Angular 2 with Visual Studio

Developer:	*Developer*		
Author(s):	*Majo Manoj*	Project	*PROJECT*
Document Status:	*Initial Draft*	**Date of Last Change:**	*22/2/2017*

Revision History

Date	Version	Description	Author
22-Feb-2017	0.1	Initial Document Creation	

Contents

Introduction 3
Setting up environment 3
Components of Angular 2 6
Module 6
Components 7
Main.ts........ 8
Common Controls 8
One way binding 8
Two way binding 8
Looping Through an Arraylist 9
Handling Click Events 9
Condition Statement 10
Getting User Inputs using Forms 10
Directives in angular 2 11
Routing in angular 2 12
Creating a Service with http (Ajax) call in angular 2 14

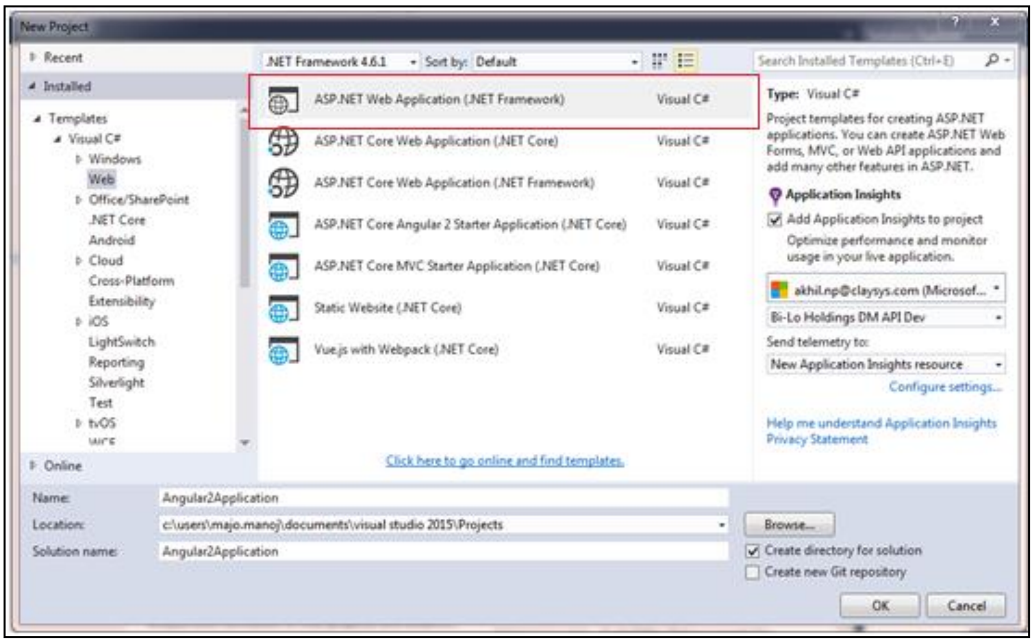
Introduction

In this document we will discuss basics of Angular 2.Angular 2 is the latest web development framework from Google. Besides this angular 2 is one of the most popular framework for developing single page application. Angular 2 follows modular approach and each components has to be included according the need, which reduces the loading overhead and thus has a quick response time.

Setting up environment

- Download and install NodeJS from "https://nodejs.org/en/download/"
- Run command npm --v on cmd. If it gives an error. Do the bellow steps 4 steps.
  i) Use the global Search Charm to search "Environment Variables".
  ii) Click "Edit system environment variables".
  iii) Click "Environment Variables" in the dialog.
  iv) In the "System Variables" box, search for Path and edit it to include C:\Program Files\nodejs. Make sure it is separated from any other pathsby ";". You will have to restart any currently-opened command prompts before it will take effect.

- Make sure you have visual studio update 3.
- Install typescript 2.1.5 for visual studio from "https://www.microsoft.com/en-us/download/details.aspx?id=48593"
- Download Angular 2 quickstart files from https://github.com/angular/quickstart and extract them.
- Create a new asp.net web application give a name and click ok.



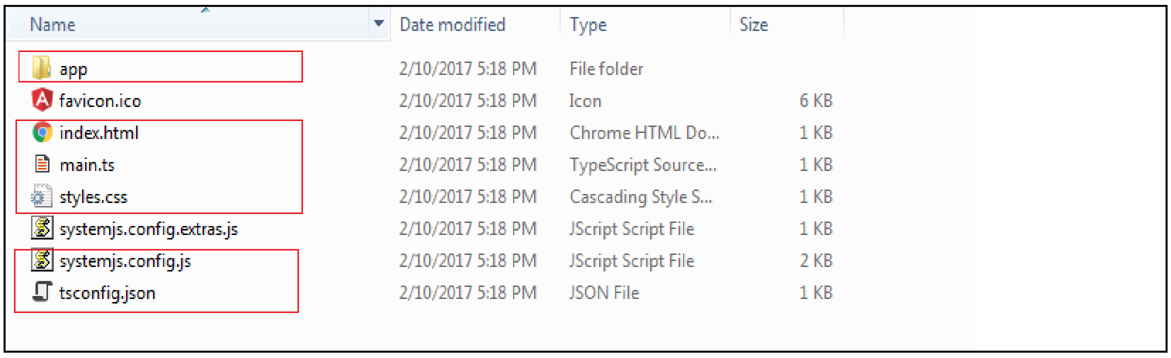
ASP.NET Web Application (.NET Framework) Visual C#
ASP.NET Core Web Application (.NET Core) Visual C#
ASP.NET Core Web Application (.NET Framework) Visual C#
ASP.NET Core Angular 2 Starter Application (.NET Core) Visual C#
ASP.NET Core MVC Starter Application (.NET Core) Visual C#
Static Website (.NET Core) Visual C#
Vue.js with Webpack (.NET Core) Visual C#

Type: Visual C#
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights
☑ Add Application Insights to project
Optimize performance and monitor usage in your live application.
akhil.np@claysys.com (Microsof...
Bi-Lo Holdings DM API Dev
Send telemetry to:
New Application Insights resource
Configure settings...
Help me understand Application Insights
Privacy Statement

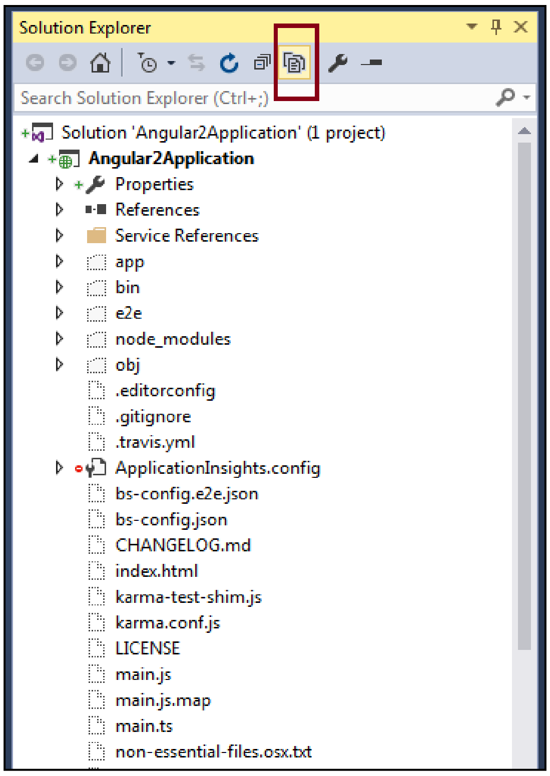
Name: Angular2Application
Location: c:\users\majo.manoj\documents\visual studio 2015\Projects
Solution name: Angular2Application
☑ Create directory for solution
☐ Create new Git repository

- Now choose an empty application template and click Ok.
- Now go back to extracted quickstart files folder open it and copy package.json, then thesefiles from src folder.



Name	Date modified	Type	Size
app	2/10/2017 5:18 PM	File folder	
favicon.ico	2/10/2017 5:18 PM	Icon	6 KB
index.html	2/10/2017 5:18 PM	Chrome HTML Do...	1 KB
main.ts	2/10/2017 5:18 PM	TypeScript Source...	1 KB
styles.css	2/10/2017 5:18 PM	Cascading Style S...	1 KB
systemjs.config.extras.js	2/10/2017 5:18 PM	JScript Script File	1 KB
systemjs.config.js	2/10/2017 5:18 PM	JScript Script File	2 KB
tsconfig.json	2/10/2017 5:18 PM	JSON File	1 KB

- Place these files in the newly created project folder containing **.csproj** file.
- Now go back to visual studio and refresh solution explorer.
- Click the show all files button in solution explorer taskbar, It will show all the hidden files in project folder.



Solution Explorer

Search Solution Explorer (Ctrl+;)

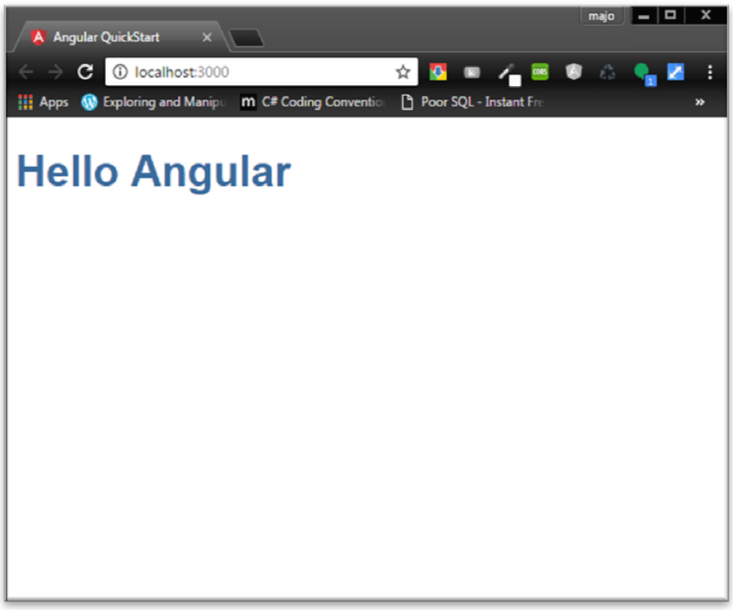
```
Solution 'Angular2Application' (1 project)
  Angular2Application
    Properties
    References
    Service References
    app
    bin
    e2e
    node_modules
    obj
    .editorconfig
    .gitignore
    .travis.yml
    ApplicationInsights.config
    bs-config.e2e.json
    bs-config.json
    CHANGELOG.md
    index.html
    karma-test-shim.js
    karma.conf.js
    LICENSE
    main.js
    main.js.map
    main.ts
    non-essential-files.osx.txt
```

- Now you can see the copied Files.
- Select the files that were copied right click and include them in to folder.
- Now go to project.json file and in dependencies change the typescript version to the installed version (2.1.5).
- Now right click on the project.json file and select restore packages.
- Visual Studio will download all the packages.

- After completion run the application and we will get "Hello Angular" on screen.
- On running, Main html page (index.html) is loaded. It has the parent Component tag in its body (<my-app>).

```
<body>
  <my-app>Loading AppComponent content here ...</my-app>
</body>
```

- Browser view of application on running.



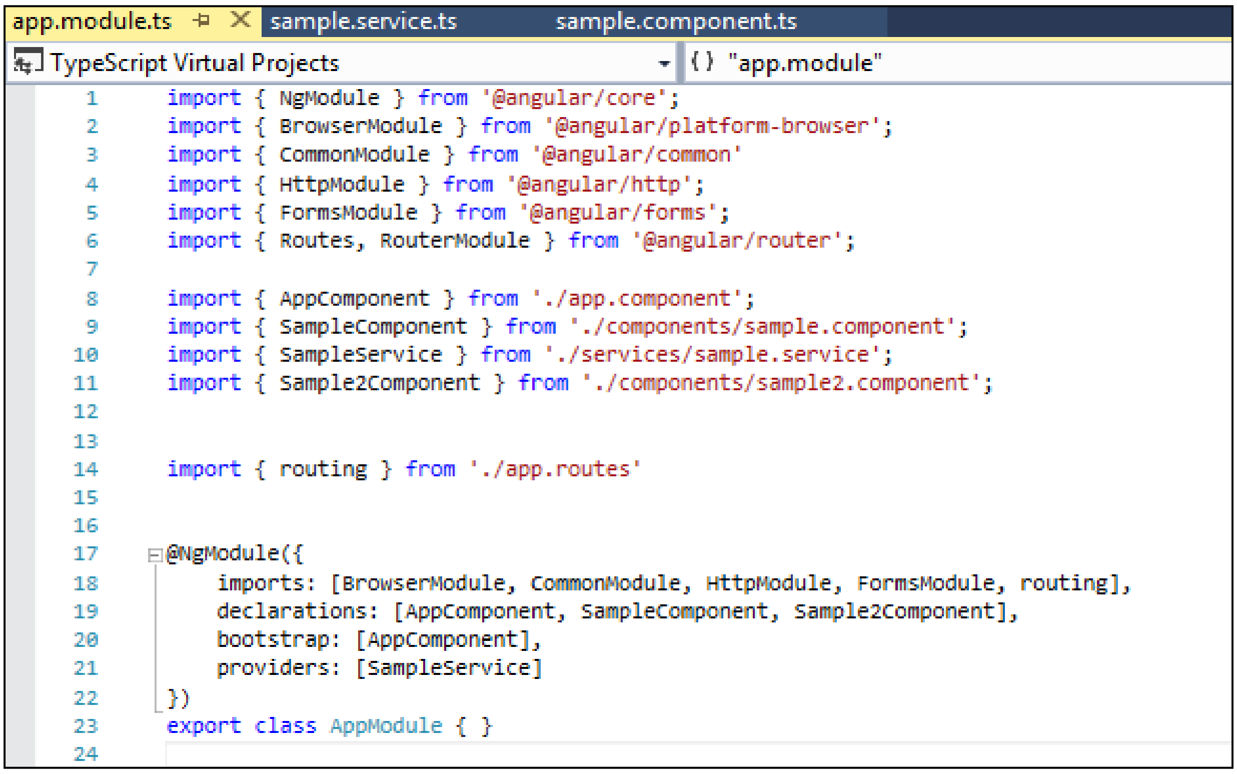
Hello Angular

Components of Angular 2

Modules and components are the basics of building angular 2 applications.

Module is a group of components and services. Components are Sections of code that are related to a similar task.

Module



app.module.ts | sample.service.ts | sample.component.ts

TypeScript Virtual Projects | {} "app.module"

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common'
import { HttpModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { Routes, RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { SampleComponent } from './components/sample.component';
import { SampleService } from './services/sample.service';
import { Sample2Component } from './components/sample2.component';


import { routing } from './app.routes'


@NgModule({
    imports: [BrowserModule, CommonModule, HttpModule, FormsModule, routing],
    declarations: [AppComponent, SampleComponent, Sample2Component],
    bootstrap: [AppComponent],
    providers: [SampleService]
})
export class AppModule { }
```

This (app.module.ts) is our basic module. **NgModule** should be imported from @angular/core to use @NgModule annotation. You can see we have imported different other modules to use it in this module. This is similar to having the **"using"** clause in the C#. In the above example there are many external modules imported but these needs to be imported in your case based on your need. You have to import and register all the components and services for this module.

@NgModule annotation

This notation is used for defining a module. Below are the basic selectors present in @NgModule annotation.

Imports

All the external Modules to be used in the project have to be registered in this. For example **HttpModule** is an external module that is used for making Http calls,**BrowserModule** for letting angular render views to the browser.**CommonModule** for using ng tags in html pages. If you have your own external module you can include it here.

Declarations

It is the place where all the components in the module are to be registered.

Bootstrap

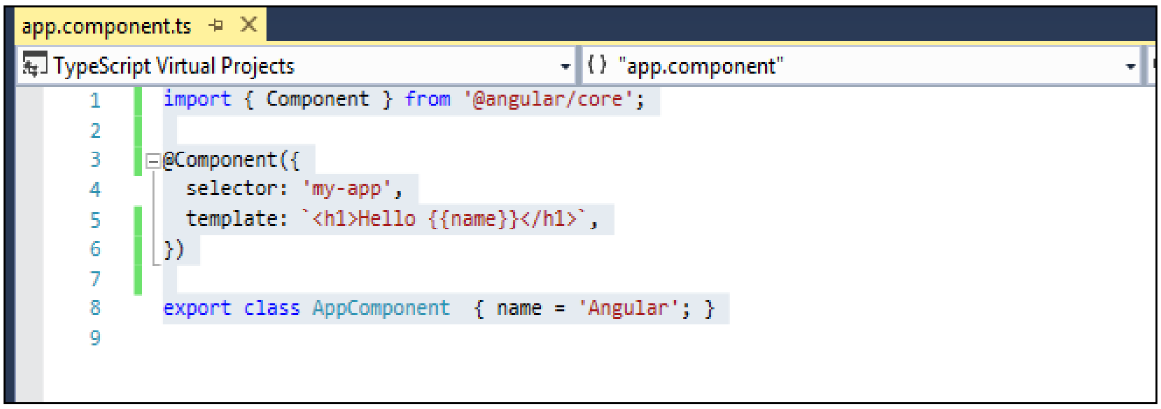
It is used for bootstrapping our application. The parent component is registered here.

Providers

Here all the services have to be registered here.

Components

app.component is the basic parent component present in our app. We have to import Component from@angular/core to use Components.



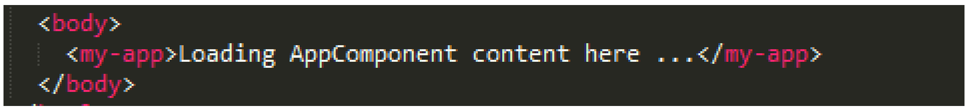
```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})

export class AppComponent  { name = 'Angular'; }

```

@Component

To let angular know we are dealing with components we use this annotation. Basic Properties present in @Component are

selector

This notation lets angular know that we are defining a component which can be used in our html page.Here my-app is the main component selector.



```
<body>
  <my-app>Loading AppComponent content here ...</my-app>
</body>
```

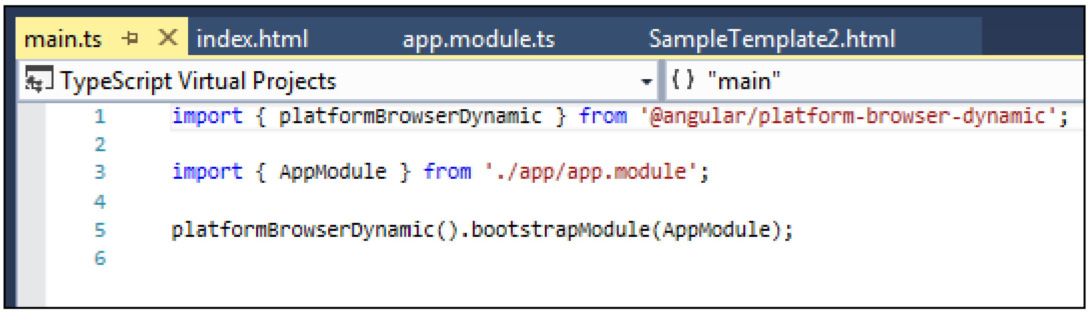
template or templateUrl

These two are used to define how the views look like. We can either use template to give the html tags or we can use templateURL to give a path of html file present in our application.

providers

Providers are used to register services that are to be used in our component. You will need to import services or components that need to be used in the current component.

Main.ts



```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

There is a **main.ts** page which is the starting point of application. Here we register the app module and **platformBrowserDynamic** which is used to render angular2 in to a web browser.

```
<script>
  System.import('main.js').catch(function(err){ console.error(err); });
</script>
```

We have to include our main in our html file.

Common Controls

One way binding

[innerText] is used for binding variable in angular2

```
<h1 [innerText]="employee.name"></h1>
```

Two way binding

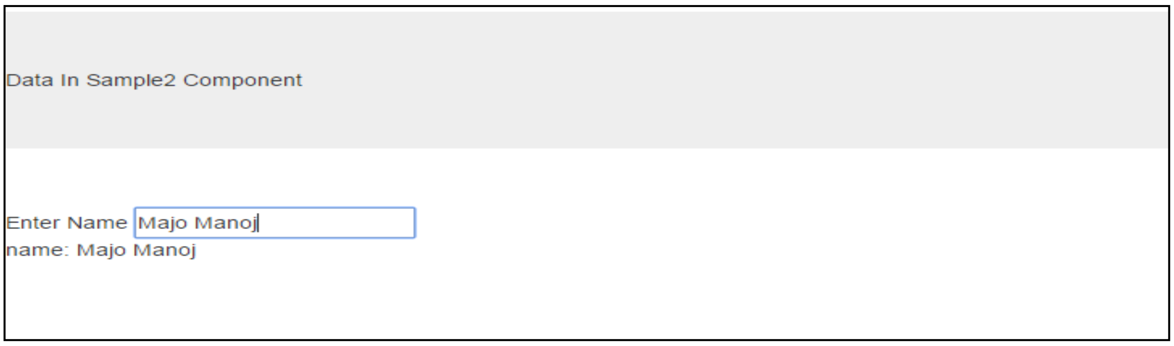
By using this feature of angular2 we can bind variable each time it's value is changed. To do that we will use **[(ngModel)]**keyword.

example:

```
Enter Name<inputtype="text"placeholder="two-way binding"[(ngModel)]="variable"/>
name: {{variable}}
```

{{ }} is used to display the value of the variable on to the browser.

Here when we input some text in to the textbox, the value gets bound to the control instantly.


Data In Sample2 Component

Enter Name Majo Manoj

name: Majo Manoj

Looping Through an Arraylist

If we have an array of objects and we have to render it to screen, some type of looping must be done, to do that we have to use *[ngFor].

In our ajax call we get all the data in to listSampleData variable. We cannot display this data withoutlooping.

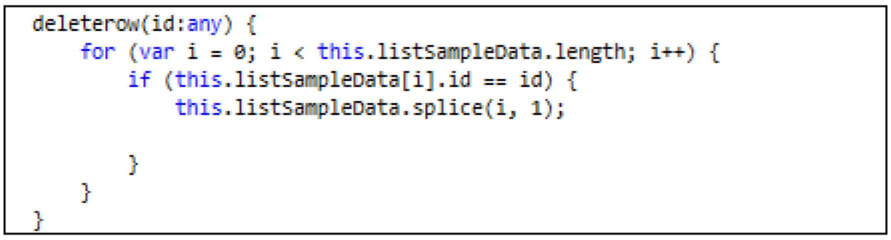
```
<table class="table-bordered" style="overflow-y:scroll;height:400px;display:block;"width="70%">
    <tr>
        <th>id</th>
        <th>name</th>
        <th>delete</th>
    </tr>
    <tbody style="height: 200px;overflow-y:scroll;">
        <tr *ngFor="let item of (listSampleData)">
            <td>{{item.id}}</td>
            <td>{{item.title}}</td>
            <td><button class="btn" (click)="deleterow(item.id)">delete</button></td>
        </tr>
    </tbody>
</table>
```

Handling Click Events

```
<table class="table-bordered" style="overflow-y:scroll;height:400px;display:block;"width="70%">
    <tr>
        <th>id</th>
        <th>name</th>
        <th>delete</th>
    </tr>
    <tbody style="height: 200px;overflow-y:scroll;">
        <tr *ngFor="let item of (listSampleData)">
            <td>{{item.id}}</td>
            <td>{{item.title}}</td>
            <td><button class="btn" (click)="deleterow(item.id)">delete</button></td>
        </tr>
    </tbody>
</table>
```

Here we can see **(click)** event this is similar to OnClick() in asp.net.

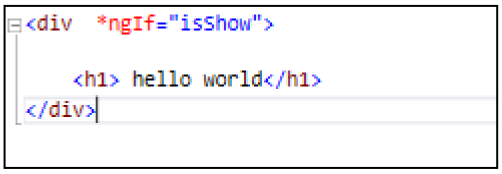
Here when a click event happens deleterow method is called with a parameter, clicked id



```
deleterow(id:any) {
    for (var i = 0; i < this.listSampleData.length; i++) {
        if (this.listSampleData[i].id == id) {
            this.listSampleData.splice(i, 1);

        }
    }
}
```

Condition Statement

We can use *ngIf to hide or show div tags in angular 2



```
<div  *ngIf="isShow">

    <h1> hello world</h1>
</div>
```

isShow is a Boolean variable and if the value is true div is shown else it stays hidden.

Getting User Inputs using Forms

Forms are important part of a web application. Forms are used to get user inputs. In angular 2 we have Formbulider for creating forms.

- To use formbulider we have to import **FormModule** and **FormBuilder** from **@angular/forms** to the component. Also import validators from **@angular/forms**for validations

- We also have to add **FormModule** and **ReactiveFormsModule** in to the app.module and these to NgModule**imports** as dependencies.

- Here is a sample login form created using formbulider.

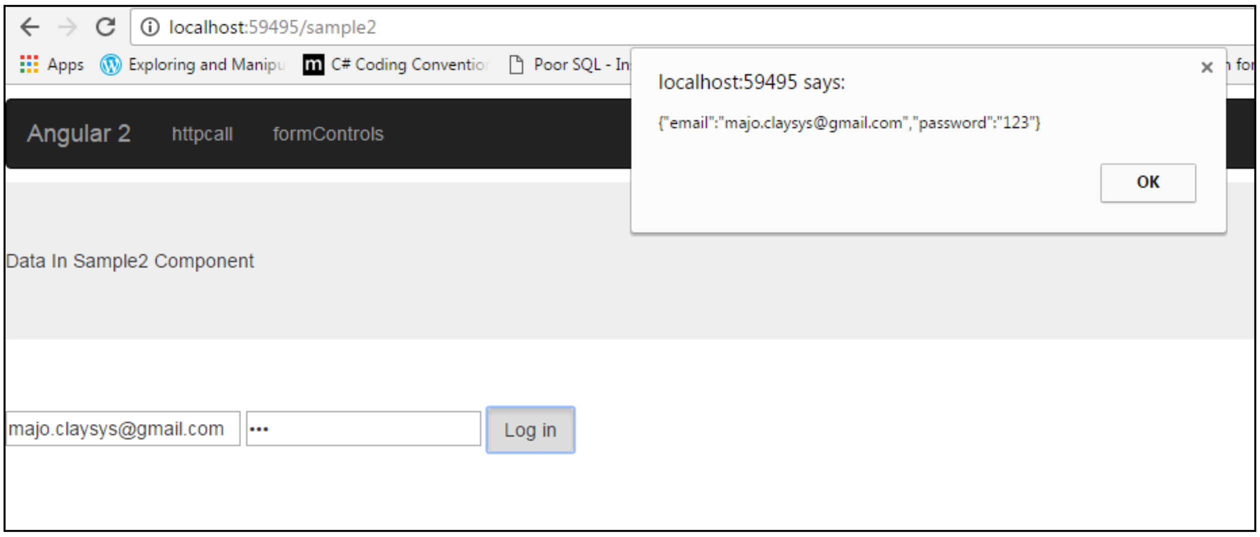
```
publicloginForm = this.fb.group({
email: ["", Validators.requiredTrue],
password: ["", Validators.required]
         });
```

- Now from inside the html page.

```
<form[formGroup]="loginForm"(ngSubmit)="doLogin($event)">
<inputformControlName="email"type="email"placeholder="Your email"required>
<inputformControlName="password"type="password"placeholder="Your password"
      required>
<buttonclass="btn"type="submit">Log in</button>
</form>
```

- Here when the user clicks submit button doLogin method is invoked we have to specify our actions from inside this method.

```
doLogin(event: Event) {

     console.log(event);
     alert(JSON.stringify(this.loginForm.value));
     console.log(this.loginForm.value);
  }
```



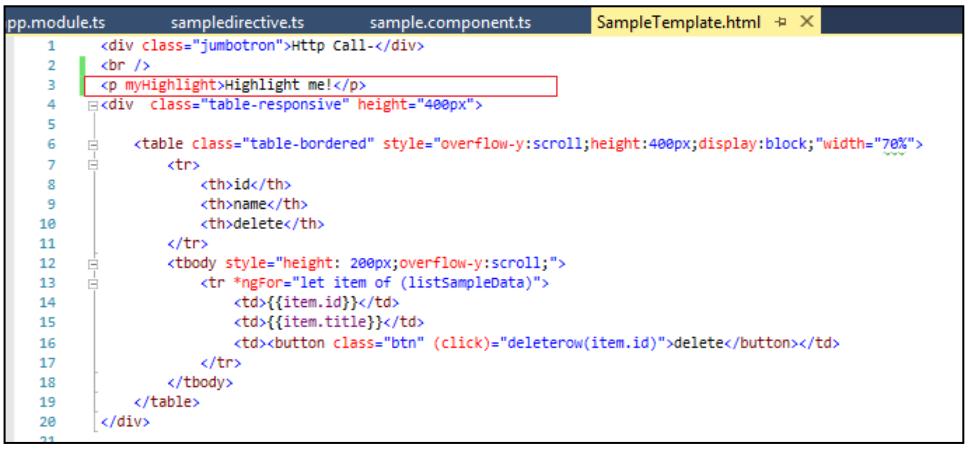
localhost:59495 says:

```
{"email":"majo.claysys@gmail.com","password":"123"}
```

Data In Sample2 Component

Directives in angular 2

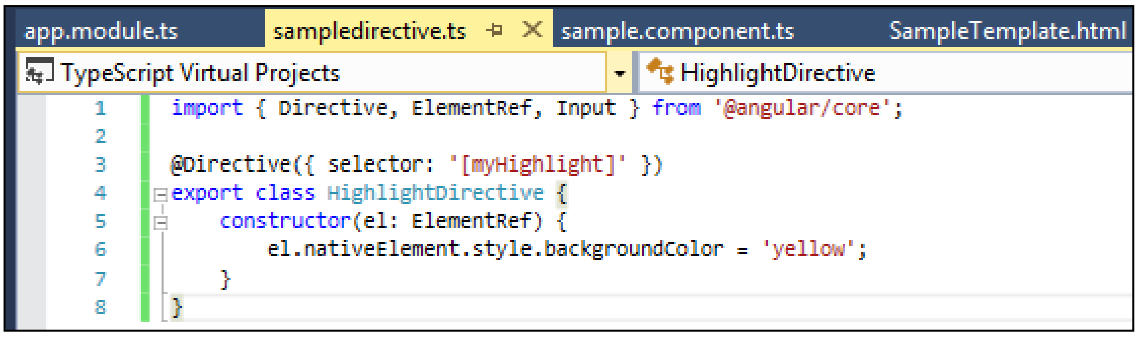
In angular 2 we can use custom tags to perform some actions.



```html
<div class="jumbotron">Http Call-</div>
<br />
<p myHighlight>Highlight me!</p>
<div  class="table-responsive" height="400px">

    <table class="table-bordered" style="overflow-y:scroll;height:400px;display:block;"width="70%">
        <tr>
            <th>id</th>
            <th>name</th>
            <th>delete</th>
        </tr>
        <tbody style="height: 200px;overflow-y:scroll;">
            <tr *ngFor="let item of (listSampleData)">
                <td>{{item.id}}</td>
                <td>{{item.title}}</td>
                <td><button class="btn" (click)="deleterow(item.id)">delete</button></td>
            </tr>
        </tbody>
    </table>
</div>
```

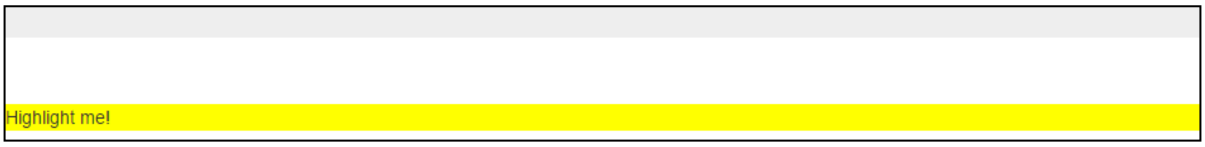
Here **myHighlight**is a user defined keyword. To do this we have to

- Create a new ts file in directives folder.
- Import Directives, Input and ElementRef(to get DOM elements)from @angular/core.
- Use @Directive to register a new directive and specify the selector name for directive.
- Write the methods for the directive. Here are highlighting a DOM element.



```
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({ selector: '[myHighlight]' })
export class HighlightDirective {
    constructor(el: ElementRef) {
        el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

- Now register the component in app module declarations.
- Import directive file in to the component we want to use.
- Use the directive in an element and the output we get is.



Highlight me!

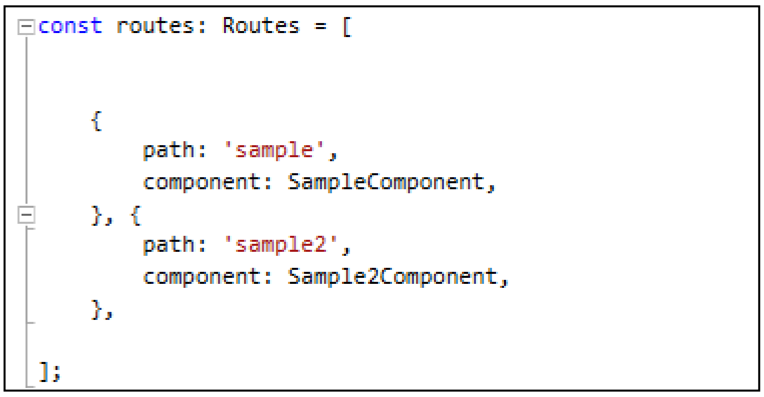
Routing in angular 2

Routing is used to navigate between different views or components in angular2.

In angular 2 routing is done on the parent component, which means parent component is always shown. The views are swapped from inside parent component by using `<router-outlet></router-outlet>`

For routing

- Create a new typescript file called app.routes.
- Import Routes and RouterModule from @angular/router.
- Import all the components you have to route between.
- Create const variable named routes of type Routes.
- To that variable specify routes as given bellow.



```
const routes: Routes = [


    {
        path: 'sample',
        component: SampleComponent,
    }, {
        path: 'sample2',
        component: Sample2Component,
    },

];
```

- Export the route like bellow

```
export const routing = RouterModule.forRoot(routes);
```

- Register the routes in app.module as

```
import { routing } from './app.routes'


@NgModule({
    imports: [BrowserModule, CommonModule, HttpModule, FormsModule, routing],
    declarations: [AppComponent, SampleComponent, Sample2Component],
    bootstrap: [AppComponent],
    providers: [SampleService]
})
export class AppModule { }
```

- Now specify base address in index.html page as

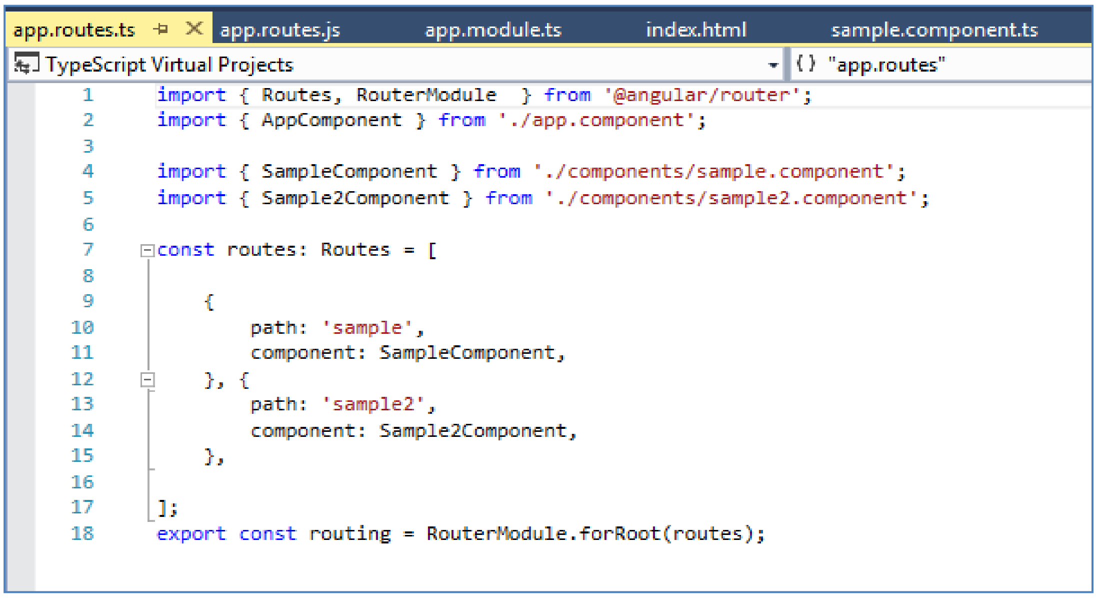
```
    <base href="/">
```

- Now put where our views are to be showed in app.component (parent component) by usingbellow tag.

```
<router-outlet></router-outlet>
```

- Finally specify when routes are to be changed (Links). Here The routes are changed on nav bar click.

```
<ul class="nav navbar-nav">
    <li>
        <a href="#" [routerLink]="['/sample']">
            Click to show Sample Component
        </a>
    </li>
    <li>
        <a href="#" [routerLink]="['/sample2']">
            Click to show Sample Component2
        </a>
    </li>
```



```
import { Routes, RouterModule  } from '@angular/router';
import { AppComponent } from './app.component';

import { SampleComponent } from './components/sample.component';
import { Sample2Component } from './components/sample2.component';

const routes: Routes = [

    {
        path: 'sample',
        component: SampleComponent,
    }, {
        path: 'sample2',
        component: Sample2Component,
    },

];
export const routing = RouterModule.forRoot(routes);
```

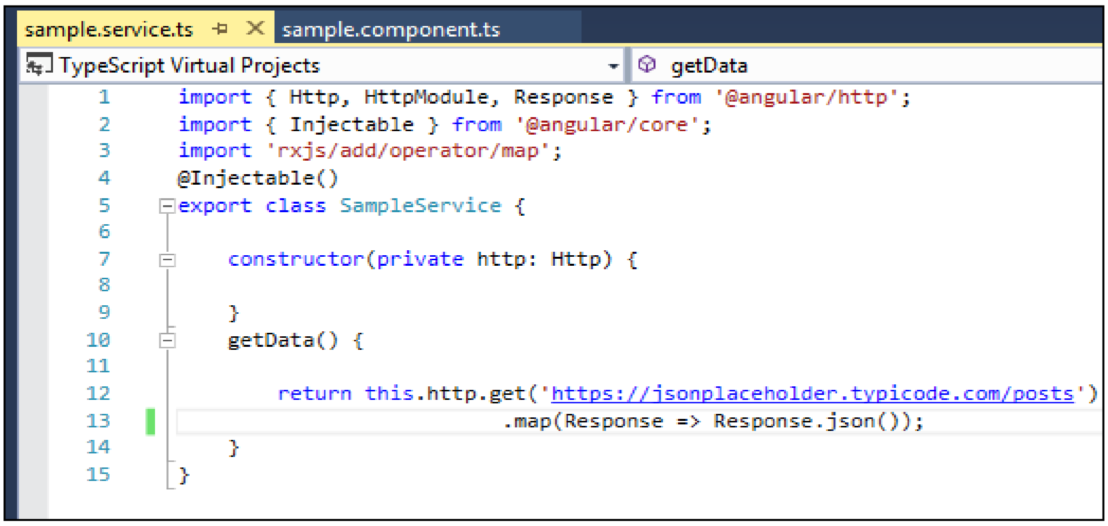
app.routes.ts

Creating a Service with http (Ajax) call in angular 2

Services are used for storing data or functions that is to be reused or shared among different components. The data inside components is lost each time they are inter changed. Ajax call is used to replace parts of our application. It is faster and more efficient than loading the entire page. We usually make http calls from inside a service.

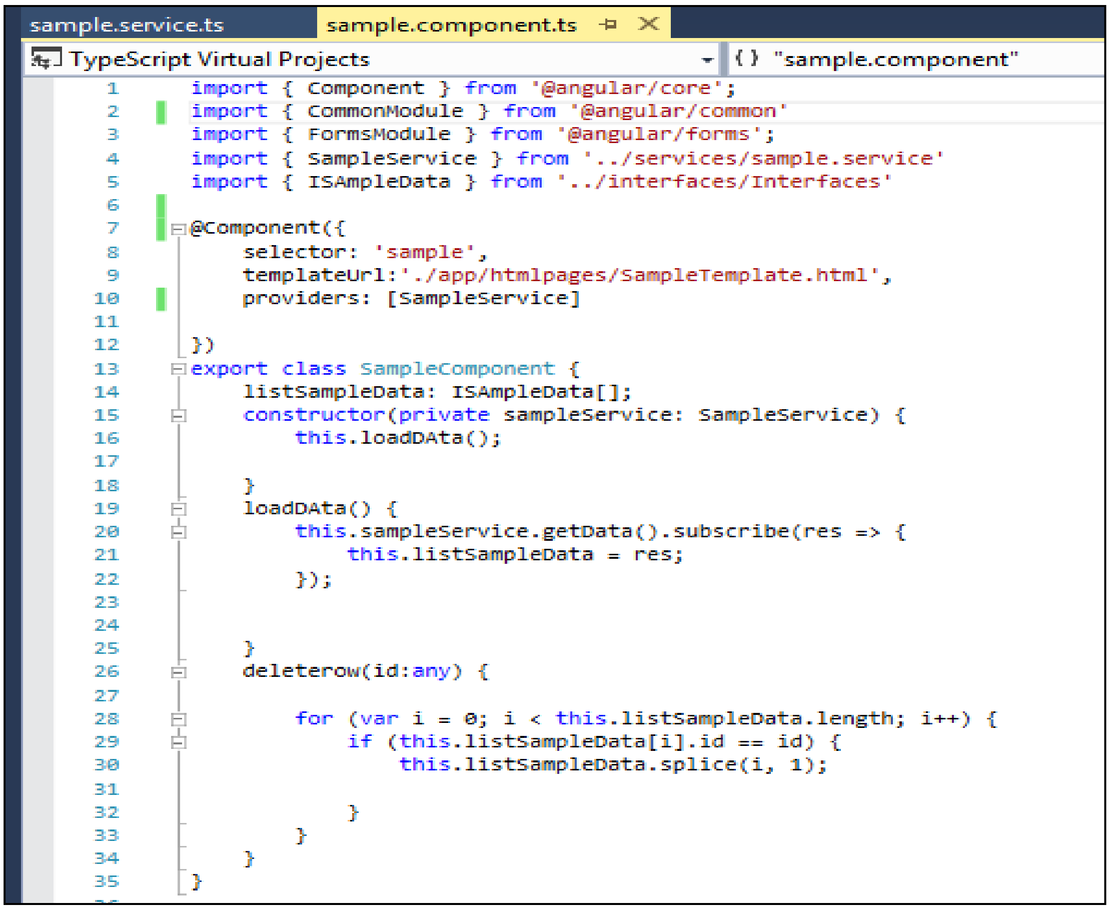
To create a service with http call

- **ImportInjectable** from @angular/core.
- **@Injectable()** annotation is used to let angular know we are creating a service .
- Create a Service class and defines required methods and variables inside it.
- To make http call you will have to **import http httpmodule and responsefrom @angular/http** and also import rxjs.
- Register httpmodule in app.module as well.
- You have to **register Service** in the components from which service will be accessed also have to register it in module.
- To do that in we will have to import service class from the location of the service.ts file.
- Then in the **provides** tag from inside @Component register the service class.



```
import { Http, HttpModule, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import 'rxjs/add/operator/map';
@Injectable()
export class SampleService {

    constructor(private http: Http) {

    }
    getData() {

        return this.http.get('https://jsonplaceholder.typicode.com/posts')
                              .map(Response => Response.json());
    }
}
```

service.ts

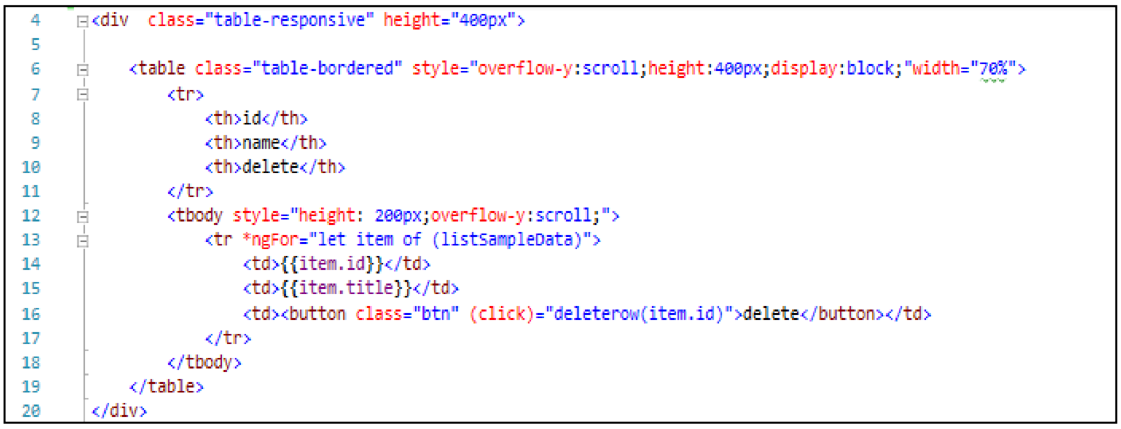

sample.service.ts | sample.component.ts

TypeScript Virtual Projects | {} "sample.component"

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common'
import { FormsModule } from '@angular/forms';
import { SampleService } from '../services/sample.service'
import { ISAmpleData } from '../interfaces/Interfaces'

@Component({
     selector: 'sample',
     templateUrl:'./app/htmlpages/SampleTemplate.html',
     providers: [SampleService]

})
export class SampleComponent {
     listSampleData: ISAmpleData[];
     constructor(private sampleService: SampleService) {
         this.loadDAta();

     }
     loadDAta() {
         this.sampleService.getData().subscribe(res => {
             this.listSampleData = res;
         });


     }
     deleterow(id:any) {

         for (var i = 0; i < this.listSampleData.length; i++) {
             if (this.listSampleData[i].id == id) {
                 this.listSampleData.splice(i, 1);

             }
         }
     }
}
```

component.ts

- Registering Service in Component.
- After registering service in component provider, we can simply use it in constructor and access the methods and variables in that service.
- Similarly we have to register service in provider of the module.
- To display the data in html page.


```
<div  class="table-responsive" height="400px">

    <table class="table-bordered" style="overflow-y:scroll;height:400px;display:block;"width="70%">
        <tr>
            <th>id</th>
            <th>name</th>
            <th>delete</th>
        </tr>
        <tbody style="height: 200px;overflow-y:scroll;">
            <tr *ngFor="let item of (listSampleData)">
                <td>{{item.id}}</td>
                <td>{{item.title}}</td>
                <td><button class="btn" (click)="deleterow(item.id)">delete</button></td>
            </tr>
        </tbody>
    </table>
</div>
```