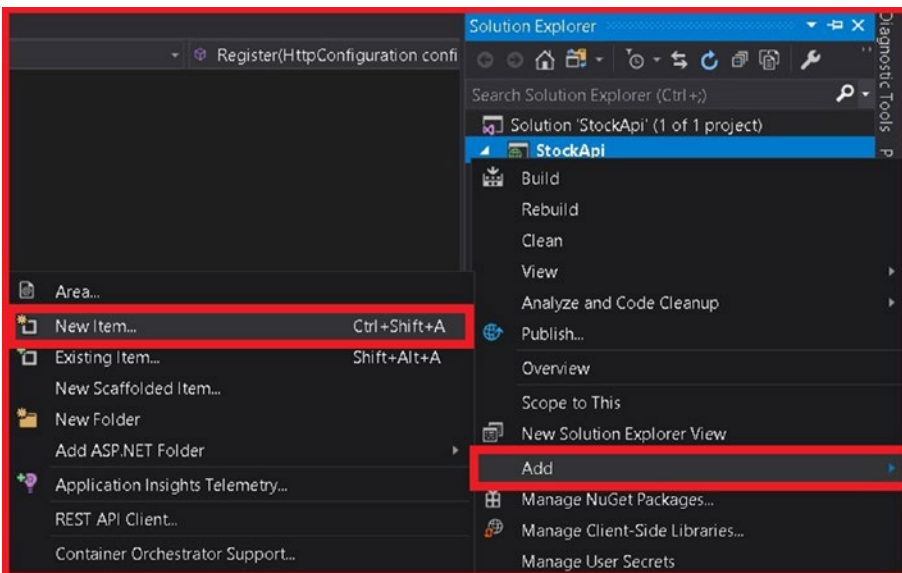
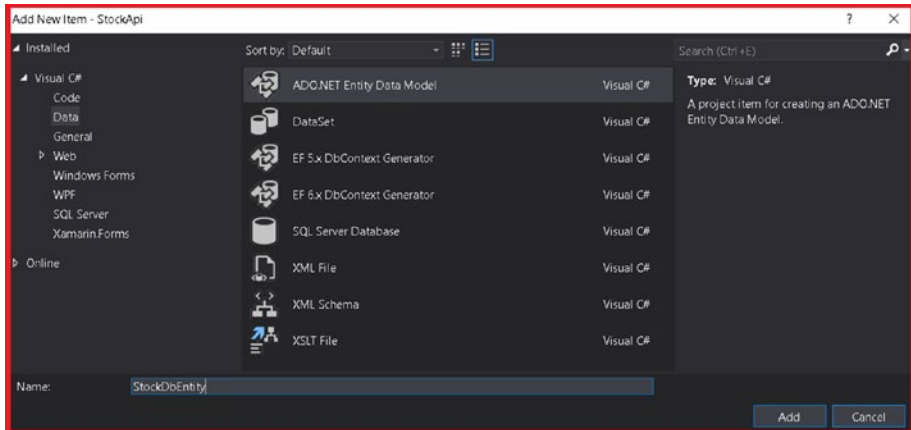


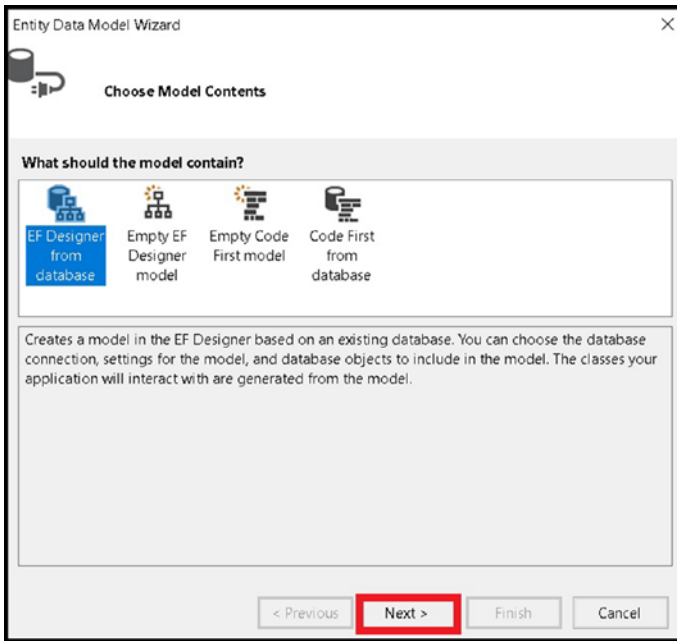
Figure 7-12. Add New Item screen

Once you click New Item, you will get the Add New Item screen. Choose the ADO.NET Entity Data Model, provide the name (StockDbEntity), and click the Add button (Figure 7-13).



**Figure 7-13.** Add New Item screen

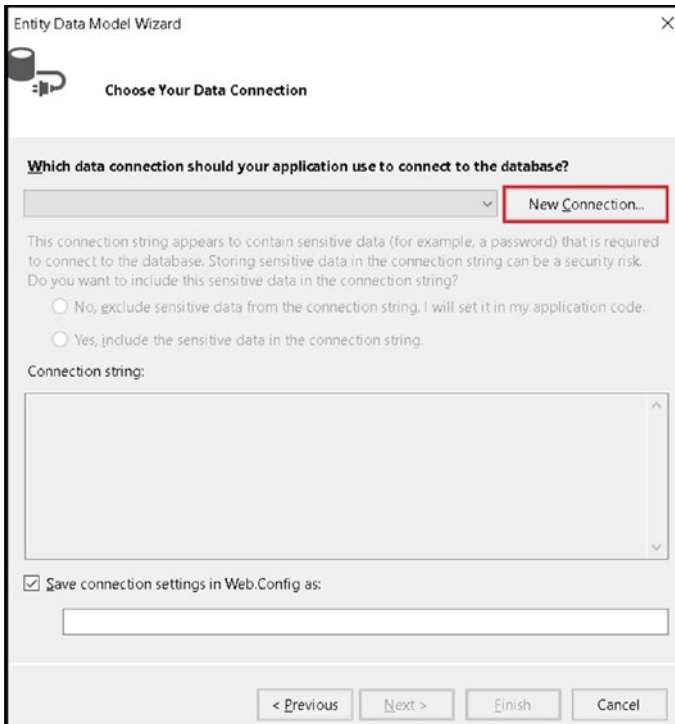
Next you'll see the Entity Data Model Wizard. Here you choose EF Designer from the database and click the Next button (Figure 7-14).



**Figure 7-14.** Entity Data Model wizard screen

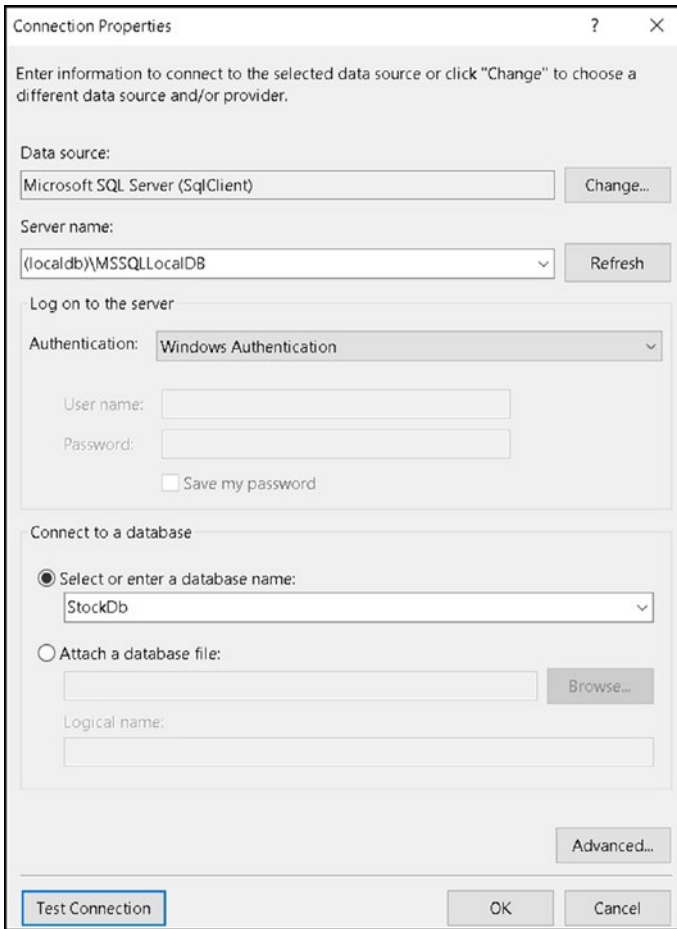
In the Choose Your Data Connection screen, click the New Connection button (Figure 7-15).





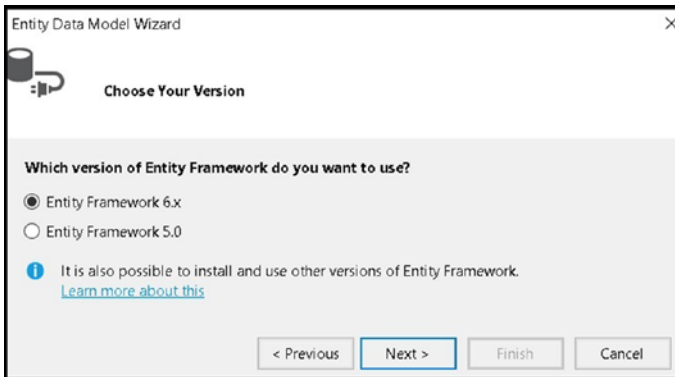
**Figure 7-15.** Choose Your Data Connection screen

In the Connection Properties screen, set the data source, server name, and database name. Once you set all these, click the Test Connection Button and then click the OK button (Figure 7-16).



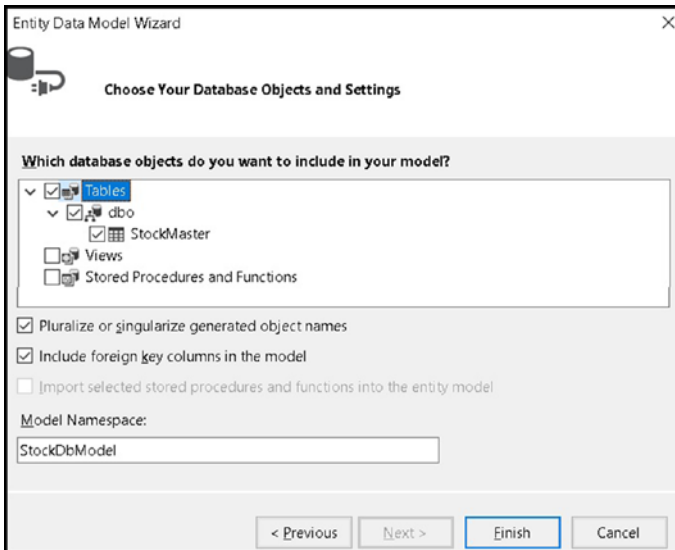
**Figure 7-16.** *Connection Properties screen*

Click the Next button to get to the Choose Your Version screen, where you'll select Entity Framework 6.X, and click the Next button (Figure 7-17).



**Figure 7-17.** Choose Your Version screen

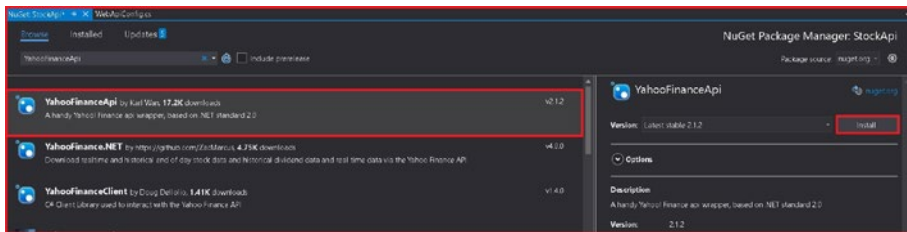
In the Choose Your Database Objects and Settings screen, expand the table and select the table name and then click the Finish button (Figure 7-18).



**Figure 7-18.** Choose Your Database Objects and Settings screen

Once you click the Finish button, the Entity Model for the StockDB database will be added to your Solution Explorer.

In this application, for learning purposes, you will use the Yahoo Finance API so people can get real data from the stock market. So open Solution Explorer and right-click in the project area and select Manage NuGet Packages. On the NuGet Package screen, click the Browse tab, search for YahooFinanceApi, and press Enter (Figure 7-19).



**Figure 7-19.** Installing the API through a NuGet package

Select YahooFinanceApi and click the Install button.

After this, you will get the Preview Changes screen dialog box. Click the Ok button; in the next step, click the I Accept button. Once you click it, this process will install the API in your project.

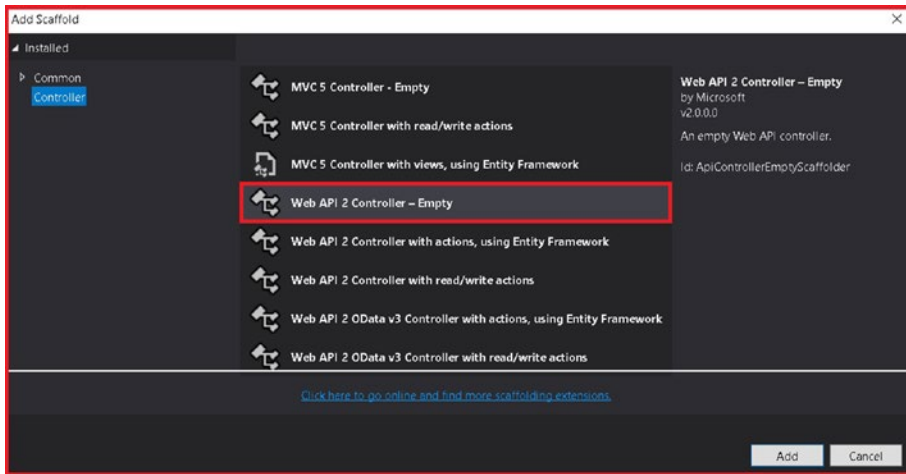
Now add the Model class. This will help in future development, so for this application you'll create a StockModel class. For this, go to Solution Explorer and right-Click in the Model class and provide the name for the class and click the Add button. Now add Listing 7-3's code to this StockModel class.

**Listing 7-3.** StockModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
namespace StockApi.Models
{
    public class StockModel
    {
        public string StockId { get; set; }
        public string StockName { get; set; }
        public string Date { get; set; }
        public decimal Open { get; set; }
        public decimal High { get; set; }
        public decimal Low { get; set; }
        public decimal Close { get; set; }
        public decimal Volume { get; set; }
        public int? BuyPrice{ get; set; }
        public int? Qty{ get; set; }
        public bool IsActive { get; set; }
        public int? TotalInvested{ get; set; }
        public int? CurrentValue{ get; set; }
        public int? TotalGain{ get; set; }
    }
}
```

Now it's time to create an API controller where you can write your business logic. Open Solution Explorer, right-click in the Controller folder and Add ► Controller ► Select **Web API 2 Controller - Empty** from the list (Figure 7-20).



**Figure 7-20.** Adding a new web API controller

Click the Add button. You will get the Add Controller screen. Provide your controller name. For this application, use `StockController`. After this action, you will see the `StockController` added into your `Controller` folder. This controller will start to add some methods, which will help to fetch and insert data.

So let's understand the methods.

- `AddStock()`: This method is used to add the new stock into the `StockMaster` table. In this method, you require a `StockId`. In this particular field, you have to provide a stock short name; every company has an alias in NASDAQ. For example, Microsoft is `MSFT`, Infosys is `INFY`, Google is `GOOGL`. You can quickly get these names from the NASDAQ website (for help, you can refer Figure 7-11), because the Yahoo Finance API will track data based on this stock id only, so please carefully add the stock ids; otherwise data will not

come. Then you need the stock name, buy price, total stock quantity, and IsActive. You write this method with the use of the Entity Framework.

- `GetStock()`: This method will fetch all active records from the `StockMaster` table and display the stock collection in the dropdown list for the market radar screen.
- `GetStockData()`: With the use of this method, the user can search their stock performance based on stock id, start date, end date, and period. So this method helps explore stock performance based on time.
- `GetActiveStock()`: This method provides the list of stocks where `IsActive = true` in the `StockMaster` table.
- `GetDashboardPortfolioData()`: This method provides a list of active stocks, total investment, current value of shares, and total gain so you can predict and develop charts very quickly.
- `GetGainerLoserStockData()`: This method provides data for the top gainer and top loser stock from the `StockMaster` table.

Now copy Listing 7-4's code into the `StockContoller.cs` file.

**Listing 7-4.** `StockController.cs`

```
using StockApi.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
```

```

using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using YahooFinanceApi;

namespace StockApi.Controllers
{
    public class StockController : ApiController
    {
        StockDbEntities _StockDbEntities = null;

        [Route("~/api/GetDashboardPortfolioData")]
        [HttpGet]
        public List<StockModel>GetDashboardPortfolioData()
        {
            _StockDbEntities = new StockDbEntities();
            var stockmasters = _StockDbEntities.StockMasters
                .Where(x =>x.IsActive == true).ToList();
            List<StockModel>stockModels = new
                List<StockModel>();
            stockmasters.ForEach(x => {
                var period = "daily";
                var p = Period.Daily;
                if (period.ToLower() == "weekly") p = Period.Weekly;
                else if (period.ToLower() == "monthly") p = Period.Monthly;
                var startDate = DateTime.Now.AddDays(-3);
                var endDate = DateTime.Now;

                var stockData = Yahoo.GetHistoricalAsync(x.StockId, startDate,
                    endDate, p).Result;
                if (stockData.Count > 0)
                {
                    StockModel stockModel = new StockModel();

```



```

var lastRecord = stockData.LastOrDefault();
stockModel.StockId = x.StockId;
stockModel.StockName = x.StockName;
stockModel.BuyPrice = x.BuyPrice;
stockModel.Qty = x.Qty;
stockModel.TotalInvested = x.Qty * x.BuyPrice;
stockModel.CurrentValue = x.Qty * (int)lastRecord.
AdjustedClose;
stockModel.TotalGain = stockModel.CurrentValue - stockModel.
TotalInvested;
stockModels.Add(stockModel);
        }
    });
return stockModels.OrderBy(c =>c.TotalGain).ToList();
}

[Route("~/api/GetStock")]
[HttpGet]
public List<StockModel>GetStock()
{
    _StockDbEntities = new StockDbEntities();
    List<StockModel>stockModels = new
List<StockModel>();
var query = _StockDbEntities.StockMasters.ToList();
query.ForEach(x =>
    {
        StockModel stockModel = new StockModel();
        stockModel.StockId = x.StockId;
        stockModel.StockName = x.StockName;
        stockModels.Add(stockModel);
    });
}

```

```

return stockModels;
    }

    [HttpPost]
    public bool AddStock(StockModel stockModel)
    {
        try
        {
            _StockDbEntities = new StockDbEntities();
            StockMaster stockMaster = new StockMaster();
            stockMaster.StockId = stockModel.StockId;
            stockMaster.StockName = stockModel.StockName;
            stockMaster.BuyPrice = stockModel.BuyPrice;
            stockMaster.Qty = stockModel.Qty;
            stockMaster.IsActive = stockModel.IsActive;
            _StockDbEntities.StockMasters.Add(stockMaster);
            _StockDbEntities.SaveChanges();
        }
        catch (Exception ex)
        {
            return false;
        }
        return true;
    }

    [HttpGet]
    public List<StockMaster> GetActiveStock()
    {
        _StockDbEntities = new StockDbEntities();
        var stockmasters = _StockDbEntities.StockMasters.Where(x =>x.
            IsActive == true).ToList();
        return stockmasters;
    }

```

```

    [Route("~/api/GetStockData/{ticker}/{start}/{end}/
      {period}")]
    [HttpGet]
    public async Task<List<StockModel>>GetStockData(string
    ticker = "", string start = "",
    string end = "", string period = "")
        {
        var p = Period.Daily;
        if (period.ToLower() == "weekly") p = Period.Weekly;
        else if (period.ToLower() == "monthly") p = Period.Monthly;
        var startDate = DateTime.Parse(start);
        var endDate = DateTime.Parse(end);

        var query = await Yahoo.GetHistoricalAsync(ticker, startDate,
        endDate, p);

        List<StockModel> models = new List<StockModel>();
        foreach (var r in query)
            {
            models.Add(new StockModel
                {
                StockName = ticker,
                    Date = r.DateTime.ToString("yyyy-MM-dd"),
                    Open = r.Open,
                    High = r.High,
                    Low = r.Low,
                    Close = r.Close,
                    Volume = r.Volume
                });
            }
        return models;
        }

```

```

        [Route("~/api/GetGainerLoserStockData/{ticker}/
        {period}")]
        [HttpGet]
public async Task<List<StockModel>>GetGainerLoserStockData
(string ticker = "",
string period = "")
    {
var p = Period.Daily;
if (period.ToLower() == "weekly") p = Period.Weekly;
else if (period.ToLower() == "monthly") p = Period.Monthly;
var startDate = DateTime.Now.AddMonths(-11);
var endDate = DateTime.Now;
var query = awaitYahoo.GetHistoricalAsync(ticker, startDate,
endDate, p);

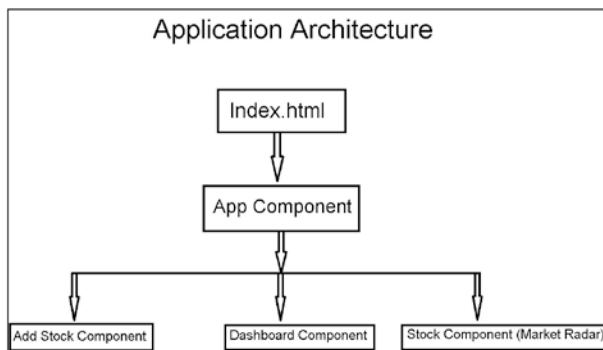
    List<StockModel> models = new List<StockModel>();
    foreach (var r in query)
        {
models.Add(new StockModel
            {
StockName = ticker,
                Date = r.DateTime.ToString("yyyy-MM-dd"),
                Open = r.Open,
                High = r.High,
                Low = r.Low,
                Close = r.Close,
                Volume = r.Volume
            });
        }
return models;
    }
}
}

```

Press F5 and run the code. Your web API will start running. Now let's develop an Angular app using Highcharts. If you want to learn how to configure the new Angular app, please refer to Chapter 3.

## Routing in an Angular App

For navigation between different pages, routing plays an essential role. Every application has different pages, and routing is a way for the user to communicate between different pages. Figure 7-21 describes the application architecture. For this application, you have `index.html`, which calls `AppComponent`. In this part, all menus are configured; when a user clicks a menu, that respective component is called.



**Figure 7-21.** Application architecture for the e-portfolio learning app

If you want to configure routing in your application, first you must import `RouterModule` from `'@angular/router'` into the `app.module.ts` file and then, with the use of `.forRoot([ ])`, define the path and component properties.

- `path`: Here you can define the URL. For example, when a user clicks the dashboard, it will redirect to the `localhost/dashboard`. Whatever name you specify into

the path area the same name, you have to define in an anchor tag in the Html page. In the next example, you will understand this in more detail.

- **component:** In this property, you define the name of the component you want to call at the time of the menu click.

**Example:**

```
import {RouterModule} from '@angular/router'
imports: [
  RouterModule.forRoot([
    { path: 'stock', component: StockComponent },
    { path: 'addstock', component: AddstockComponent },
    { path: 'dashboard', component: DashboardComponent },
  ])
]
```

In the above example, in the `.forRoot` method array, you define path and component for all three menus. Now copy Listing 7-5's code into the `app.module.ts` file so you can enable routing in your application.

**Listing 7-5.** `app.module.ts`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HighchartsChartComponent } from 'highcharts-angular';
import { RouterModule } from '@angular/router'
import { ReactiveFormsModule } from "@angular/forms";
import { StockComponent } from './stock/stock.component';
```

```

import { AddstockComponent } from './addstock/addstock.
component';
import { DashboardComponent } from './dashboard/dashboard.
component';
@NgModule({
  declarations: [
    AppComponent,
    HighchartsChartComponent,
    StockComponent,
    AddstockComponent,
    DashboardComponent,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    ReactiveFormsModule,
    RouterModule.forRoot([
      { path: 'stock', component: StockComponent },
      { path: 'addstock', component: AddstockComponent },
      { path: 'dashboard', component: DashboardComponent },
    ])
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Now open `app.component.ts` and add Listing 7-6's code.

**Listing 7-6.** app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor() {
  }
  ngOnInit() {
  }
}
```

In this file, nothing is special because `app.component.ts` only provides the HTML and CSS; when the user clicks the hyperlinks, it will redirect to a different component. Now copy Listing 7-7's code into `app.component.css`.

**Listing 7-7.** app.component.css

```
.topnav {
background-color: #333;
overflow: hidden;
}
/* Style the links inside the navigation bar */
.topnav a {
float: left;
color: #f2f2f2;
text-align: center;
padding: 14px 16px;
text-decoration: none;
```



```
font-size: 17px;
}
/* Change the color of links on hover */
.topnav a:hover {
background-color: #ddd;
color: black;
}
/* Add a color to the active/current link */
.topnava.active {
background-color: rgb(221, 97, 25);
color: white;
}
```

Open `app.component.html` and copy Listing 7-8's code. Here you configure your menu hyperlinks.

**Listing 7-8.** `app.component.html`

```
<div class="topnav">
<a class="active" [routerLink]="['/dashboard']">Dashboard</a>
<a [routerLink]="['/stock']">Market Radar</a>
<a [routerLink]="['/addstock']">Add Stocks</a>
</div>
<br/>
<router-outlet></router-outlet>
```

In Listing 7-8, the `[routerLink]` directive is used to open your routing path at the time of the click. This is directly linked with `RouterModule.forRoot([])`, which you define in `app.module.ts`.

Now it's time to add a model class for your Angular application. The class name is `stockmodel.ts`, so open a new terminal window in Visual Studio and type the following command and press Enter. This command will add a `stockmodel.ts` file into the `model` folder:

**ng generate class model/stockmodel**

Now copy Listing 7-9's code into the `stockmodel.ts` file.

**Listing 7-9.** `stockmodel.ts`

```
export class Stockmodel {
  public StockName: string;
  public Date: string;
  public Open: number;
  public High: number;
  public Low: number;
  public Close: number;
  public Volume: number;
  public StockId:string;
  public BuyPrice:number;
  public Qty: number;
  public IsActive:boolean;
  public TotalInvested:number;
  public CurrentValue:number;
  public TotalGain:number;
}
```

You can see that this model class property is the same as you created in the web API model class. The reason to create this file is, whenever you send the request to the web API, you will receive data into the `stockmodel` at the time of response. After the response, this model will also help generate data binding into different charts.

Now it's time to add a service to your Angular application. In this application, you have only one service file, which will communicate with the web API to insert and fetch records based on the requirements. To create a service file, type the following command into the terminal window of Visual Studio. This command will create a `stock.service.ts` file in the `services` folder.

### **ng generate service services/stock**

Now, copy Listing 7-10's code into the `stock.service.ts` file.

#### ***Listing 7-10.*** `stock.service.ts`

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Stockmodel } from '../model/stockmodel';
import { Observable } from 'rxjs';
import { tick } from '@angular/core/testing';
@Injectable({
  providedIn: 'root'
})
export class StockService {
  constructor(private http: HttpClient) {
    console.log('Stock Service called');
  }
  GetStockByTicks(url, ticker:string,start:string,end:string,
  period:string): Observable<Stockmodel[]> {
    return this.http.get<Stockmodel[]>(url+"/"+ticker+"/"+start+
    "/" +end+"/"+"/" +period);
  }
  GetGainerLoserStockData(url, ticker:string,period:string):
  Observable<Stockmodel[]> {
```

```

return this.http.get<Stockmodel[]>(url+"/"+ticker+"/"+
"/"+period);
}

GetStocks(url): Observable<Stockmodel[]> {
return this.http.get<Stockmodel[]>(url);
}

addStock(url,stockmodel: Stockmodel) {
return this.http.post(url, stockmodel);
}
}

```

If you want to learn more about `httpClient`, `Observable`, and `providerIn`, please refer to Chapter 5, where I describe how to communicate with the Angular service in more detail.

Now one by one you will add new components and connect them with `stock.service.ts`.

The first section creates an `addstock.component.ts` file, which is responsible for inserting stock records into the database.

Type the following command into the terminal window of Visual Studio to generate the `addstock.component.ts` file:

### **ng generate component addstock**

Now copy Listing 7-11's code into the `addstock.component.ts` file.

#### ***Listing 7-11.*** `addstock.component.ts`

```

import { Component, OnInit } from '@angular/core';
import { Stockmodel } from '../model/stockmodel';
import { StockService } from '../services/stock.service';
import { FormBuilder, FormGroup, Validators } from "@angular/
forms";
import { element } from 'protractor';

```

```

@Component({
  selector: 'app-addstock',
  templateUrl: './addstock.component.html',
  styleUrls: ['./addstock.component.css']
})
export class AddstockComponent implements OnInit {
  url: string = 'https://localhost:44311/api/Stock/AddStock';
  addForm: FormGroup;

  constructor(private stockService: StockService, private
  formBuilder: FormBuilder) { }

  ngOnInit() {
    this.addForm = this.formBuilder.group({
      StockId: ['', Validators.required],
      StockName: ['', Validators.required],
      BuyPrice: ['', Validators.required],
      Qty: ['', Validators.required],
      IsActive: ['', Validators.required],
    });
  }

  postApiResponse(formVal, url) {
    return this.stockService.addStock(this.url, formVal)
      .toPromise().then(res => {
        return res;
      });
  }

  onSubmit() {
    console.log(this.addForm.value)

    this.postApiResponse(this.addForm.value, this.url).then(
      data => {

```

```

        if(data===true)
        {
            alert('Stock added Successfully')
        }
        else{
            alert('Stock not added Successfully')
        }
    });
}
}

```

In Listing 7-11, you have the `onSubmit()` method, which will call from the page Add Stock button click. Once the user fills all the details and clicks the Add Stock button, this `onSubmit()` method will send the request to the `postApiResponse()` method with its parameter.

This method will forward the request to the `stockservice.addStock()` method, and at the time of return, you will get an alert Success message on the page.

Here you must remember one thing: once you run your web API application, you will get a port like `(:44311) https://localhost:44311/`. So whatever port number you get for your web API, use the same port in the URL variable in the `addstock.component.ts` file; otherwise, the request will never go to the web API. Follow this practice for all upcoming components.

Now copy Listing 7-12's code into `addstock.component.html`, where you define the page for this component for adding stocks.

**Listing 7-12.** `addstock.component.html`

```

<form [formGroup]="addForm" novalidate class="form">
  <h2>Add Stock Portfolio</h2>
  <table>

```

```
<tr>
  <td>Stock Id (Ex. Msft, infy, abb)</td>
  <td>
    <input type="text" id=txtStockId
      formControlName="StockId"/>
  </td>
</tr>
<tr>
  <td>Stock Name</td>
  <td>
    <input type="text" id=txtStockName
      formControlName="StockName"/>
  </td>
</tr>
<tr>
  <td>Buy Price</td>
  <td>
    <input type="text" id=txtBuyPrice
      formControlName="BuyPrice"/>
  </td>
</tr>
<tr>
  <td>Qty</td>
  <td>
    <input type="text" id=txtQty
      formControlName="Qty"/>
  </td>
</tr>
<tr>
  <td>IsActive into Portfolio</td>
```

```

        <td>
            <input type="checkbox" id=txtIsActive
            formControlName="IsActive"/>
        </td>
    </tr>
    <tr>
        <td>
            <input type="button" id="btnAdd" value="Add"
            (click)="onSubmit()"/>
        </td>
    </tr>
</table>
</form>

```

Now it's time to add `stock.component.ts`, where you define your market radar and where users can search for stock performance daily and monthly based on selected dates. Type the following command into the terminal window of Visual Studio:

### **ng generate component stock**

This command will add the `stock.component.ts` file into your folder structure.

Now copy Listing 7-13's code into the `stock.component.ts` file.

#### **Listing 7-13.** `stock.component.ts`

```

import { Component, OnInit } from '@angular/core';
import { Stockmodel } from '../model/stockmodel';
import { StockService } from '../services/stock.service';
import { FormBuilder, FormGroup, Validators } from "@angular/
forms";
import * as Highcharts from 'highcharts';
import { debug } from 'util';

```



```

@Component({
  selector: 'app-stock',
  templateUrl: './stock.component.html',
  styleUrls: ['./stock.component.css']
})

export class StockComponent implements OnInit {
  url: string = 'https://localhost:44311/api/GetStockData';
  addForm: FormGroup;
  start:string
  end:string
  stockDates: any;
  stockModel: Stockmodel[];
  SelStockId: string;
  SelPeriodId: string;
  Stocks:Stockmodel[];
  constructor(private stockService: StockService, private
  formBuilder: FormBuilder) {
  }
  public options: any = {
  chart: {
  type: 'line',
  },
  title: {
  text: 'E- Portfolio'
  },
  credits: {
  enabled: false
  },
  xAxis: {
  categories: [],
  },

```

```

yAxis: {
  title: {
    text: ''
  },
},
series: [],
}
ngOnInit() {
  this.StockDDL();
  this.addForm = this.formBuilder.group({
    Stock: ['', Validators.required],
    Period: ['', Validators.required],
    StartDate: ['', Validators.required],
    EndDate: ['', Validators.required],
  });
}
onSubmit() {
  this.stockService.GetStockByTicks(this.url, this.SelStockId,
  this.start, this.end, this.SelPeriodId)
    .toPromise().then(data => {
const stockData = [];
const dates = [];
data.forEach(row => {
const temp_row = [
row.High,
];
dates.push(row.Date);
stockData.push(row.High);
});
this.stockModel = stockData;
this.stockDates = dates;

```

```

var dataSeries = [];
for (var i = 0; i<this.stockModel.length; i++) {
dataSeries.push(
this.stockModel[i]
    );
}
this.options.series = [{ data: dataSeries, name: this.
SelStockId }]
this.options.xAxis.categories = this.stockDates
Highcharts.chart('container', this.options);
    },
error => {
console.log('Something went wrong. ');
    });
}
StockDDL()
{
this.stockService.GetStocks("https://localhost:44311/api/
getstock")
    .toPromise().then(data => {
const stockLData = [];
data.forEach(row => {
stockLData.push({
StockName: row.StockName,
StockId:row.StockId
    });
});
return this.Stocks = stockLData;
    });
}
}

```

In Listing 7-13, you have three methods. The first is `StockDDL()`, which is responsible for fetching data from the `StockMaster` table and binding it into the dropdown list; this method calls the `ngOnInit()` method. As you already know, the `ngOnInit()` method runs automatically at the time of page load, so whenever this page loads the first time, this will bind active stock information into the dropdown list.

Next is the `onSubmit()` method. This method generates an event once the user clicks the search button. So this method sends all field parameters to the `GetstockByTicks()` service method. At the time of response, it collects information from the Yahoo API service based on data that will populate the line charts using Highcharts. If you want to learn this process in more detail, please refer to Chapter 5.

Now copy Listing 7-14's code into the `stock.component.html` file.

**Listing 7-14.** `stock.component.html`

```
<form [formGroup]="addForm" novalidate class="form">
<div>
  <hr/>
  <table>
    <tr>
      <td>
        Select Stock*
      </td>
      <td>
        <select class="form-control"
          formControlName="Stock" [(ngModel)]="selStockId">
          <option *ngFor="let Stock of Stocks"
            value={{Stock.StockId}}>
            {{Stock.StockName}}
          </option>
        </select>
      </td>
    </tr>
  </table>
</div>
```

```

<td>
  Period*
</td>
<td>
  <select class="form-control"
    formControlName="Period" [(ngModel)]="SelPeriodId">
    <option selected value="daily">Daily</option>
    <option value="monthly">Monthly</option>
  </select>
</td>
<td>
  Start Date*
</td>
<td>
  <input type="date" id="txtStartDate"
    formControlName="StartDate" [(ngModel)]="start"/>
</td>
<td>
  End Date*
</td>
<td>
  <input type="date" id="txtEndDate"
    formControlName="EndDate" [(ngModel)]="end"/>
</td>
<td>
  <input type="button" id="btnAdd" value="Search"
    (click)="onSubmit()"/>
</td>
</tr>
</table>
<hr/>
</div>

```

```

<div class="content" id="container" role="main">
</div>

</form>

<router-outlet></router-outlet>

```

In the Listing 7-14 code, your chart will populate the container `<div>` once the user clicks the Search button.

Now, after adding the stock and market radar features, you can start the development of the dashboard feature. Type the following command into the terminal window of Visual Studio:

### **ng generate component dashboard**

In the `dashboard.component.ts` code, there are three sections. In the top section, you get all the stocks tables, which you added into the `StockMaster` table. After that there is a button called Dashboard Chart. When you click this button, you will get three charts in the dashboard.

1. **Profit/Loss chart:** This is a pie chart that tells you about the profit/loss of the portfolio.
2. **Top Gainer Stock chart:** This is a line chart. It finds and shows the most profitable stock from your portfolio and its performance.
3. **Top Loser Stock chart:** This is a line chart; it finds the worst performing stock from your portfolio and displays its performance.

Now copy Listing 7-15's code into the `dashboard.component.ts` file.

#### **Listing 7-15.** `dashboard.component.ts`

```

import { Component, OnInit } from '@angular/core';
import { Stockmodel } from '../model/stockmodel';
import { StockService } from '../services/stock.service';

```

```

import {FormBuilder, FormGroup, Validators} from "@angular/
forms";
import * as Highcharts from 'highcharts';
import { debug } from 'util';
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  addForm: FormGroup;
  url: string = 'https://localhost:44311/api/GetStockData';
  Stocks: Stockmodel[];
  PortfolioStocks : Stockmodel[]; //For holding Portfolio data
  with profit/loss.
  constructor(private stockService: StockService, private
  formBuilder: FormBuilder) {
  }
  ngOnInit() {
    // this.GetActiveStocks();
    this.GetDashboardPortfolioData();
  }
  GetDashboardPortfolioData()
  {
    this.stockService.GetStocks("https://localhost:44311/api/
    GetDashboardPortfolioData")
    .toPromise().then(data => {
      return this.PortfolioStocks = data;
    });
  }
}

```

```

public profitLossChart: any = {
  chart: {
    type: 'pie',
  },
  title: {
    text: 'Profit/Loss Chart'
  },
  credits: {
    enabled: false
  },
  series: [],
}

GetProfitLossChart(){
  let totalInvestment:number=0;
  let totalGain:number=0;
  this.PortfolioStocks.forEach(row=>{
    totalInvestment+=row.TotalInvested;
    totalGain += row.CurrentValue;
  });
  this.GetTopGainerChart();
  this.GetTopLoserChart();
  this.profitLossChart.series =[{
    data: [{
      name: 'Total Investment#',
      y: totalInvestment,
    },
    {
      name: 'Current Value',
      y: totalGain,
    }
  ]
}]
}

```



```

    Highcharts.chart('containerProfitLoss', this.
    profitLossChart);
}

public topGainerChart: any = {
  chart: {
    type: 'line',
  },
  title: {
    text: 'Top Gainer'
  },
  credits: {
    enabled: false
  },
  xAxis: {
    categories: [],
  },
  yAxis: {
    title: {
      text: ''
    },
  },
  series: [],
}

GetTopGainerChart() {
  let length = this.PortfolioStocks.length;
  if (length > 0) {
    this.stockService.GetGainerLoserStockData('https://
    localhost:44311/api/GetGainerLoserStockData', this.
    PortfolioStocks[length - 1].StockId, "Monthly")
    .toPromise().then(data => {

```

```

const stockData = [];
const dates = [];
data.forEach(row => {
  const temp_row = [
    row.High,
  ];
  dates.push(row.Date);
  stockData.push(row.High);
});
var dataSeries = [];
for (var i = 0; i < stockData.length; i++) {
  dataSeries.push(
    stockData[i]
  );
}

this.topGainerChart.series = [{ data: dataSeries, name:
this.PortfolioStocks[length - 1].StockId }]
this.topGainerChart.xAxis.categories = dates
Highcharts.chart('topGainerChart', this.
topGainerChart);
},
error => {
  console.log('Something went wrong.');
```

```

});
}
}

```

```

public topLoserChart: any = {
  chart: {
    type: 'line',
  },
},

```

```

title: {
  text: 'Top Loser'
},
credits: {
  enabled: false
},
xAxis: {
  categories: [],
},
yAxis: {
  title: {
    text: ''
  },
},
series: [],
}

```

```

GetTopLoserChart() {
  let length = this.PortfolioStocks.length;
  if (length > 0) {
    this.stockService.GetGainerLoserStockData('https://
localhost:44311/api/GetGainerLoserStockData',
this.PortfolioStocks[0].StockId, "Monthly")
    .toPromise().then(data => {
      const stockData = [];
      const dates = [];
      data.forEach(row => {
        const temp_row = [
          row.High,

```



3. `GetTopGainerChart()`: This method sends the request to the `GetGainerLoserStockData()` service method, and at the time of return, generates a data series for a line chart.
4. `GetTopLoserChart()`: This method sends the request to the `GetGainerLoserStockData()` service method. At the time of response, it generates a data series for a line chart for the worst performing stock into the portfolio.
5. `GetProfitLossChart()`: This method binds a data series for the pie chart. This is reflected in the dashboard, after a dashboard button click. Internally it calls the `GetTopGainerChart()` and `GetTopLoserChart()` methods.

Now copy Listing 7-16's code into `dashboard.component.html`.

**Listing 7-16.** `dashboard.component.html`

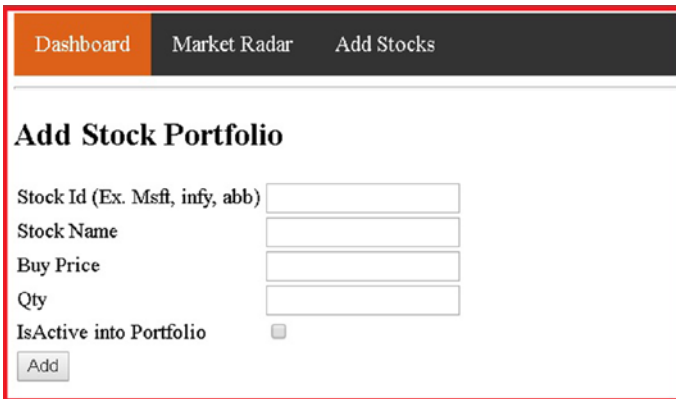
```
<table border="1" style="border-color: black; border-collapse: collapse; margin-left: 35%">
  <thead>
    <tr>
      <th>StockId</th>
      <th>Stock Name</th>
      <th>Buy Price</th>
      <th>Qty</th>
      <th>TotalInvested</th>
      <th>CurrentValue</th>
      <th>TotalGain</th>
    </tr>
  </thead>
```

```
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| {{stock.StockId}} | {{stock.StockName}} | {{stock.BuyPrice}} | {{stock.Qty}} | {{stock.TotalInvested}} | {{stock.CurrentValue}} | {{stock.TotalGain}} |


|                                                                               |                                                                     |
|-------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <div class="containerProfitLoss" id="containerProfitLoss" role="main"> </div> | <div class="topGainerChart" id="topGainerChart" role="main"> </div> |
|-------------------------------------------------------------------------------|---------------------------------------------------------------------|


```

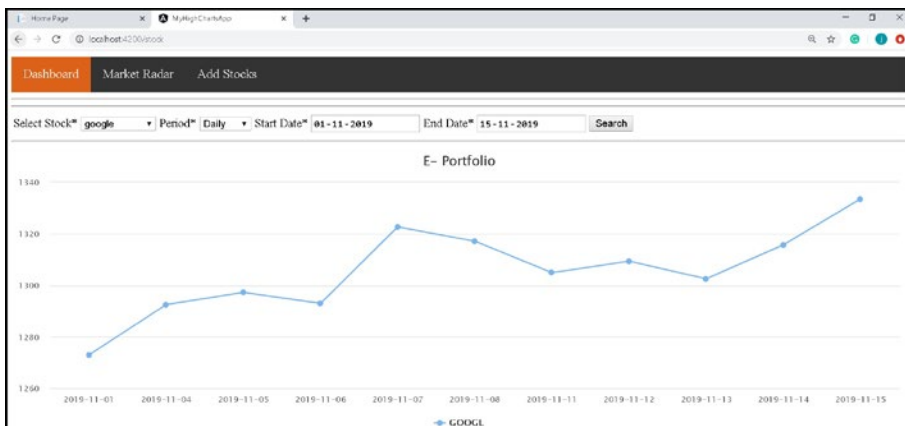




**Figure 7-23.** Add Stock Portfolio screen

After you fill in all the fields and click the Add button, it will store data in the StockMaster table. After insertion completes, you will get a Success alert message. Don't forget to write the exact stock id or the Yahoo Finance web API will never return the right results. To get the stock id symbols, you can refer to the NASDAQ website.

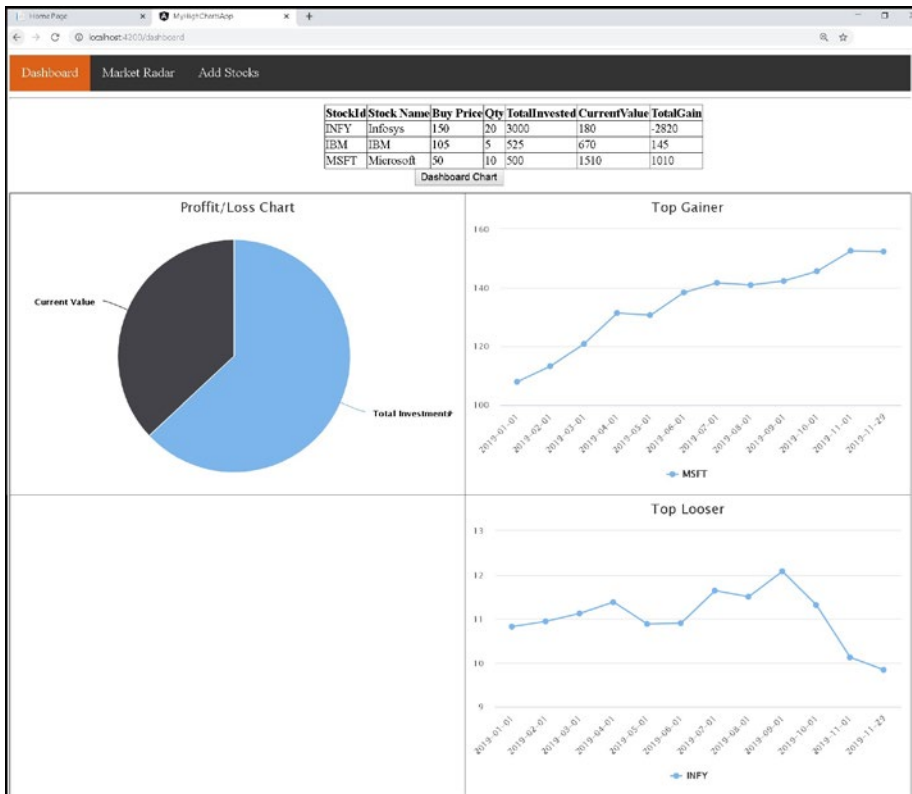
Now click the Market Radar menu. You will get the output shown in Figure 7-24.



**Figure 7-24.** Market radar for selected stock



Now it's time to see the dashboard. On the panel is one table grid, one pie chart, and two line charts (Figure 7-25).



**Figure 7-25.** Dashboard screen

So you saw how easily you can develop a complete application and build an interactive dashboard using Angular and Highcharts with a web API.

## Summary

In this chapter, you developed an e-portfolio learning application with the use of a web API, Angular, and Highcharts. In this chapter, you also learned how to create routing and forms modules so your controls can communicate with components to service a user very efficiently.

You also learn how to consume the Yahoo Finance API from the NuGet package so you can get historical stock data.

With the use of Angular and Highcharts, you can develop your charts more interactive and quickly.

You saw in each chapter, step by step, how easily you can build a dashboard with the use of Angular and Highcharts. I hope you enjoyed this journey with me. Thanks for reading and for your support.

Happy programming!

# Index

## A

AddStock(), 258

Alignment, 21–22

Angular

CLI configuration

creation, 34

generation, 34

installation, 32

structure, 35

configuration

code editor, 31, 32

Node.js, setting up, 28–30

definition, 27

reusable code, 27

two-way model data binding, 27

validations, routing, and

binding, 27

web apps, 28

Angular-Highcharts UI application

app.component.html, 135

app.component.ts, 130–133

app.module.ts, 129, 130

asynchronous, 128

Configure() method, 137

ConfigureService

(IServiceCollection

service) method, 136

CORS, 136

Get method, 128

getApiResponse(url), 133, 135

injectable decorator, 126

lambda expression, 137

MarksModel[] array, 128

marks-model.ts, 127

myFirstAngularHighchart

application, 125

ngOnInit() code, 135

ngOnInit() function, 133

observable, 128

real-time line chart, 139

RxJs, 128

Startup.cs, 138, 139

studentModel, 133

studentNames and

studentMarks array type

variables, 135

StudentService, 125

studentserviceService into

constructor, 133

studentservice.service.ts,

126, 127

troubleshoot CORS

issues, 136

Angular IDE, 31

Application programming

interface (API), 101

## INDEX

### Area charts

- app.component.ts, [66–71](#)
- area-spline chart, [71](#)
- display quantitative data, [65](#)
- Highcharts, [67](#)
- negative values, [67, 69](#)
- plotBands property, [71](#)

### Area spline chart, [69, 71, 72](#)

### Attribute routing

- ActionResult, [112](#)
- ApiController, [112](#)
- API/controller name, [110](#)
- ControllerBase, [112](#)
- empty API controller, add, [114](#)
- methods, web API, [112](#)
- StudentController.cs, [114](#)
- ValuesController.cs, [110, 112](#)
- web API controller, creation, [113](#)

## B

### Bar charts, [11, 17](#)

### Bell curve chart, [225–227](#)

### Bluefish, [32](#)

## C

### Charting, [3–5](#)

### Chart margins, [22](#)

### Chart Type SeriesData classes, [173](#)

### chart.zoomType property, [190](#)

### CheckBoxClick() event, [155–160](#)

### Code editor, [31, 32](#)

### Column bar charts, [17](#)

### Column pyramid charts

- app.component.ts, [87–91](#)
- colorByPoint, [89](#)
- dependencies, [86](#)
- Highcharts, [89](#)
- plotOptions, [91](#)
- stacked, [92](#)

### Combinations

- app.component.html, [185](#)
- app.component.ts, [181–184, 186, 187, 189, 190](#)
- column chart, [181](#)
- column/spline chart, [185](#)
- column/spline/pie chart, [190](#)
- type property, [182](#)

### Command-line interface (CLI), [32](#)

### Configure() method, [137](#)

### Controller, [108](#)

### Credit property, [25](#)

### Cross-origin resource sharing

- (CORS), [136, 243, 246, 247](#)

### Cylinder chart, [198–201](#)

## D

### Dashboard chart, [282](#)

### Database first approach, entity framework

- add new item screen, [250](#)

### AddStock(), [258](#)

### API controller, [257](#)

### API installation, NuGet

- package, [256](#)

### connection properties screen, [254](#)

- database objects and settings
    - screen, 255
  - data connection screen, 253
  - entity data model wizard
    - screen, 252
  - GetActiveStock(), 259
  - GetDashboardPortfolio
    - Data(), 259
  - GetStock(), 259
  - GetStockData(), 259
  - StockController.cs, 259–264
  - StockModel.cs, 256
  - version screen, choosing, 255
  - web API controller, adding, 258
  - Dependency injection, 108, 109, 128, 133
  - Dial chart, *see* Gauge charts
  - Donut chart, 54–56
  - Drilldown charts
    - angular versions, 63
    - app.component.ts, 59–61
    - coding, 62
    - creating series, 58
    - dependencies, 57
    - detailed information, 57
    - details, getting, 58
    - feature, 61
    - pie with drilldown detailed
      - effect, 63
    - unique name, 61
    - with pie feature, 62
  - Drilldown event
    - app.component.html, 150
    - app.component.ts, 145, 147, 148
  - bar chart, 151
  - detail screen, 151
  - Drilldown dependency, 148
  - drilldownsmethod(), 150
  - events method, 149, 150
  - name property, 149, 150
- ## E
- Entity framework, 104
    - ADO.NET, 119
    - GetStudents(), 121
    - launchSettings.json, 122, 123
    - Scaffold-DbContext
      - command, 119
    - StudentController.cs, 120
    - StudentDbContext.cs, 120
    - StudentMarks.cs, 120
    - trusted connection, 119
    - web API, running, 124
  - E-portfolio learning application
    - angular routing
      - addstock.component.html, 274, 276
      - addstock.component.ts, 272
    - add stock portfolio
      - screen, 292
    - app.component.css, 268
    - app.component.html, 269
    - app.component.ts, 268
    - app.module.ts, 266, 267
    - class property, 270
    - dashboard.component.html, 289

E-portfolio learning application (*cont.*)

- dashboard.component.ts
  - code, 282
- dashboard screen, 293
- .forRoot method array, 266
- GetDashboardPortfolio
  - Data(), 288
- GetProfitLossChart(), 289
- GetTopGainerChart(), 289
- GetTopLoserChart(), 289
- home page with menus, 291
- httpClient, Observable, and
  - providerIn, 272
- import RouterModule, 265
- market radar, 292
- ngOnInit(), 280, 288
- onSubmit() method, 274
- path and component
  - properties, 265
- postApiResponse()
  - method, 274
- stock and market radar
  - features, 282
- stock.component.html, 280
- stock.component.ts, 276, 277, 279
- stockmodel.ts, 270
- stockservice.addStock()
  - method, 274
- stock.service.ts file, 271, 272
- application architecture, 265
- database first approach (*see*
  - Database first approach, entity framework)

## database setting up

- adding new database, 247
- add new table screen, 248
- creation screen, 248
- manual data, add, 250
- StockMaster, 247
- StockMaster.sql, 249
- table script, adding, 249

## features

- add stock, 240
- dashboard, 240
- market radar, 240

## web API, creation

- configuration, project, 242
- Microsoft.AspNet.WebApi.
  - Cors, installation, 245
- NuGetPacakge,
  - installation, 245
- NuGet package,
  - management, 244
- project creation, 241
- WebApiConfig.cs, 246

## Events

- app.component.ts, 140, 141, 143
- chart series, plotOptions series
  - click event, 143
- CheckBoxClick(), 155–160
- drilldown (*see* Drilldown event)
- LegendItem click, 152–155
- ngOnInit() method, 144
- plot options series, 144
- Export and print charts, 214–216

**F**

Flash-based animated graphics  
charts, 3

.forRoot method array, 266

Funnel 3D chart type, 201–205

**G**

Gantt chart, 234–237

Gauge charts

aircraft pilots, 92

app.component.ts, 92–94, 96

endAngle, 97

Highcharts and angular, 99

pane section, 96

startAngle, 97

Gauge series chart, 166–171

GaugeSeriesData class, 166

Get method, 128, 133

GetActiveStock(), 259

getApiResponse(url)

method, 133, 135

GetDashboardPortfolio

Data(), 259, 288

GetProfitLossChart(), 289

GetStock(), 259

GetstockByTicks() service

method, 280

GetStockData(), 259

GetStudents(), 121

GetTopGainerChart(), 289

GetTopLoserChart(), 289

Graph, 4, 5

gridLineColor, 176, 177

gridLineDashStyle, 176, 177

gridLineWidth, 176, 177

**H**

Heat map series charts, 79–82

Highcharts

alignment, 21

bar charts, 17

benefits, 2

chart margins, setting up, 22

creation, 8–14

credit property, 25

export and print features,

214–216

fast rendering, 1

history, 3

JavaScript-based library, 1

legend alignment, 22, 23

licenses, 2

line charts, 18

map charts, 19

plot lines, setting, 23, 24

scatter plot, 18, 19

setting layouts, 20

setup and configuration, 6–8

SVG-based line chart

presentation, 16

themes (*see* Themes)

Highcharts wrapper for .NET

Gauge Series chart

HomeController.cs, 166

index.cshtml, 167–170

window.setTimeout, 171

## INDEX

Highcharts wrapper for .NET (*cont.*)

- LineSeries chart
  - demo with.NET
    - Framework, 166
  - HomeController.cs, 162
  - index.cshtml, 163–165
  - installation, 161

Histogram charts, 76–79

HttpClientModule, 128, 129

HttpDelete, 113

HttpGet, 112

HttpPost, 113

HttpPut, 113

## I, J, K

Image setting in chart area

- adding image, 193
- app.component.ts, 191–193
- renderer.image method, 191
- render.events method, 191

## L

legendItemClick(), 152

LegendItem click event, 152–155

Legends, 5, 22, 23

Line charts, 18, 63–65

lineColor, 178

Line plot, 18, 63

LineSeries chart, Highcharts

- wrapper, 160–166

LineSeriesData class, 163, 173

lineWidth, 178

## M

Map charts, 19

Matrix, 79, 82

Microsoft Visual Studio IDE, 31

## N

ngOnInit() method, 133, 144,  
280, 288

Node.js, 28–30

## O

Object-relational mapping (ORM)

- framework, 119

onSubmit() method, 274, 280

Organization chart, 227–230

## P, Q

Pareto chart, 17, 219–224

Pie charts

- allowPointSelect, 51, 52
- angular and highcharts, 48
- app.component.html, 50
- app.component.ts, 48–51
- creation, 48
- dataLables property, 52
- description, 48
- donut feature, 56
- legends, 53
- plotOptions, 51
- sliced and slicedOffset
  - properties, 53, 54



Pie 3D chart, 209–214  
 plotBands property, 71, 98  
 Plot lines, 23, 24  
 postApiResponse() method, 274  
 Pyramid 3D chart, 205–209

## R

Radar chart, 216–219  
 Reactive Extensions for JavaScript  
 (RxJs), 128  
 Representational states  
   transfer (REST)  
   resource/method, 103  
   response, 103  
   web API development, VS  
     creation, 104  
     project configuration, 105  
     project template selection  
     screen, 106  
 Resources, 101  
 Routing  
   attribute (*see* Attribute routing)  
   database creation  
     database screen, 116  
     record process into table, 118  
     records into table, insert, 119  
     SQL Server Object Explorer, 116  
     table through code, 117

## S

Scaffold-DbContext, 120  
 Scalable vector graphics (SVG), 15, 16

Scatter charts, 18, 72–76  
 Scatter graph, 72  
 Scatter plot, 18, 19, 72  
 SeriesData Classes, 172–173  
 services.AddCors() method, 137  
 Solution explorer  
   Configure(), 109  
   ConfigureService(), 108, 109  
   files and folders  
     appSettings.json, 108  
     controller, 108  
     dependencies, 107  
     Program.cs, 108  
     properties, 108  
     Startup.cs, 108  
   Visual Studio, 106  
 Speedometer chart, *see* Gauge  
   charts  
 SQL Server database, 104  
 Stacked bar charts, 82–86  
 Stacked column pyramid  
   chart, 89, 92  
 StockController, 258  
 StockDb, 247  
 StockDDL(), 280  
 stockservice.addStock() method, 274  
 StudentService.service.ts  
   class, 126, 127

## T, U, V

Themes  
   alternateGridColor, 175  
   app.component.ts, 176, 177

## INDEX

### Themes (*cont.*)

- axis, [175](#)
- colors and graphical representations, [175](#)
- combinations (*see* [Combinations](#))
- dash style series to line chart
  - app.component.ts, [178](#), [180](#)
- gridLineDashStyle,
  - gridLineWidth, and
  - gridLineColor, [177](#)
- major tick-related properties, [178](#)
- tickColor, [177](#)
- tickLength, [177](#)
- tickPosition, [177](#)
- tickWidth, [177](#)

### 3D charts

- app.component.ts, [194–197](#)
- column type chart, [198](#)
- cylinder chart, [198–201](#)
- funnel 3D, [201–205](#)
- pie chart, [209–214](#)
- properties, [195](#)
- pyramid chart, [205–209](#)
- required dependencies, [193](#)

### 3D donut chart, [214](#)

### Timeline chart, [233–235](#)

### Trusted\_Connection, [120](#)

### TypeScript

- Highcharts angular wrapper
  - Angular app, area chart, [45](#)

### AppComponent, [41](#)

### app.component.html, [41](#)

### app.components.ts,

[39](#), [40](#), [43](#), [45](#)

### app.module.ts, [37](#)

### code line, [38](#), [39](#)

### @Component

decorator, [41](#)

### highcharts and

chartOptions, [42](#)

### index.html, [42](#)

### installation, [36](#)

### metadata properties, [41](#)

### ng serve command, [45](#)

### running, ng serve, [43](#)

### open-source programming

language, [36](#)

## W, X, Y

### Web API

framework, [102](#)

HTTP verbs, [101](#)

request-response model, [102](#)

verbs, [101](#)

### WebStorm, [32](#)

## Z

Zoom option, [190](#)