workshop

Angular

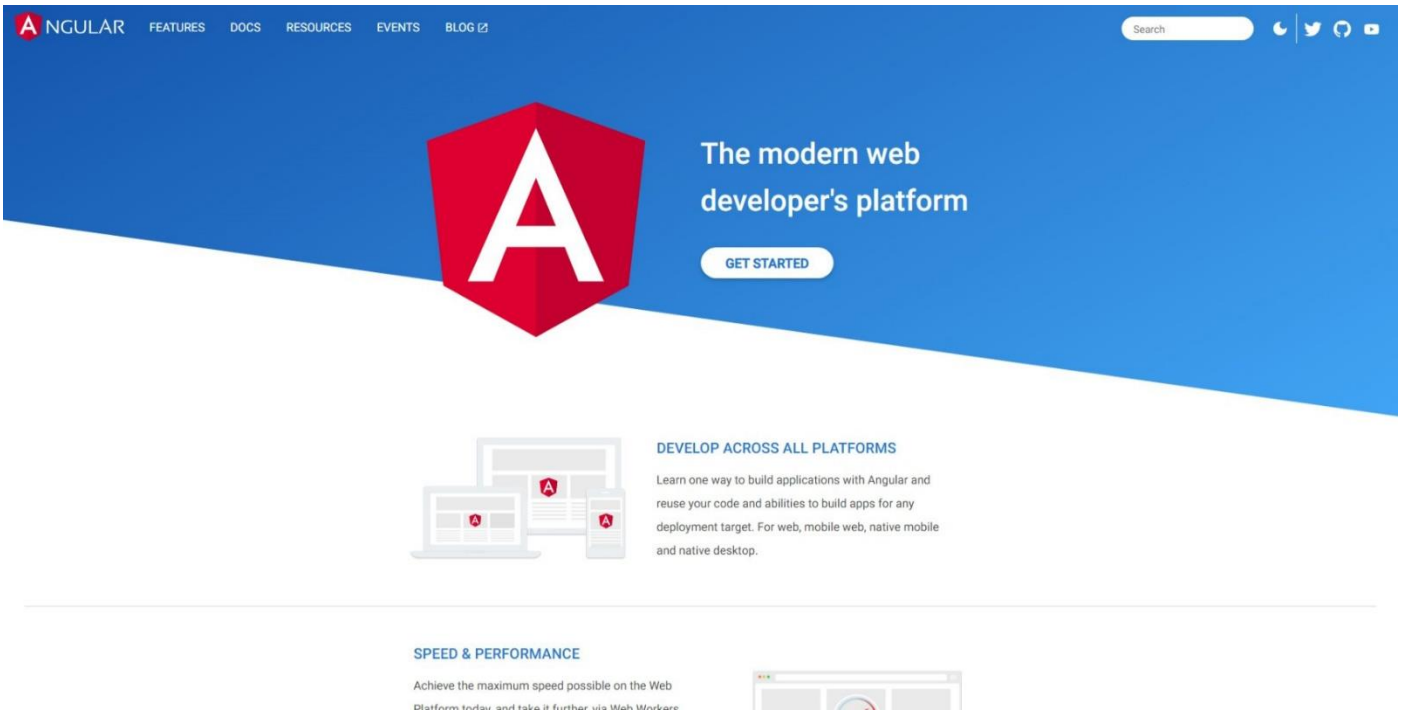
Contents

Contents

Introduction

What is Angular?

Angular is a modern web development platform created by **Google** that allows you to build Applications for Web, Mobile Web, Native Mobile and Native Desktop.



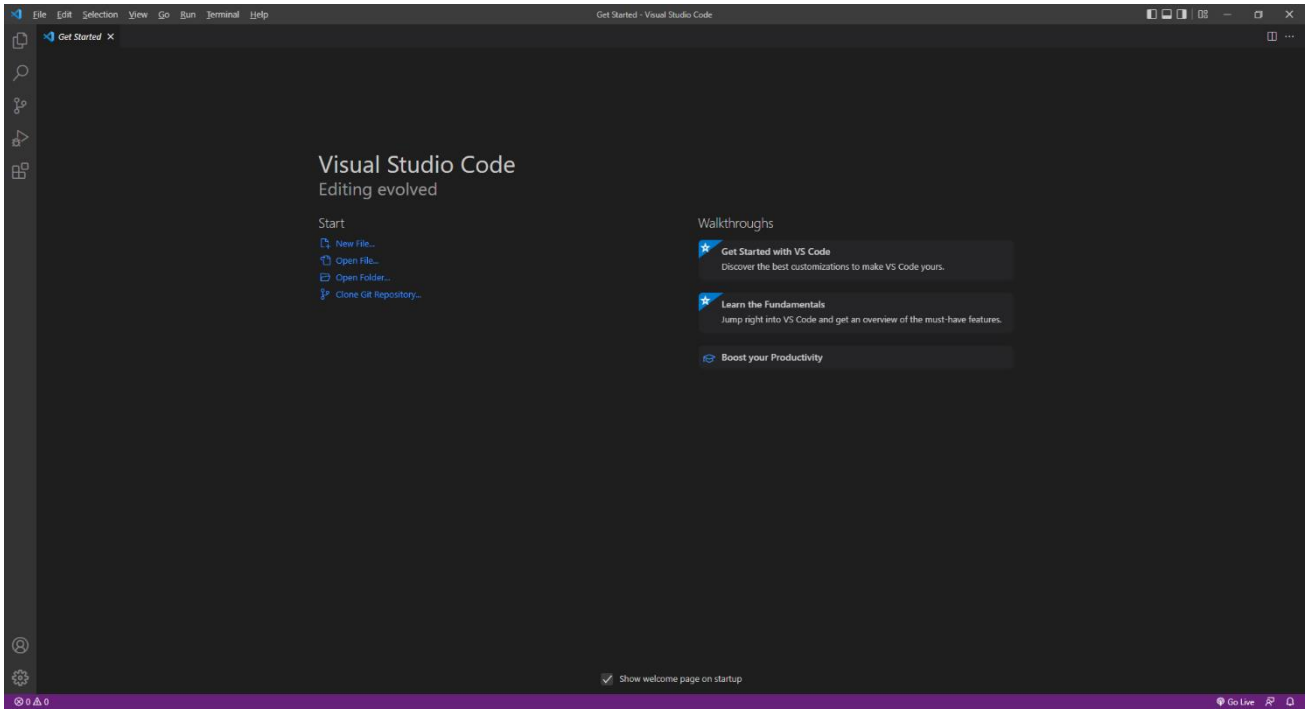
Angular uses **TypeScript**, which is a strongly typed programming language that builds upon **JavaScript**, which is a core programming language and development platform used on the web. **JavaScript** allows the web to be more interactive and offer dynamic content with **TypeScript** offering better tooling and support at any scale of project, you can find out more about **TypeScript** at typescriptlang.org.

Angular requires **Node.js** which is a **JavaScript** runtime that allows Applications to be used such as **Angular** you can find out more about **Node.js** at nodejs.org. Angular along with any applications created also depend on **npm** packages, **npm** is a software registry for applications, you can find out more about **npm** at docs.npmjs.com.

Angular allows you to build features quickly with simple and declarative templates to create your own **Components** that make up Angular or even use a variety of existing ones. **Components** are the building blocks of applications and they contain **CSS** to define how the component will look and **HTML** which will control how the component will be displayed or rendered with **Templates**. **Angular** is referred to as being **Opinionated**, which means it provides a way of doing things that everyone follows. **Angular** supports a **Command Line Interface or CLI**, which is simplest and also recommended way of creating applications, it supports the ability to compile, serve, generate files and more for an Application and you can find out more about **Angular** including documentation, examples and more at angular.io.

What is Visual Studio Code?

Visual Studio Code will help create **Angular** applications even more easily, it is a free **Integrated Development Environment or IDE** created by **Microsoft**.

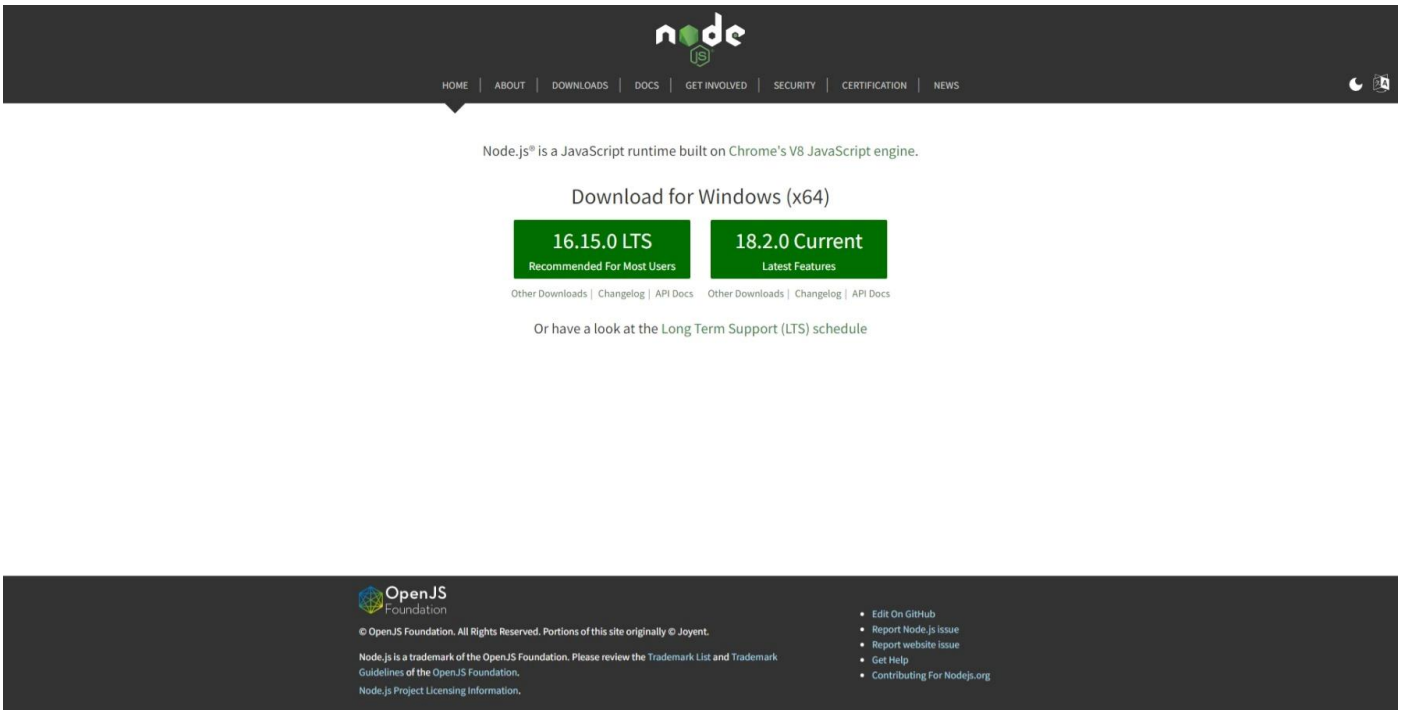


Visual Studio Code supports syntax highlighting which will add colours to certain parts of the text and make it easy to make sure everything is being entered correctly when writing **Angular** Applications. You can also use **Visual Studio Code** to edit any other **TypeScript**, **CSS**, **HTML** and more making more than just creating **Angular** applications more straightforward. If you want to find out more about **Visual Studio Code** along with documentation, extensions and more you can visit code.visualstudio.com.

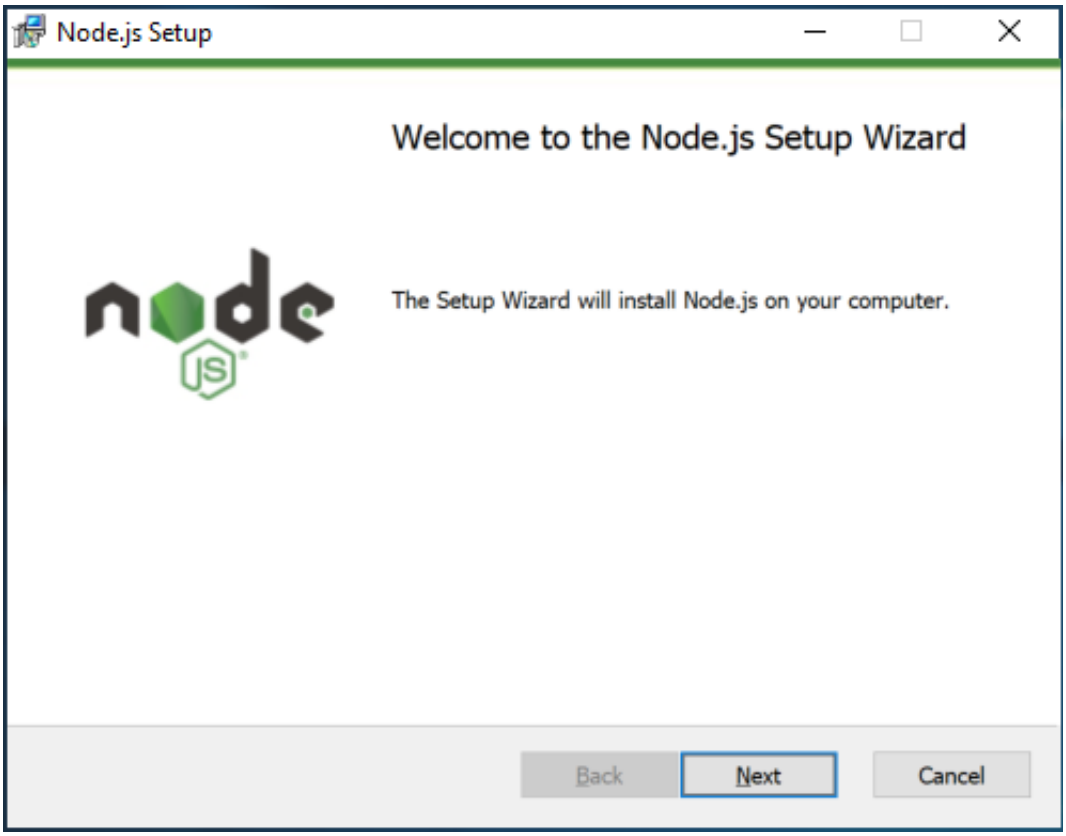
Setup and Start

Angular

Angular requires an **Active** or **Long Term Support / LTS** version of **Node.js** which if you don't have it already, you can **Download** the **LTS** version for your Platform such as **Windows** from nodejs.org.



Once **Downloaded,** you can then **Install** it by following the steps in the **Installation Wizard**



Once **Node.js** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Command Prompt** and then select it.

Once in the **Command Prompt** you can use **mkdir** to **Create** a new **Folder**, then **cd** to **Change Directory** to this new **Folder** as follows:

```
mkdir Angular
cd Angular
```

Then you can to **Install** the **Angular CLI (Command Line Interface)** by typing in the following command using **npm** which comes with **Node.js** and then press **Enter**:

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

Once the **Angular CLI** has been **Installed**, while still in the **Command Prompt** you can then create a New **Angular Workspace** by typing in the following command and then press **Enter**:

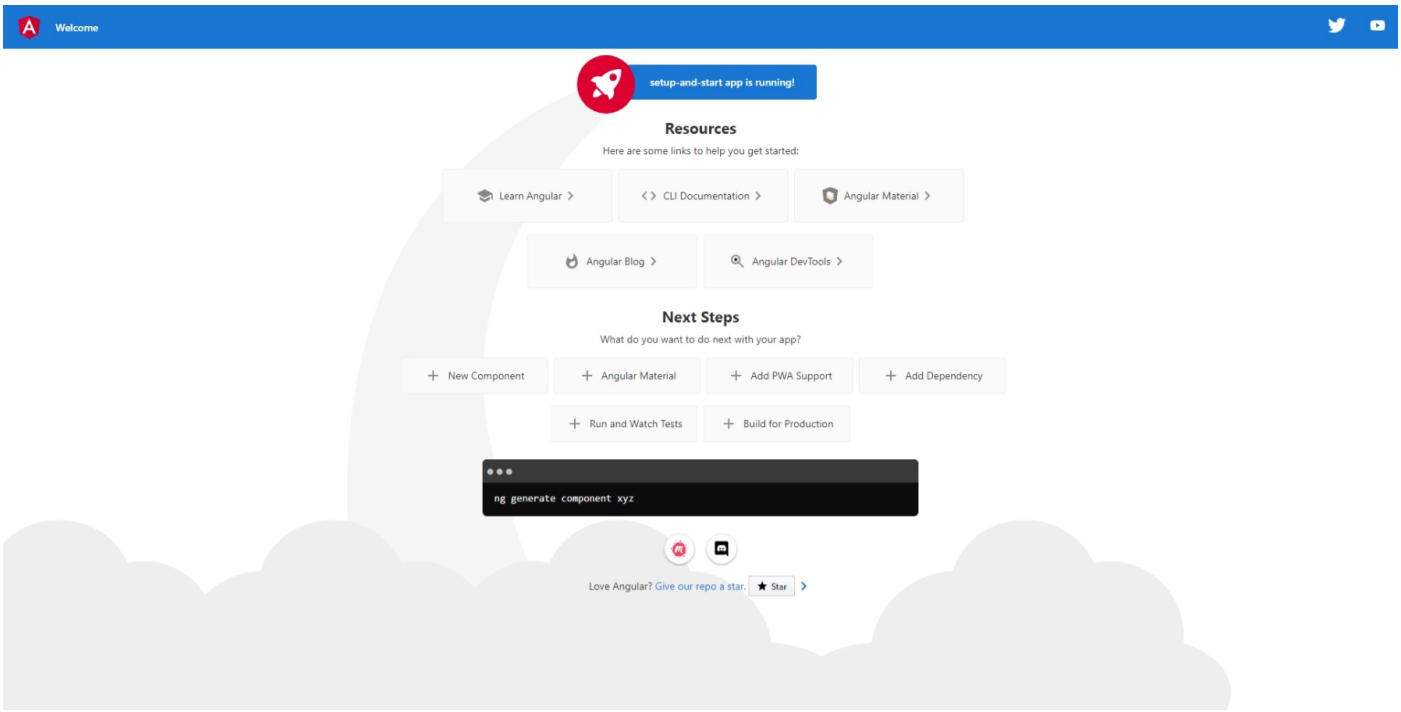
```
ng new workshop
```

You will be asked **Would you like to add Angular routing?** and **Which stylesheet format would you like to use?** You can just accept the defaults by pressing **Enter**. After this in the **Command Prompt** you will need to change to the **Folder** for the **Workshop** by typing in the following command and then press **Enter**:

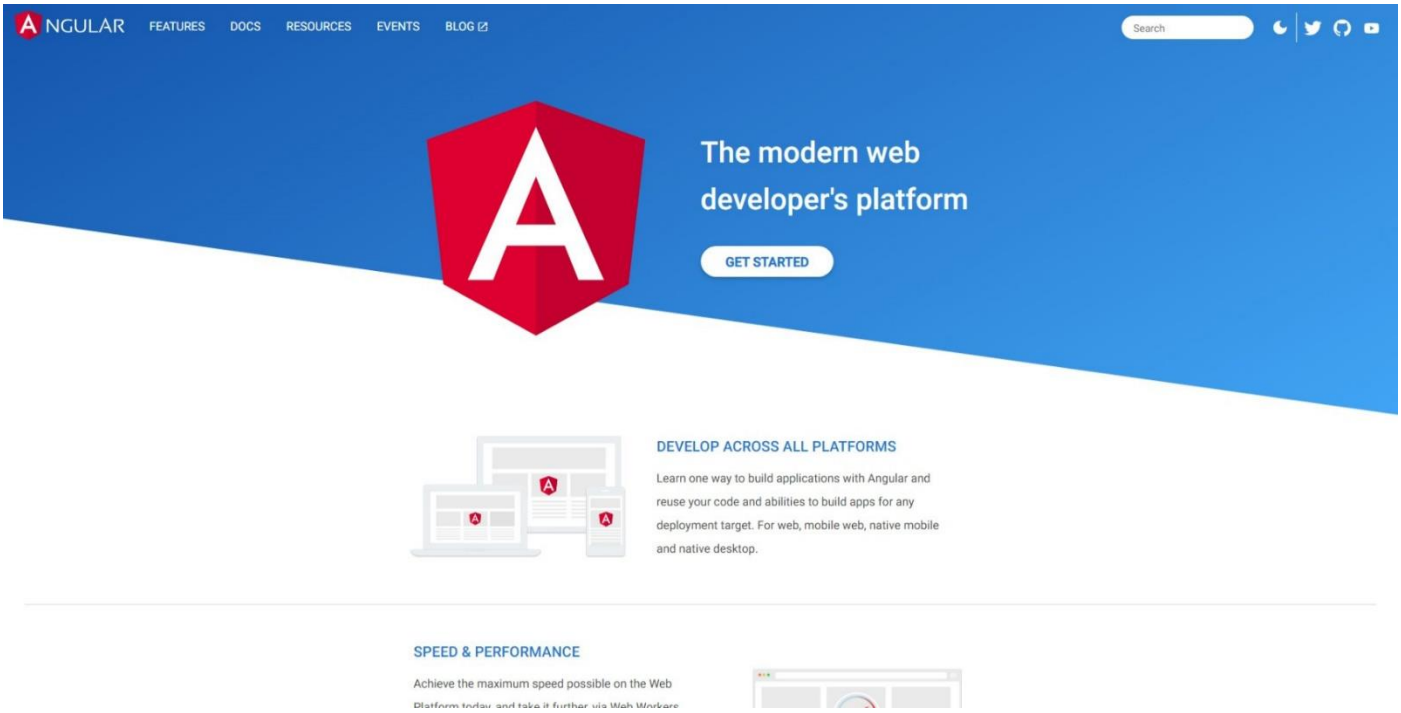
```
cd workshop
```

Once done while still in the **Command Prompt** you can type in the following command followed by **Enter** which will **Build** and **Serve** the Application which will also display it in your **Browser** you need to keep the **Command Prompt** open but you won't need to do anything else using the **Command Prompt**.

```
ng serve --open
```



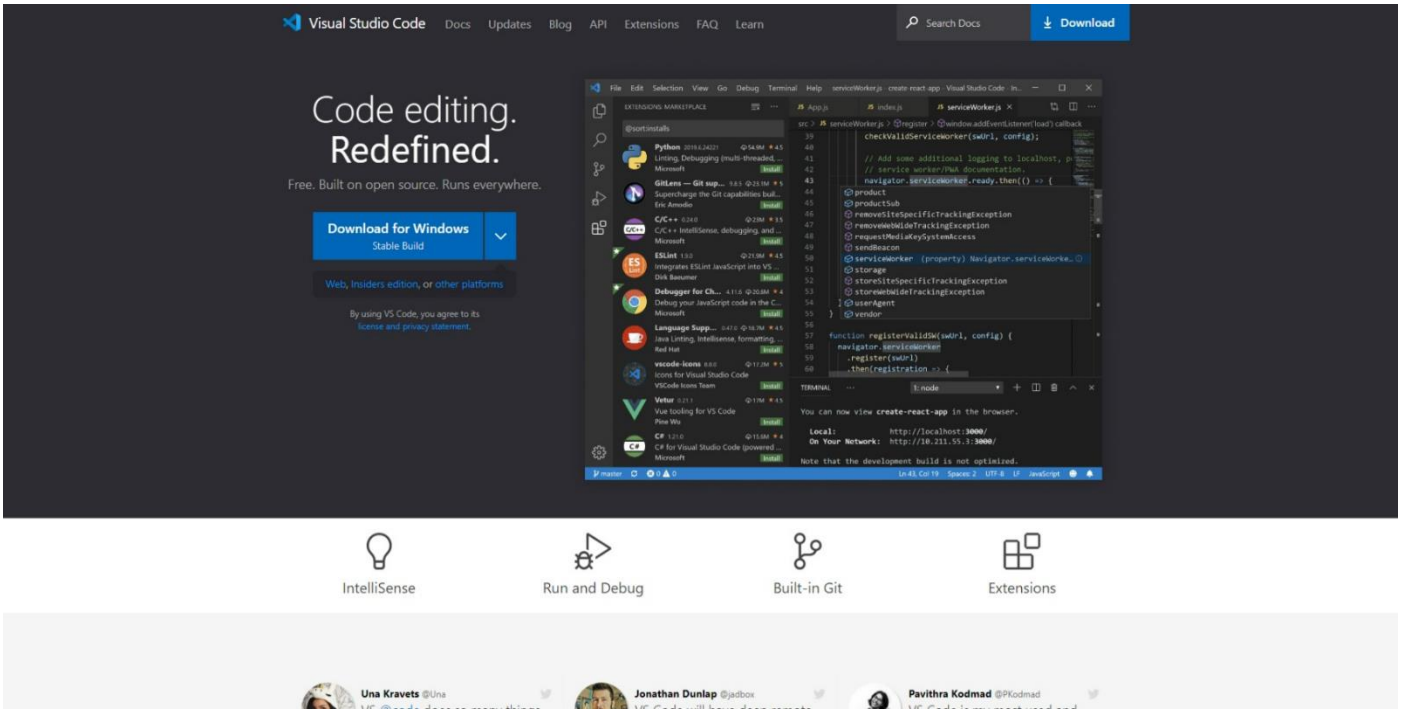
Should you need to you can get information, documentation and more about **Angular** at angular.io



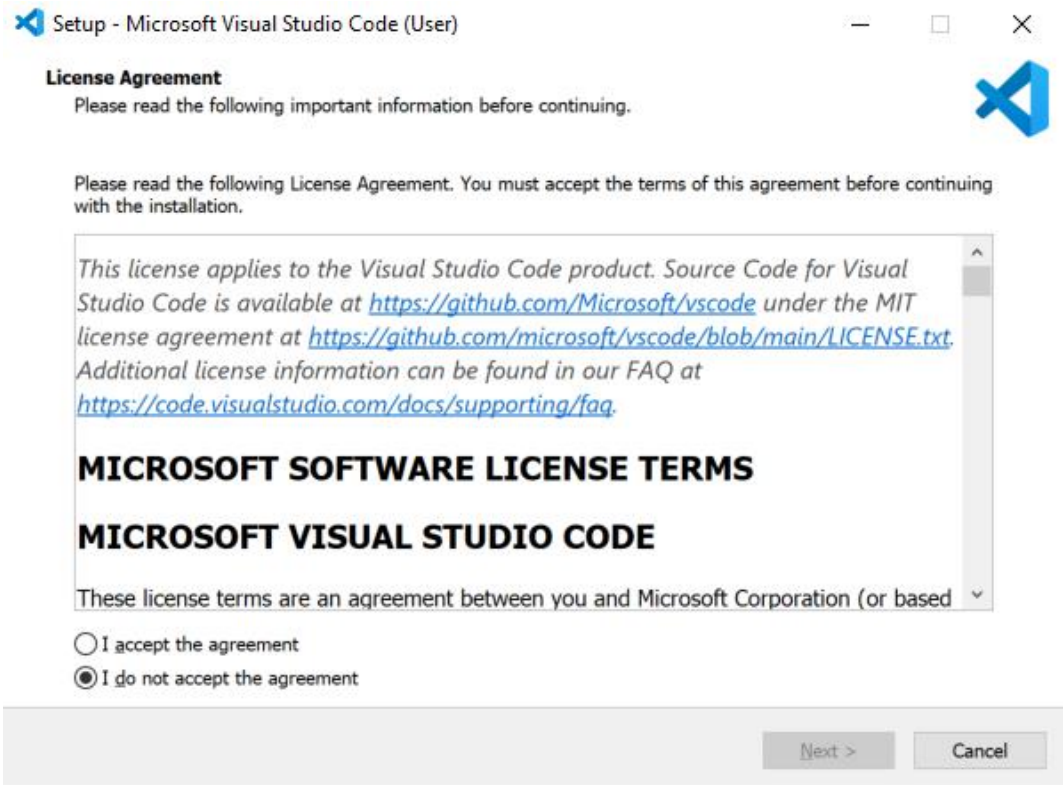
This **Workshop** supports at least **Version** *11* of **Angular** with **Version** *14* being used throughout.

Visual Studio Code

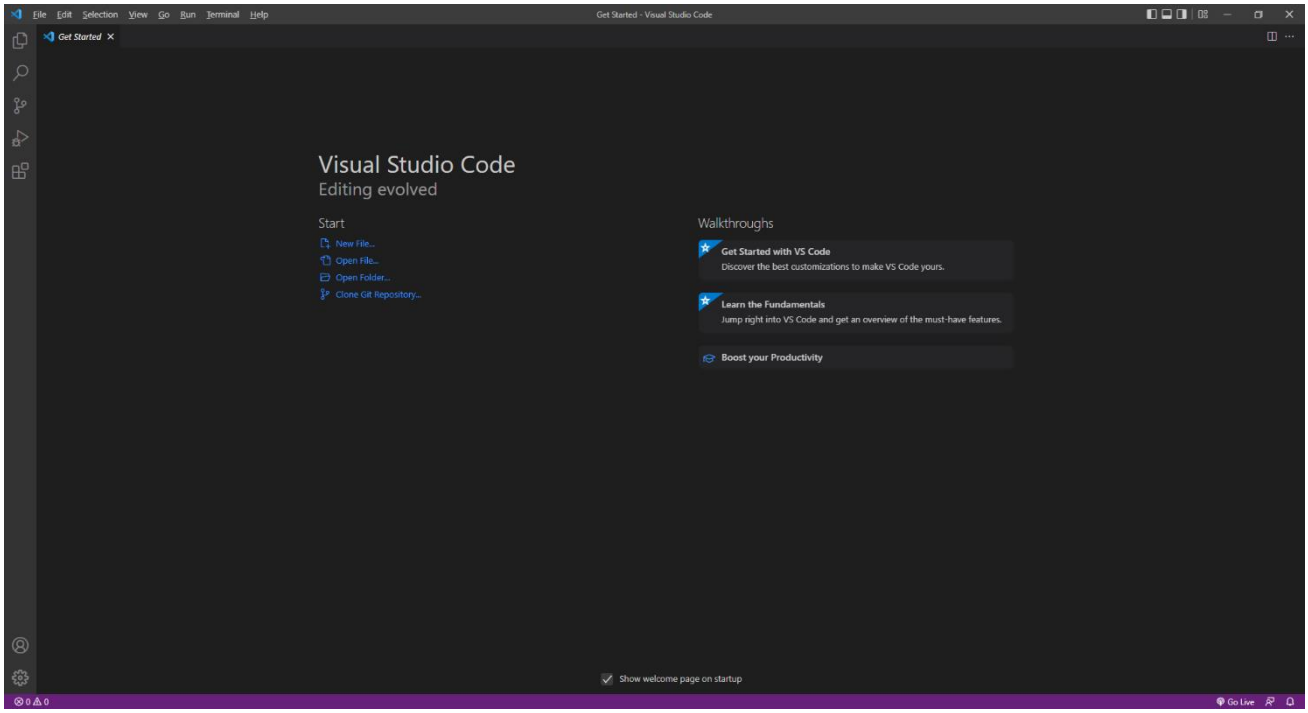
To be able to **Edit** your Application you will need to **Download**, if you don't have it already, **Visual Studio Code** for your Platform such as **Windows** from code.visualstudio.com.



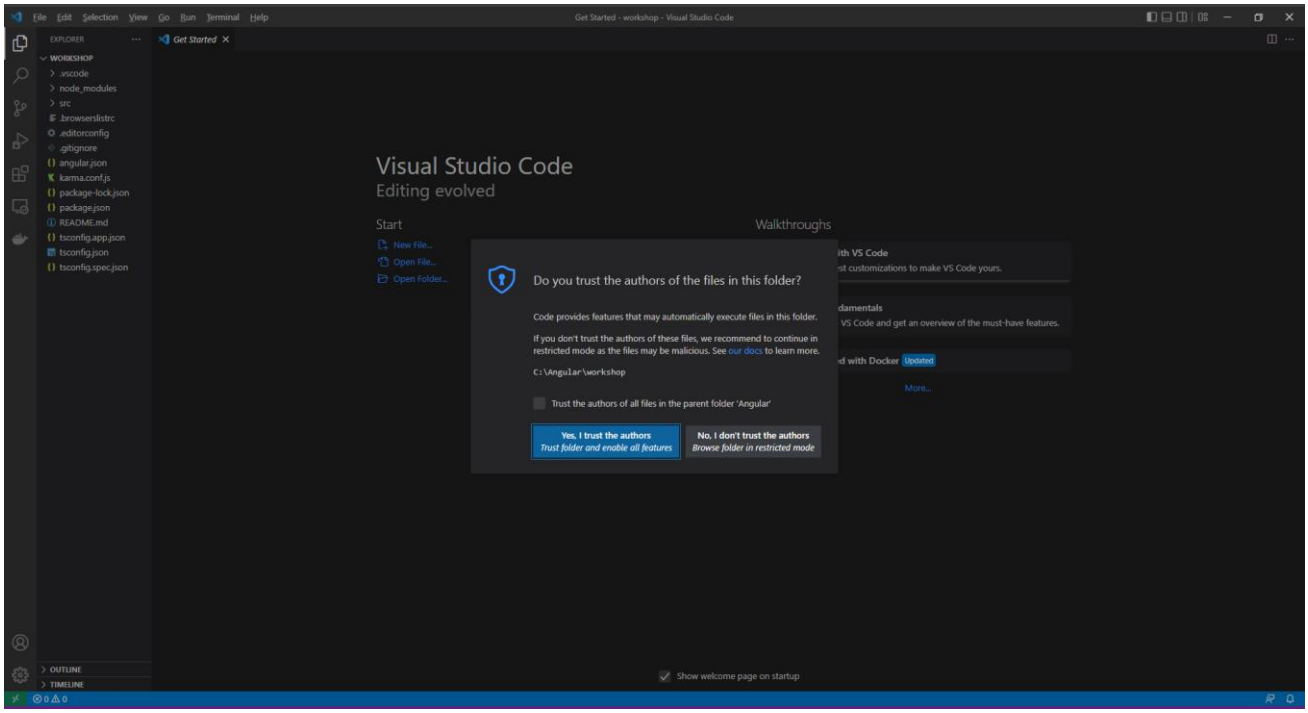
Once **Downloaded,** you can then **Install** it by following the steps in the **Installation Wizard**



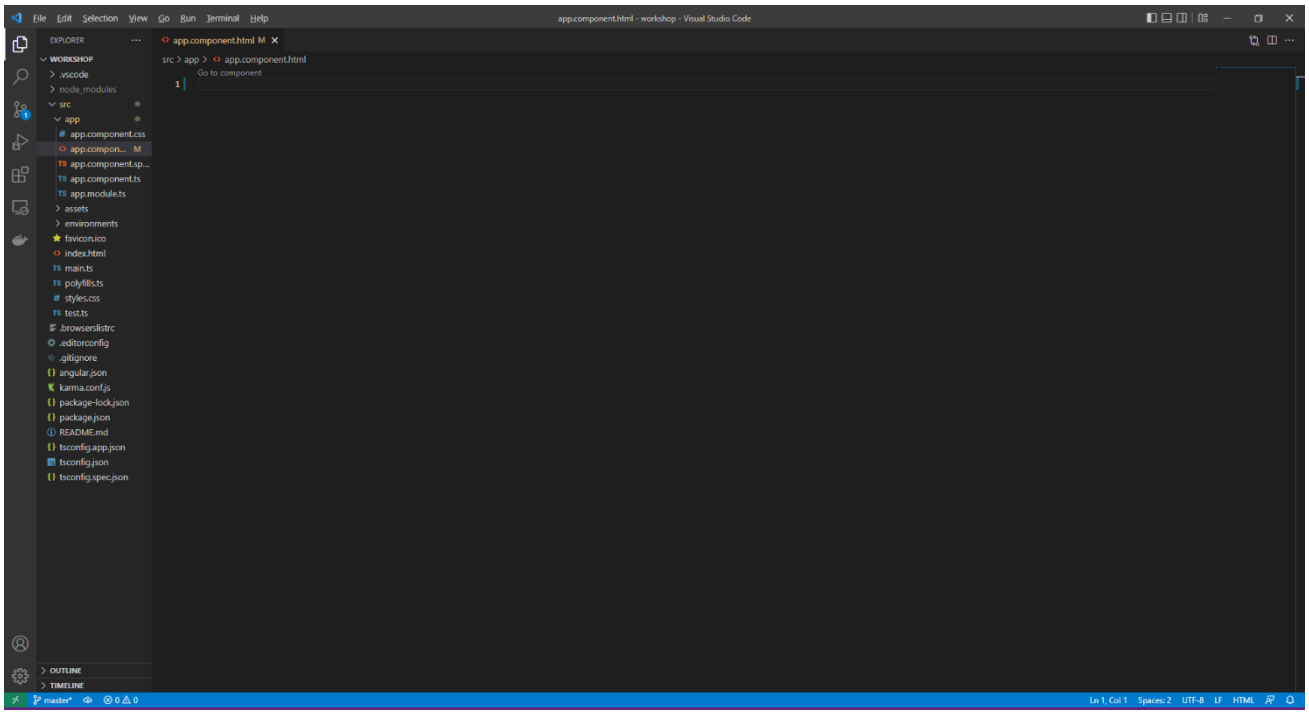
Once **Visual Studio Code** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Visual Studio Code** and then select it.



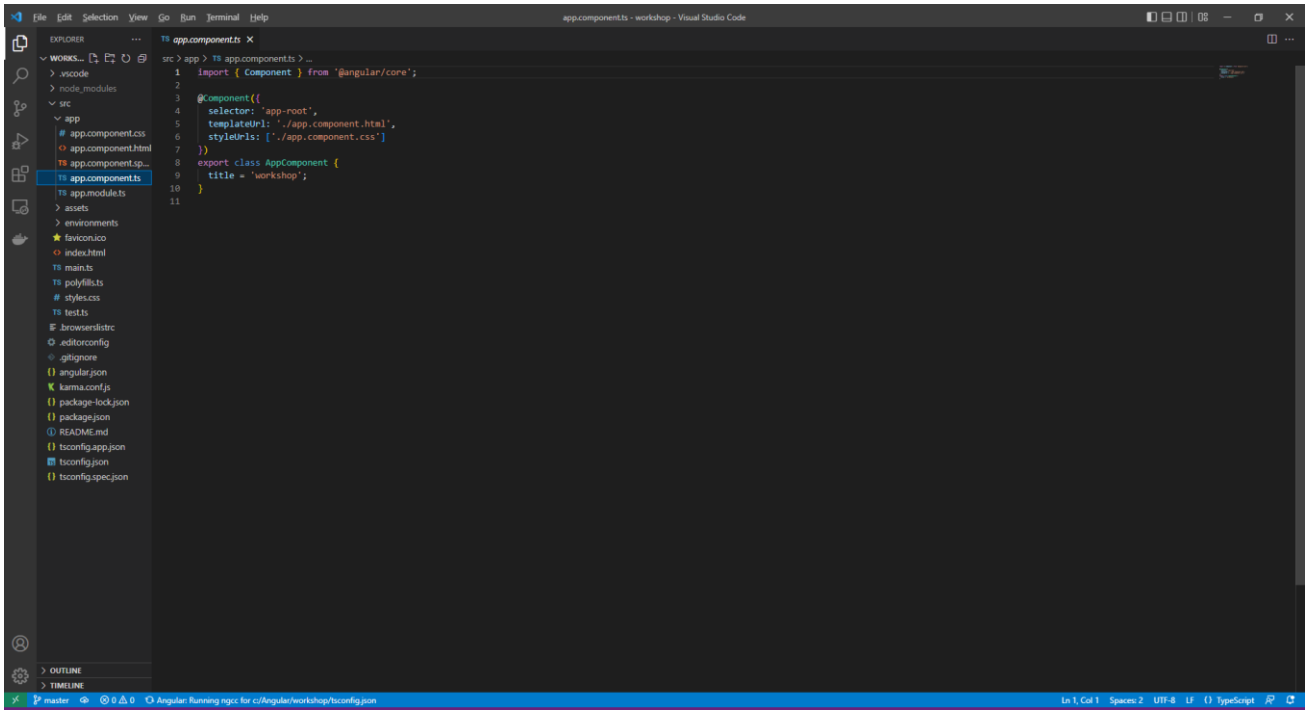
Once **Visual Studio Code** has opened from the **Menu** choose **File** then **Open Folder...** then select the **Folder** for your Application e.g. *C:\Angular\workshop*. Then once the **Folder** has been opened Select the **Yes, I trust the authors** option in the **Do you trust the authors of the files in this folder?** if this is displayed.



Within **Visual Studio Code** will be the **Explorer** you can then Expand the **Folder** for **src** and then **app** to find *app.component.css* file which defines any **CSS** Styles for the **Template**, the **Template** itself is *app.component.html*, you should Clear the contents of this file so that it is blank as follows:



You will also find the main **Component** for the Application which is *app.component.ts* which is where you will be spending most of your time in the **Workshop**.



You will also find *app.component.spec.ts* and see other files like these but they won't be used in the **Workshop** but they are used when **Testing** an **Angular** Application.

Within the **Component** of *app.component.ts* below **`title = 'workshop';`** you should type in the following **Comment**:

```
// Variables
```

During the **Workshop** when you need to add or declare a **Variable** for *app.component.ts* then these should be placed on their own line below this **Comment** for each part of the **Workshop** as you can use the same **Angular Workspace** for the entire **Workshop**, you do not need to remove anything unless explicitly told to do so. Then add a Blank line by pressing **Enter** and then type in the following **Comment** followed by another Blank line:

```
// Methods
```

During the **Workshop** when you need to add a **Method** for *app.component.ts* then these should be placed below this **Comment** for each part of the **Workshop** when needed, as again you can use the same **Angular Workspace** part of the **Workshop**.

Once the **Comments** have been added to *app.component.ts* it should look something like the following and once done you are ready to continue the **Workshop**:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'workshop';
  // Variables

  // Methods

}
```

Components

Components in **Angular** are the key building blocks for Applications, they are comprised of a **Template** in **HTML** that declares what will be rendered on the Page, there is then a **Class** in **TypeScript** that will define any Behaviour, there is then a **CSS** selector to define how the component is used in a **Template** and also optionally any **CSS** Styles applied to the **Template**.

After following **Setup and Start** you can create a **Component**, you can do so on **Windows** using the **Command Prompt** and the **Angular CLI**, by going to **Start** then search for **Command Prompt** and then select it, this should launch another **Command Prompt** with the other being used to **Build** and **Serve** your Application. With the new **Command Prompt** open use **cd** to change to the **Folder** for the **Workshop** Application e.g. *C:\Angular\workshop* by typing the following and then press **Enter**:

```
cd C:\Angular\workshop
```

If another **Command Prompt** is not open then you can launch one then type in the above **`cd C:\Angular\workshop`** command and press **Enter**, then the following command and then press **Enter**:

```
ng serve --open
```

In the **Command Prompt** that is not being used to **Build** and **Serve** your Application, you can create a **Component** using the **Angular CLI** by typing the following, and then press **Enter**:

```
ng generate component hello-world
```

You can close this **Command Prompt** once done. Select **Visual Studio Code**, if not started you can in **Windows** go to **Start** then find **Visual Studio Code** select it and once opened from the **Menu** choose **File** then **Open Folder...** then select the **Folder** for your Application e.g. *C:\Angular\workshop*.

Within **Visual Studio Code** from the **Explorer** there will be a new **Folder** in **app** under **src** of *hello-world* which will contain the *hello-world.component.css* for the **CSS**, *hello-world.component.html* for the **HTML** of the **Template** and the **Component** itself of *hello-world.component.ts* and a **Module** of *app.module.ts*.

After following **Setup and Start** and **Components**, within **Visual Studio Code** select the **Component** of *hello-world.component.ts* in **Explorer** within **Visual Studio Code** which will be found be within the **Folder** of **src**, then **app** and then **hello-world**.

Then in the **Component** of *hello-world.component.ts* above the line **`constructor() { }`** type in the following:

```
message: string = 'Hello World';
```

This is a **Variable** for **`message`** which contains the **`string`**, which defines some text, using single quotes or **`'`** around it with the **Value** of **`Hello World`**.

Templates

Templates in **Angular** support all standard **HTML** features along with specific syntax supported by **Angular** that you can use in **Templates**.

Interpolation

Interpolation allows you to include Dynamic values in **Templates** using double curly-braces such as **{{** and **}}**.

After following **Setup and Start** and **Components**, within **Visual Studio Code** select the **Template** of *hello-world.component.html* in **Explorer** within **Visual Studio Code** which should be within the **Folder** of **src**, then **app** and **hello-world**. Then in **Template** for *hello-world.component.html* remove ** hello-world works! ** and then type in the following:

```
<h1>{{message}}</h1>
```

This will be used in the **Template** to display the contents of the **Variable** of **message** for the **Component** using **HTML** within a **h1** Tag.

Within **Visual Studio Code** select the **Template** for the Application of *app.component.html* which can be located within the **Folder** of **src** then **app** from **Explorer** and type in the following:

```
<app-hello-world></app-hello-world>
```

You can then select the **Browser** that was opened and you should see the text *Hello World* displayed.

You can also use **Expressions** within a **Template** that you can use to invoke **Methods**. In **Visual Studio Code** select the **Component** of *app.component.ts* from the **Explore**, then below the **Comment** of **// Methods** type in the following **Method**:

```
getMessage() {
  let message: string = 'Hello Again!';
  return message;
}
```

Then also while still in **Visual Studio Code** select the **Template** of *app.component.html* and below **<app-hello-world></app-hello-world>** type in the following:

```
<h2>{{getMessage()}}</h2>
```

This will display the **Hello Again!** message in a **h2** Tag. You can then select the **Browser** that was opened with **ng serve –open** from the **Command Prompt** and you should see the text *Hello Again!*

Statements and Event Binding

Statements can used in a **Template** to respond to an **Event** which can be done using **Event Binding** which has a target name in brackets **(** and **)** followed by the **Statement**. To display a message using Statements and **Event Binding** return to **Visual Studio Code** then in the **Component** of *app.component.ts* below the **Comment** for **`// Methods`** and below any previous **Method** by typing in the following **Method**:

```
showMessage() {
  let message: string = 'Hello World';
  alert(message);
}
```

Then you can have an **Event** on a **Button** that is triggered when it is **Clicked** in the **Template** of *app.component.html* by typing in below **`<h2>{{getMessage()}}</h2>`** the following:

```
<button type="button" (click)="showMessage()">Show Message</button>
```

You can select the **Browser** and you should see a **`button`** labelled *Show Message* which when **Clicked** will display an **`alert`** with the Message of *Hello World*.

Pipes

Pipes in **Templates** allow for the transformation of data, there are some built in **Pipes** such as **`AsyncPipe`**, **`CurrencyPipe`**, **`DatePipe`**, **`DecimalPipe`**, **`Il18nPluralPipe`**, **`Il8nSelectPipe`**, **`JsonPipe`**, **`KeyValuePipe`**, **`LowerCasePipe`**, **`PercentPipe`**, **`SlicePipe`**, **`TitleCasePipe`** and **`UpperCasePipe`** and are applied using the **Pipe** character of **`|`** .

Return to **Visual Studio Code** and select the **Component** for the Application of *app.component.ts* within the **Folder** of **src** then **app** and below the **Comment** for **`// Variables`** type in the following **Variable**:

```
dateOfBirth: Date = new Date('23-June-1912');
```

Then you can use **Pipe** to Format the output in the **Template** for the Application of *app.component.html* below **`<button type="button" (click)="showMessage()">Show Message</button>`** by typing in the following:

```
<div>{{dateOfBirth | date}}</div>
```

If you switch to the **Browser**, that was opened with the **`ng serve -open`** command, then you can see the **Date** being displayed as *Jun 23, 1912* which is Alan Turing's birthday, a pioneer in the field of computing.

Property Binding

Property Binding allows values for a **Property** of **HTML** elements within **Templates** to be set in one direction from a **Component**. To bind to a **Property** in **HTML** you just need to enclose it in square brackets of **[** and **]**.
Return to **Visual Studio Code** and in the **Component** of *app.component.ts* and below the **Comment** for **`// Variables`** type in the following **Variable** after any previously declared **Variables**:

```
grinningFace: string = 'https://openmoji.org/data/color/svg/1F600.svg';
```

You can use **Property Binding** to display an **Image** by setting the **src** to the value of the **Property** of a **Component** for the **Template** of *app.component.html* by typing below **`<div>{{dateOfBirth | date}}</div>`** the following:

```
<img [src]="grinningFace" height="150" width="150"/>
```

Back in the **Browser** you should see a *Grinning Face* displayed, image courtesy of openmoji.org

Attribute Binding

Attribute Binding can be used when there is not a **Property** of an **Element** to bind to, if there then **Property Binding** should be used instead. Define some **CSS** Styles within the **CSS** of *app.component.css* by typing in the following:

```
.inverted {
  color: white;
  background-color: black;
}

.large {
  font-size: 2.0em;
}
```

These can then be **Bound** from the **Component** of *app.component.ts* using a space-delimited **List** or a set of **Key** / **Value** pairs. You could also **Bind** to an **Array** of **Class** names by typing in the following **Variable** below the **Comment** for **`// Variables`** and after any previously declared **Variables**:

```
contrast: string = ['inverted', 'large'];
```

Then in the **Template** of *app.component.html* below **`<img [src]="grinningFace" height="150" width="150"/>`** type in the following:

```
<div><span [class]="contrast">Contrast</span></div>
```

If you switch back to the **Browser** it will have the Text of *Contrast* in *white* with a *black* Background.

Two-Way Binding

Two-Way Binding allows **Components** to share data by listening to **Events** and updating Values between **Components** to be displayed in a **Template**. **Two-way Binding** combines square brackets **[]** of **Property Binding** with the brackets **()** of **Event Binding** as **[()]**.

To create a **Component** you can do so on **Windows** using the **Command Prompt** and the **Angular CLI**, by going to **Start** then search for **Command Prompt** and then select it, then **cd** to the **Folder** for your Application e.g. *C:\Angular\workshop* with the following command and then press **Enter**:

```
cd C:\Angular\workshop
```

Then while still in the **Command Prompt** you can use the **Angular CLI** to create a new **Component** by typing in the following and then press **Enter**:

```
ng generate component Sizer
```

You can close this **Command Prompt** once done and return to **Visual Studio Code** and from the **Explorer** there will be a new **Folder** in **app** within **src** of *sizer* which will contain the *sizer.component.css* for the **CSS**, *sizer.component.html* for the **HTML** of the **Template** and the **Component** itself of *sizer.component.ts*, within this change the first line of **`import { Component, OnInit } from '@angular/core';`** to the following:

```
import { Component, OnInit, EventEmitter, Input, Output } from '@angular/core';
```

Then while still in the **Component** of *sizer.component.ts* above the line **`constructor() { }`** the **Property** for **`@Input`** and the **Event** for **`@Output`** can be defined by typing in the following:

```
@Input()  size!: number | string;
@Output() sizeChange = new EventEmitter<number>();
```

Then below the line **`constructor() { }`** can define these **Methods** to perform the functionality of the **Component** including **Methods** to **`increase`** and **`decrease`** the Size as follows:

```
resize(delta: number) {
  this.size = Math.min(40, Math.max(8, +this.size + delta));
  this.sizeChange.emit(this.size);
}

decrease() {
  this.resize(-1);
}

increase() {
  this.resize(+1);
}
```

In the **Template** of *size.component.html* remove **`<p>sizer works!</p>`** and then type in the following:

```
<div>
  <button type="button" (click)="decrease()" title="Decrease">-</button>
  <button type="button" (click)="increase()" title="Increase">+</button>
  <span [style.font-size.px]="size">Font Size: {{size}}px</span>
</div>
```

Then in the **Component** for the Application of *app.component.ts* and type in the following **Variable** below the **Comment** for **`// Variables`** and after any previously declared **Variables**:

```
fontSize: number = 20;
```

Then in the **Template** for the Application of *app.component.html* after **`<div><span [class]="contrast">Contrast</span></div>`** type in the following:

```
<app-sizer [(size)]="fontSize"></app-sizer>
<div [style.font-size.px]="fontSize">Resizable Text</div>
```

You can also see how the alternative way of **Property Binding** would work by adding another line below the ones typed above as follows:

```
<app-sizer [size]="fontSize" (sizeChange)="fontSizePx=$event"></app-sizer>
```

In the **Browser** you can use the *Sizer* to change the *font-size* of itself and the *Resizable Text*.

Template Variables

Template Variables allow you to share data between parts of a **Template** using the **Hash** as **`#`** to declare a **Template Variable**. If you want to display some **Input** using an **`Alert`** then you can do this in the **Component** for the **Application** of *app.component.ts* below the **Comment** for **`// Methods`** and below any previous **Methods** by typing in the following **Method**:

```
show(message: string) {
  alert(message);
}
```

Then within the **Template** for the Application of *app.component.html* you can use **Template Variables** when a **`Button`** is clicked to display what was entered in the **`Input`** below **`<app-sizer [size]="fontSize" (sizeChange)="fontSize=$event"></app-sizer>`** and type in the following:

```
<input type="text" #message/>
<button type="button" (click)="show(message.value)">Show</button>
```

In the **Browser** is a **`Button`** that when **Clicked** will show an **`alert`** with anything typed in the **`Input`**.

Directives

Directives in **Angular** add additional Behaviour to **Elements** within an Application.

Attribute Directives

Attribute Directives are **ngClass**, **ngStyle** and **ngModel**. You can use **ngClass** with **Property Binding** to set the **Property** of a **Directive** which adds and removes **CSS** for a **HTML** element.

After following **Setup and Start**, **Components** and **Templates** within **Visual Studio Code** select the **Component** for the Application of app.*component.ts* in **Explorer** within **Visual Studio Code** which should be within the **Folder** of **src**, then **app** and then below the **Comment** for **// Variables** type in the following **Variable** after any previously declared **Variables**:

```
styling: string = 'highlighted';
```

Define a Style within the **CSS** of *app.component.css* by typing in the following after any defined **CSS** Styles:

```
.highlighted {
  background-color:yellow;
}
```

You can then use the **Variable** from the **Component** with **ngClass** within the **Template** for the Application of *app.component.html* by typing in below **<button type="button" (click)="show(message.value)">Show</button>** the following:

```
<div><span [ngClass]="styling">Highlighted</span></div>
```

If you switch over to the **Browser** that was opened with **ng serve –open** from the **Command Prompt** you will see the Text of *Highlighted* with a Background of *yellow*.

You can use **ngStyle** to set one or more inline Styles, for example to create a Behaviour to modify a **Link** based on a **Variable** to use in a **Method**. In **Visual Studio Code** in the **Component** for the Application of *app.component.ts* below the **Comment** for **// Variables** type in the following **Variable** after any previously declared **Variables**:

```
selected: boolean = false;
styles: Record<string, string> = {};
```

Then while still in the **Component** for the Application of *app.component.ts* below the **Comment** for **// Methods** and below any previous **Methods** type in the following **Method**:

```
setStyle() {
  this.selected = !this.selected;
  this.styles = {
    'font-weight': this.selected ? 'bold' : 'normal'
  };
}
```

The **Record** of **string** and **string** allows for multiple values to be used for the Style to be modified as the **Key** and then use the **Value** to set this accordingly such as to **bold** when **selected** is **true** and **normal** when **selected** is **false** which is toggled by use of the **!** or not **Operator** meaning anything that is **true** becomes **false** and anything that is **false** becomes **true**.

Then in the **Template** of the **Component** of *app.component.html* the following can be typed in below ** <span [ngClass]="styling">Highlighted **:

```
<div><a href="#" (click)="setStyle()" [ngStyle]="styles">Toggle Style</a></div>
```

This is an Element of an **a** Tag that when an **Event** for **click** is trigged it will toggle it between being *bold* and *normal*. You can see this by going to the **Browser** and Clicking on the **Link** for *Toggle Style*.

You can use **ngModel** for **Two-way Data Binding** for Elements in **HTML**, this can be used to use a Value in one place to display it in another in a **Component**. In the **Visual Studio Code** there will be a **Module** of *app.module.ts* which can be found in the **Folder** within **src** then **app**. In an **Angular** Application a **Module** helps organise code and maintain separation between features.

In **Visual Studio Code** within the **Module** for the Application of *app.module.ts* below **import { SizerComponent } from './sizer/sizer.component';** an additional **Import** should be added by typing in the following:

```
import { FormsModule } from '@angular/forms';
```

Then while still in the **Module** for the Application of *app.module.ts* type in the following below **BrowserModule** in the **imports** section:

```
,FormsModule
```

In the **Component** for the Application of *app.component.ts* below the **Comment** for **// Variables** type in the following **Variable** after any previously declared **Variables**:

```
model: string = '';
```

To use the Value from the **Component** for the Application within the **Template** of app.*component*.html to show anything that is entered into an **input** in a **h2**, below ** Toggle Style ** type in the following:

```
<input type="text" name="model" [(ngModel)]="model"/>
<h2>{{model}}</h2>
```

If you switch over to the **Browser** then type anything into the **input** will be displayed below it in a **h2** tag.

Structural Directives

Structural Directives can be used to control **HTML** Layout and can add, remove or manipulate Elements they are used with, the common build-in ones are **`ngIf`**, **`ngFor`** and **`ngSwitch`**.

Structural Directive of **`ngIf`** can be used to either add or remove an Element using a **Conditional** expression which is something that is **`true`** or **`false`** known as a **`boolean`**.

In **Visual Studio Code**, within the **Component** for the Application of *app.component.ts* below the **Comment** for **`// Variables`** type in the following **Variable** after any previously declared **Variables**:

```
isShown: boolean = false;
```

Then while still in the **Component** for the Application of *app.component.ts* below the **Comment** for **`// Methods`** and below any previous **Methods** type in the following **Method**:

```
toggle() {
    this.isShown = !this.isShown;
}
```

This **Method** will toggle the **`boolean`** using the **`!`** operator so when **`isShown`** is **`false`** it will become **`true`** and when **`isShown`** that is **`true`** will become **`false`**.

Then while still in **Visual Studio Code** within the **Template** of app.*component*.html to call the **Method** that will use the **Method** of **`toggle`** for **`isShown`** and use this with **`ngIf`** to control the display of a **`h2`**, below **`<h2>{{model}}</h2>`** type in the following:

```
<button (click)="toggle()">Click Here</button>
<h2 *ngIf="isShown">Hello World!</h2>
```

If you switch over to the **Browser** there will be a **`button`** of *Click Here* when **Clicked** will show then hide a **`h2`** below with the Text of *Hello World!*

Structural Directive of **ngFor** can be used to display a list of items with a defining the block of **HTML** that will be used to display a single item of a list of items. The items can be defined by adding a **string[]** to store multiple values in an **Array**.

To do this return to **Visual Studio Code** and within the **Component** for the Application of *app.component.ts* below the **Comment** for **// Variables** type in the following **Variable** after any previously declared **Variables**:

```
items: string[] = ['Hello', 'World'];
```

Then while still in **Visual Studio Code** within the **Template** of app.*component*.html a **ul** or **Unordered List** and displayed as a **Bulleted List** can be defined with a **li** or **List Item** to display each item by typing in below **<h2 *ngIf="isShown">Hello World!</h2>** the following:

```
<ul><li *ngFor="let item of items">{{item}}</li></ul>
```

If you switch over to the **Browser** there will be a **Bulleted List** showing the **List Items** of *Hello* and *World*.

Structural Directive of **ngSwitch** can be used to display Elements from a possible set of Elements. **ngSwitch** is comprised of three **Directives** which are **ngSwitch** which gets the **Value** and will work with **ngSwitchCase** which will add an element when the **Value** matches and **ngSwitchDefault** where there is no **Value** matching an **ngSwitchCase**.

Return to **Visual Studio Code** and within the **Component** for the Application of *app.component.ts* to define a list of items that will be used, below the **Comment** for **// Variables** type in the following **Variable** after any previously declared **Variables**:

```
values: any[] = [
  {
    name: 'None',
    status: ''
  },
  {
    name: 'Danger',
    status: 'red'
  },
  {
    name: 'Warning',
    status: 'yellow'
  },
  {
    name: 'Proceed',
    status: 'green'
  }
];
```

Then while still in **Visual Studio Code** within the **Template** of app.*component*.html for the list of items to use **ngFor** to display each item then a **ngSwitch** to display them with the correct status using a **ngSwitchCase** for each status along with **ngSwitchDefault** for a default status by typing in below **
<li *ngFor="let item of items">{{item}}
** the following:

```
<ul>
    <li *ngFor="let value of values" [ngSwitch]="value.status">
        <span *ngSwitchCase="'red'" style="background-color: red">
          Danger
        </span>
        <span *ngSwitchCase="'yellow'" style="background-color: yellow">
          Warning
        </span>
        <span *ngSwitchCase="'green'" style="background-color: green">
          Proceed
        </span>
        <span *ngSwitchDefault>
          None
        </span>
    </li>
</ul>
```

If you switch over to the **Browser** there will be another **Bulleted List** showing the **List Items** of *None*, then *Danger* with a *red* Background, *Warning* with a yellow Background and *Proceed* with a *green* Background.

Dependency Injection

Dependency Injection is a design pattern that allows **Dependencies** to be requested from an external source rather than being created internally, these **Dependencies** are provided to **Classes** on Instantiation via the **Constructor** using **Angular DI** to create a **Service** to provide functionality.

After following **Setup and Start**, **Components**, **Templates** and **Directives** you can create **Service** on **Windows** using the **Command Prompt** and the **Angular CLI**, by going to **Start** then search for **Command Prompt** and then select it, then **cd** to the **Folder** for your Application e.g. *C:\Angular\workshop* with the following command and then press **Enter**:

```
cd C:\Angular\workshop
```

Then while still in the **Command Prompt** use the **Angular CLI** to create a new **Service** by typing in the following and then press **Enter**:

```
ng generate service demo
```

You can close this **Command Prompt** once done and then go to **Visual Studio Code** and within the **Explorer** you will find one of two new files within the **Folder** of **src** then **app** of *demo.service.ts* for the **Service** and within this below the **`constructor() { }`** line type in the following **Method**:

```
getMessage() {
  return 'Hello Demo!';
}
```

The **Method** of **`getMessage()`** will return the Text of **`Hello Demo!`**, to use this **Service** in the Application, in **Visual Studio Code** within the **Component** of *app.component.ts* below **`import { Component } from '@angular/core';`** type in the following to **`import`** the **Service**:

```
import { DemoService } from './demo.service';
```

While in the **Component** of *app.component.ts* above the **Comment** of **`// Methods`** type in the following:

```
constructor(public demoService: DemoService) { }
```

This **Constructor** allow the **Service** to be **Injected** into the **Constructor** and make a **`public`** Instance of the **`DemoService`** available to be used, to use this within the **Template** for the Application of *app.component.html* at the end of file below **`</ul>`** type in the following:

```
<h2>{{demoService.getMessage()}}</h2>
```

If you switch over to the **Browser** you will see the *Hello Demo!* Message being displayed and that concludes this **Workshop** about **Angular** from tutorialr.com!