

Learn Angular 2 step by step

By

www.questpond.com

Version 0.1 (Beta)

This book is in very preliminary stage , but if you are beginner in angular it will still serve the purpose.

Contents

Acronym used in this book.....	2
Lab 1 :- Getting started with Simple Angular 2 UI	2
Introduction	2
What is the Goal of Angular ?	3
Pre-requisite for Angular 2 before you start.....	5
Step 1:- Installing VS Code, Typescript and Node	5
With modularity comes great responsibility.....	10
Step 2:- Setting up Angular environment	11
Step 3:- Configuring the task runner.....	14
Understanding Angular 2 Component and module architecture	16
Step 4:- Following MVW Step by Step – Creating the folders	16
Step 5:- Creating the model	18
Step 6:- Creating the Component	19
Step 7:- Creating the Customer HTML UI – Directives and Interpolation.....	22
Step 8:- Creating the Module.....	23
Step 9:- Creating the “Startup.ts” file	24
Step 10:- Invoking “Startup.ts” file using main angular page	25
Step 11:- Installing http-server and running the application.....	27
How to the run the source code?	27
Lab 2 :- Implementing SPA using Angular 2 routing	28
Fundamental of Single page application.....	28
Step 1 :- Creating the Master Page	28

Step 2:- Creating the Supplier page and welcome page.....	29
Step 3:- Renaming placeholder in Index.html.....	30
Step 4:- Removing selector from CustomerComponent.....	30
Step 5:- Creating Components for Master , Supplier and Welcome page.....	31
Step 6: - Creating the routing constant collection	32
Step 7: - Defining routerLink and router-outlet.....	34
Step 8:- Loading the routing in Main modules	35
Step 9:- Seeing the output	36
Step 10:- Fixing Cannot match any routes error	37
Understanding the flow	37
Lab 3 :- Handling Refresh (HashRouting)	38

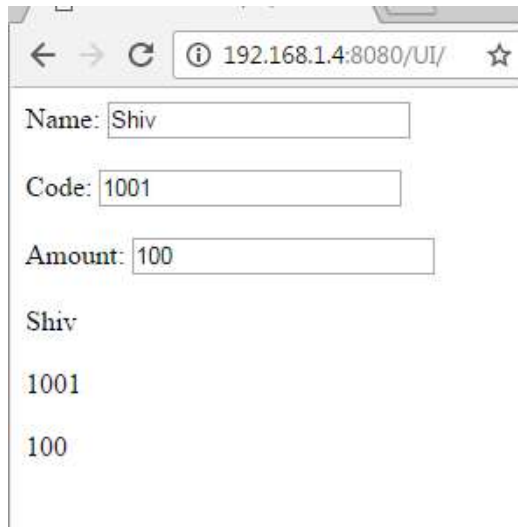
Acronym used in this book

- NPM :- Node package manager.
- TS :- TypeScript.
- JS :- Javascript.
- VS :- Visual studio.
- VS Code :- Visual studio code.
- WP :- Webpack
- OS :- Operating system.

Lab 1 :- Getting started with Simple Angular 2 UI

Introduction

In this book we will Learn Angular 2 Step by step. This book has 10 Labs and each Lab is guided by detailed steps.



The best way to learn any new technology is by creating a project. So in this step by step series we will be creating a simple Customer data entry screen project.

So for Lab 1 we will create a basic screen in which we will create three fields "Name", "Code" and "Amount". When the end user types in these 3 fields the values will be displayed down below.

So in lab 1 we will execute 11 steps to achieve the same.

Now this is a 200 page book, so reading needs patience. So if you want to go super-fast you can watch the below Angular 2 video. This video explains how to get your first Angular 2 video program running. Please note the youtube video uses Visual studio as tool while this book uses VS code.

So I have kept both options open so that Microsoft guys are happy and the non-Microsoft people do not feel left over.

<https://www.youtube.com/watch?v=oMgvi-AV-eY>

What is the Goal of Angular ?

"Angular is an open source JavaScript framework which simplifies binding code between JavaScript objects and HTML UI elements."

Let us try to understand the above definition with simple sample code.

Below is a simple "Customer" function with "CustomerName" property. We have also created an object called as "Cust" which is of "Customer" class type.

```
function Customer()
{
    this.CustomerName = "AngularInterview";
}
var Cust = new Customer();
```

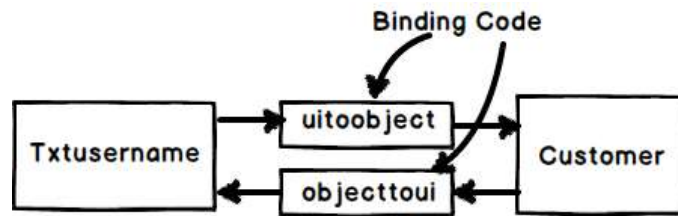
Now let us say in the above customer object we want to bind to a HTML text box called as "TxtCustomerName". In other words when we change something in the HTML text box the customer object should get updated and when something is changed internally in the customer object the UI should get updated.

```
<input type=text id="TxtCustomerName" onchange="UitoObject()" />
```

So in order to achieve this communication between UI to object developers end up writing functions as shown below. “UitoObject” function takes data from UI and sets it to the object while the other function “ObjecttoUI” takes data from the object and sets it to UI.

```
function UitoObject()
{
    Cust.CustomerName = $("#TxtCustomerName").val();
}
function ObjecttoUi()
{
    $("#TxtCustomerName").val(Cust.CustomerName);
}
```

So if we analyze the above code visually it looks something as shown below. Your both functions are nothing but binding code logic which transfers data from UI to object and vice versa.

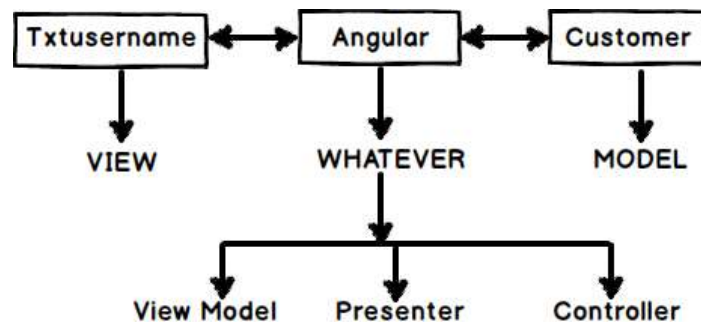


Binding Code

Now the same above code can be written in Angular as shown below. So now whatever you type in the textbox updates the “Customer” object and when the “Customer” object gets updated it also updates the UI.

```
<input type=text [(ngModel)]="Customer.CustomerName"/>
```

In short if you now analyze the above code visually you end up with something as shown in the below figure. You have the VIEW which is in HTML, your MODEL objects which are javascript functions and the binding code in Angular.



Javascript Functions

Now that binding code have different vocabularies.

- Some developers called it “ViewModel” because it connects the “Model” and the “View” .
- Some call it “Presenter” because this logic is nothing but presentation logic.
- Some term it has “Controller” because it controls how the view and the model will communicate.

To avoid this vocabulary confusion Angular team has termed this code as “Whatever”. It’s that “Whatever” code which binds the UI and the Model. That’s why you will hear lot of developers saying Angular implements “MVW” architecture.

So concluding the whole goal of Angular is Binding, Binding and Binding.

Pre-requisite for Angular 2 before you start

In my initial journey of learning Angular 2 I realized that Angular 2 by itself is easy to understand. But Angular 2 uses lot of JavaScript open sources and if you do not know these open sources Learning Angular 2 would become extremely difficult.

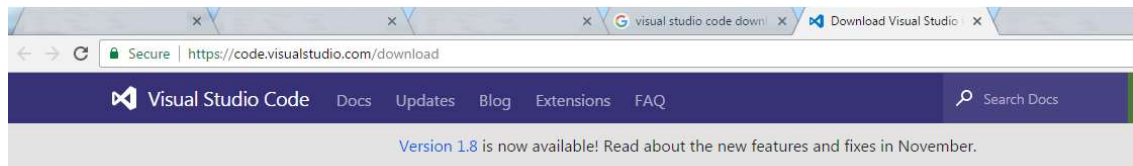
So we would recommend spending your some time going through the below videos and making notes around the same:-

Topic name	YouTube URL source
Node JS	https://www.youtube.com/watch?v=-LF_43_Mqnw
Type Script	https://www.youtube.com/watch?v=xqYD8QXJX9U
System JS	https://www.youtube.com/watch?v=nQGhdoIMKaM
Common JS concept	https://www.youtube.com/watch?v=jN4IM5tp1SE

Step 1:- Installing VS Code, Typescript and Node

Theoretically you can do Angular 2 with a simple notepad. But then that would be going back to back ages of adam and eve and reinventing the wheel. So in order to expedite the learning process we would be using some tools and software’s.

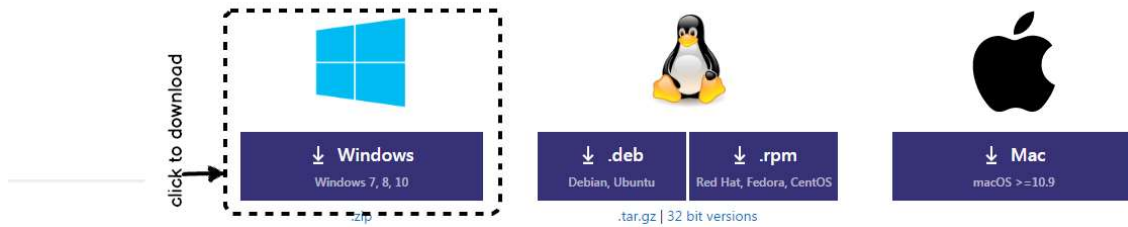
The first tool which we would need is Visual studio code editor or in short you can say VS code editor. So go to <https://code.visualstudio.com/download> and depending on your operating system install the appropriate one. For instance I am having windows OS so I will be installing the windows version.



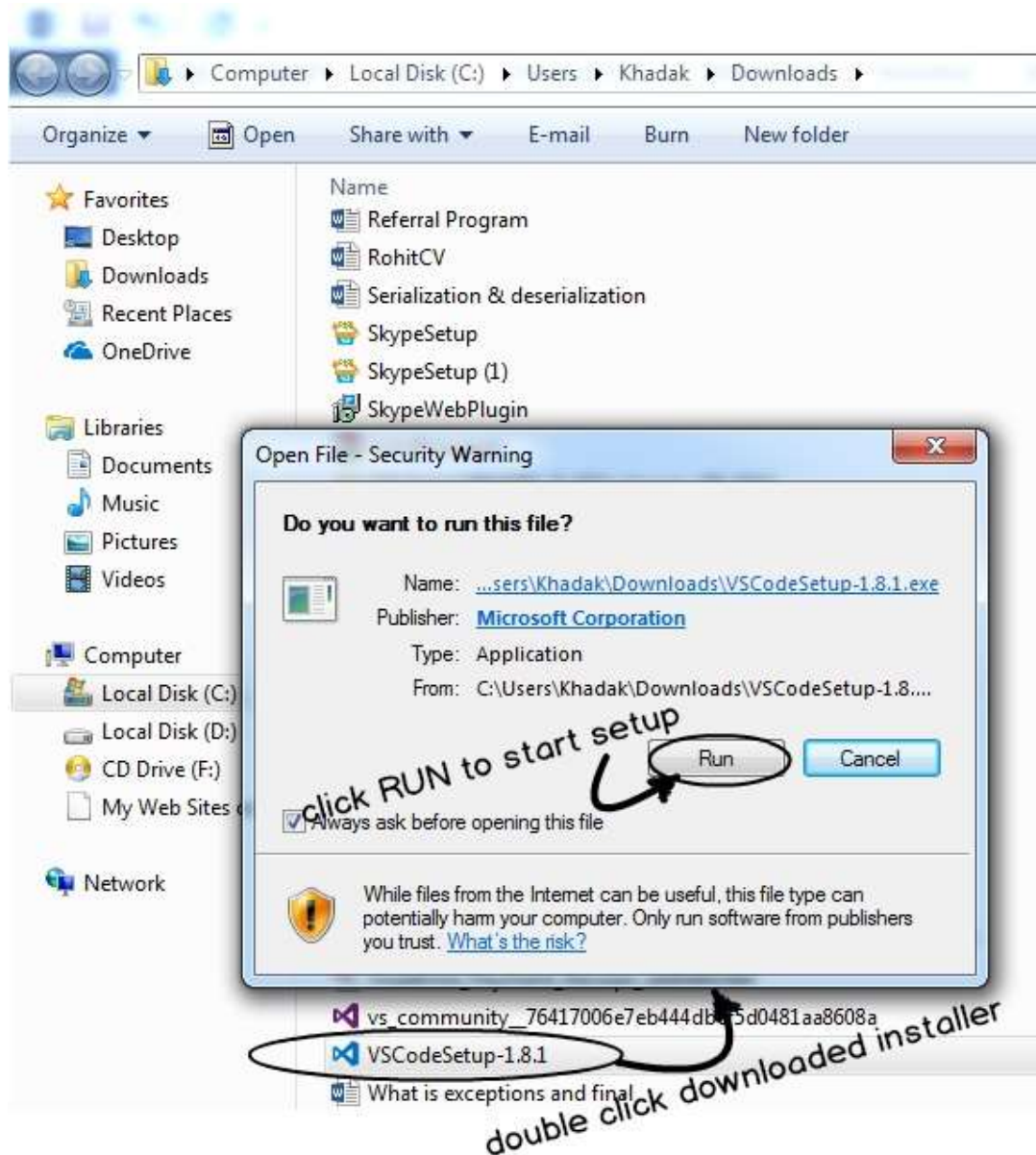
Download Visual Studio Code

Free and open source. Integrated Git, debugging and extensions.

* - this tutorial is created with Windows OS
so we will use Windows download



Once you download the setup it's a simple setup EXE run it and just hit next , next and finish.



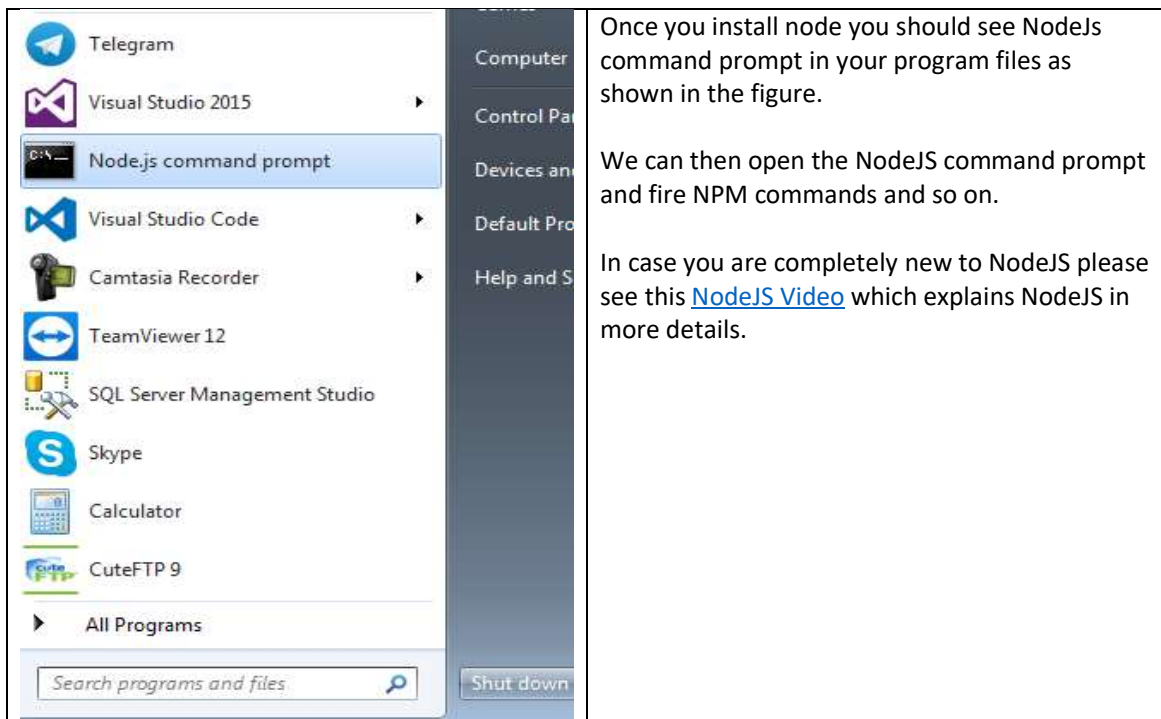
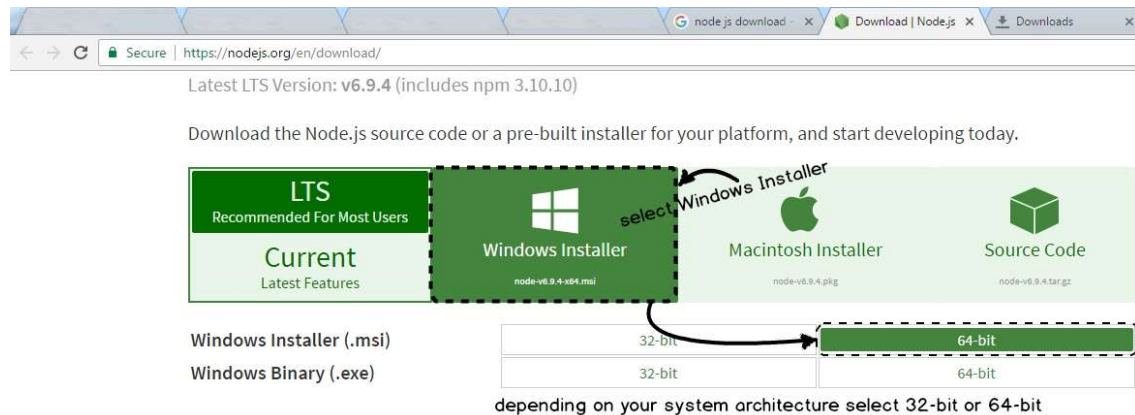
The second thing we need is “Nodejs”. We would encourage you to watch this video which explains the basic use of Node JS , [Node JS Explained](#).

If I put in simple words NodeJs does the following two things:-

1. It has something called as “NPM” node package manager. If you watch around you will see lot of JS frameworks coming up every day. Each one of these frameworks have their own website, github repository with weird names and they are releasing new versions now and then. So to get these frameworks we need to go to their site download the JS files and so on. NPM is a central repository where all these things are registered. And as developer if you want to get a specific JS open source you can just type NPM commands like “npm install jquery”. This command will bring the recent version of JQuery in your computer folder.

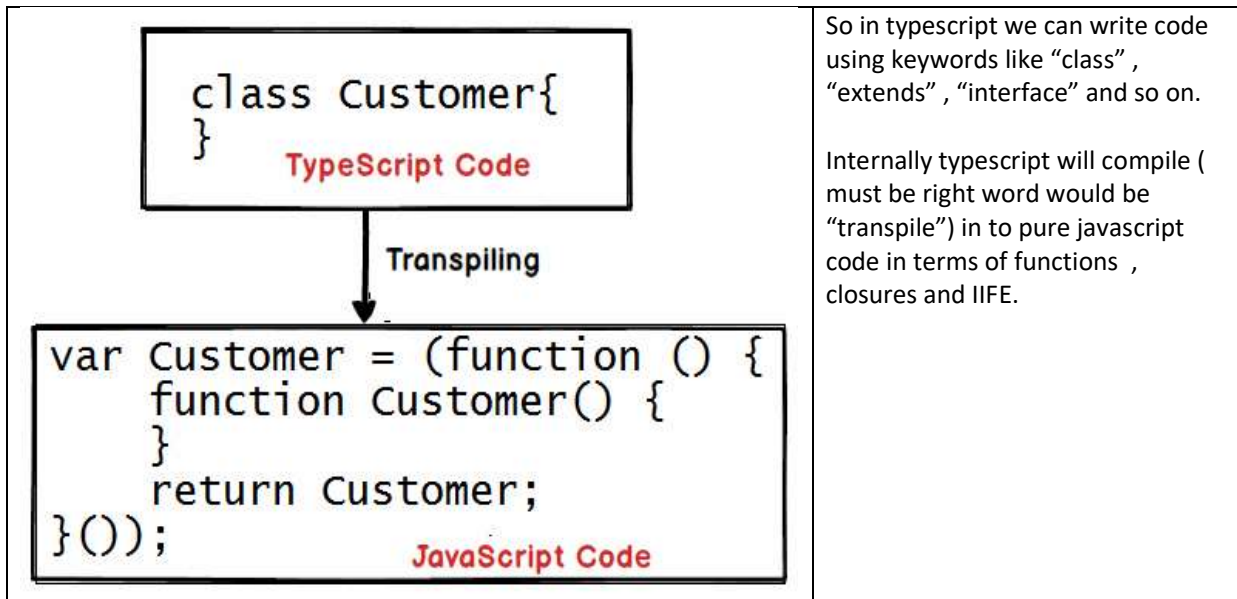
2. The second thing which node does is it helps you to run JavaScript outside the browser. So tomorrow if you want to build a server application or windows application using JavaScript then this thing might help.

So to get node you can go to [www.nodejs.org](https://nodejs.org) and depending on your OS select the appropriate download as shown in the below figure.



The third thing for Angular 2 which we need is typescript. If I put in simple words:-

"TypeScript is a sugar coated Object oriented programming language over JavaScript."

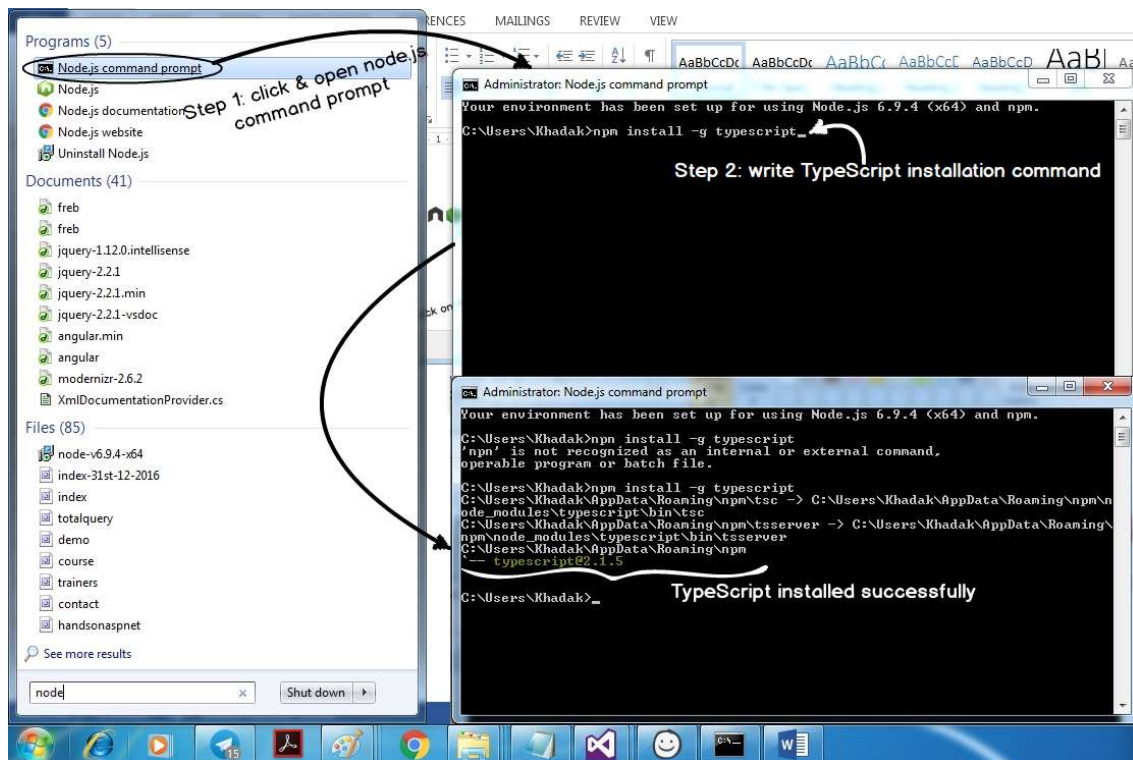


Please do watch this 1 hour [Training video on TypeScript](#) which explains Typescript in more detail.

So as we said TypeScript is javascript open source we can get the same from the node. Remember in the previous section we discussed that node has NPM by which we can get latest versions of JS framework.

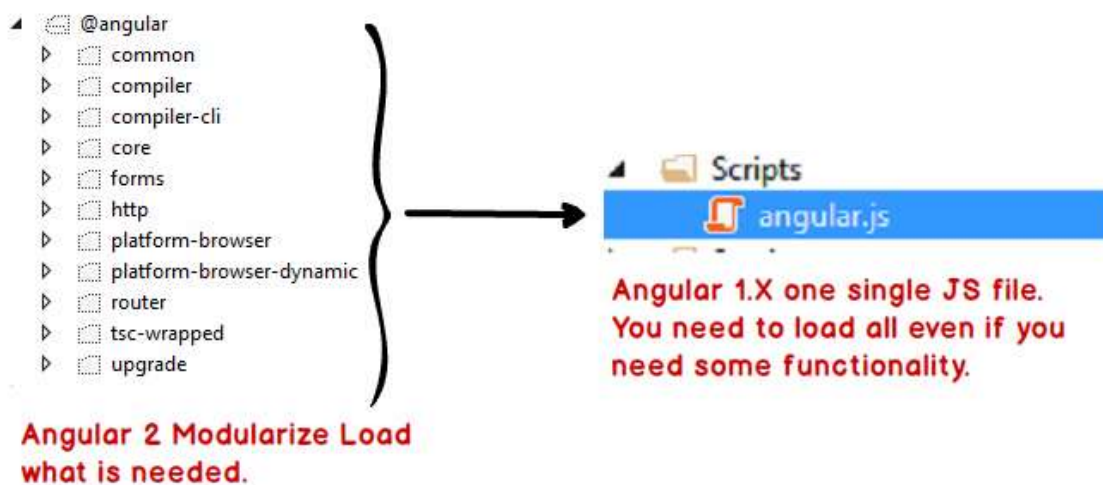
So click on node command prompt and in the command prompt type “npm install –g typescript”. This command will install typescript in your PC and make it available global throughout the computer.

Note :- The “-g” command makes typescript available throughout the computer from any folder command prompt.



So to start with Angular we need at least these three things one editor we have chosen VS Code , Node (NPM) for getting JavaScript frameworks and Typescript so that we can code JavaScript faster and with Object Oriented coding approach.

With modularity comes great responsibility



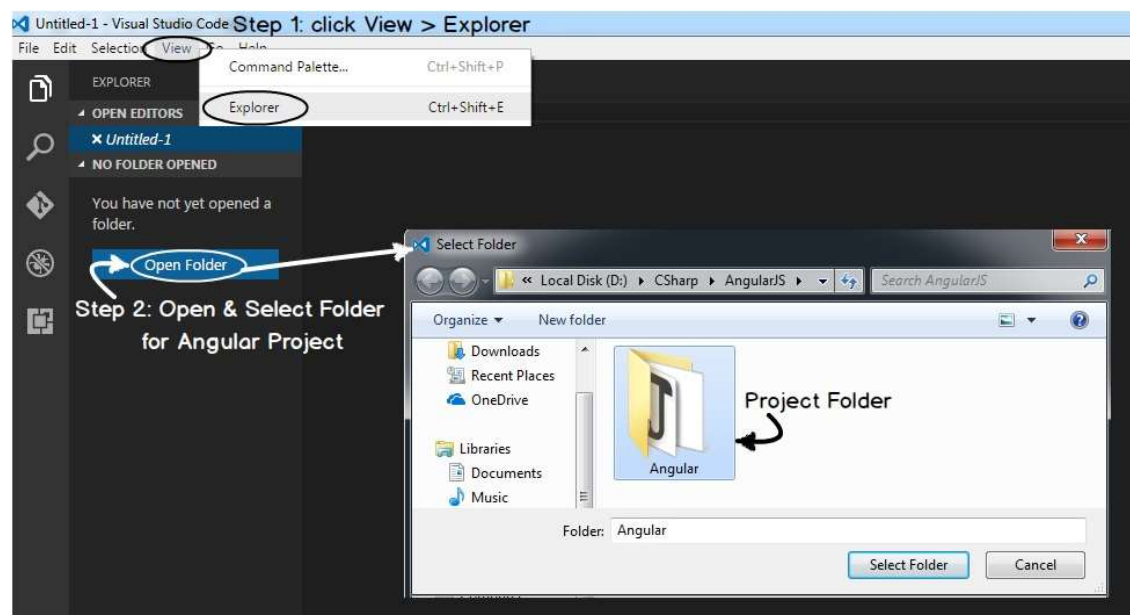
In Angular 1.X we just had [one JS file](#) which had the whole framework. So you drag and drop that single Angular 1.X JS file on HTML page and you are ready with the Angular 1.X environment. But the problem with having whole framework in one JS files was that you have to include all features even if you need it or not.

For instance if you are not using HTTP still that feature will be loaded. In case of Angular 2 we have separate discrete components. These are self-contained components which can be loaded in an isolated manner rather than loading the whole JS file.

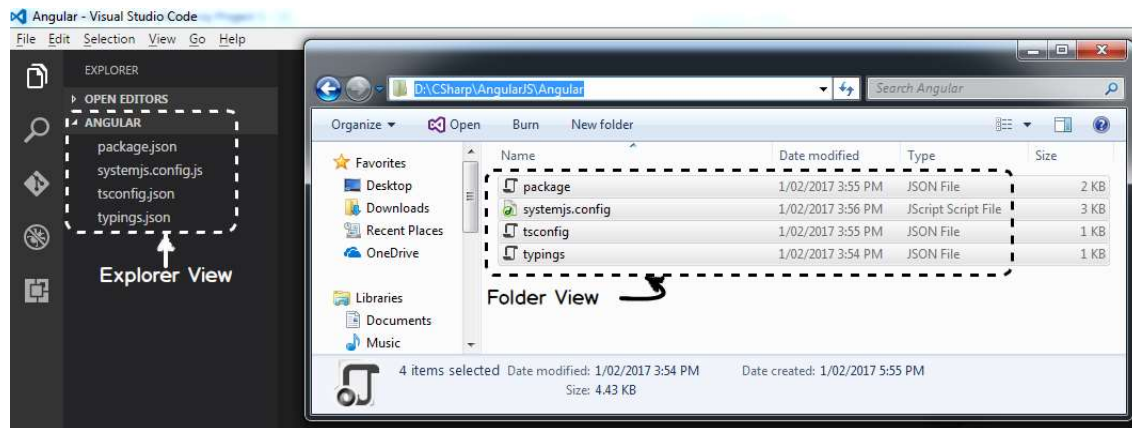
Now because we have lot of self-contained components in other words we have lot of single JS files creating a proper environment itself is a challenge. So let us first understand how to setup angular environment.

Step 2:- Setting up Angular environment

So the first step is to create a folder and open the folder using VS Code as shown in the below image.



Inside this folder we need to paste the 4 JSON files. You can download these files from this URL <http://tinyurl.com/jeehlqo>.

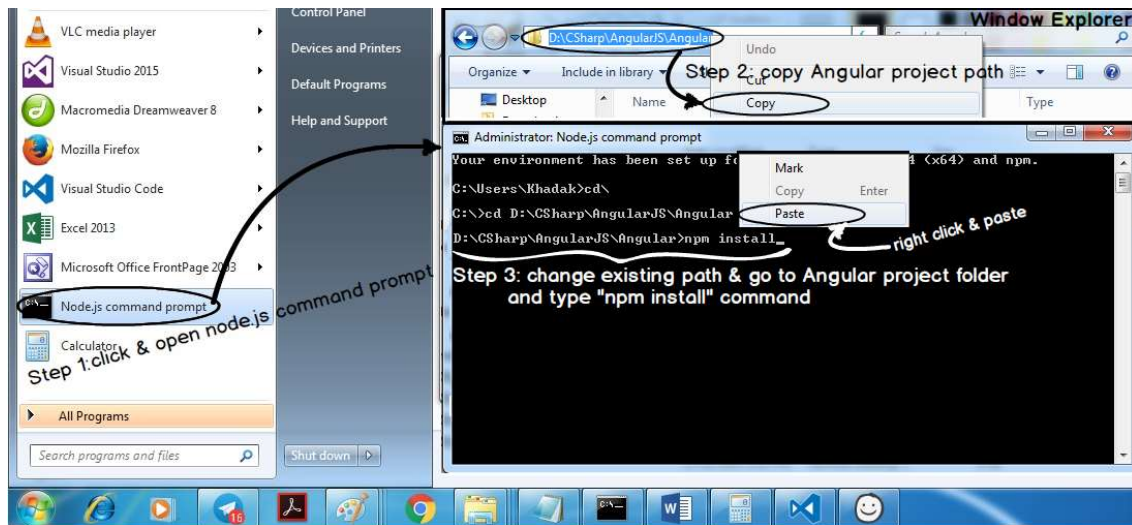


Please note these files are not created by me but I have got them from Angular 2 website or the respective open source website.

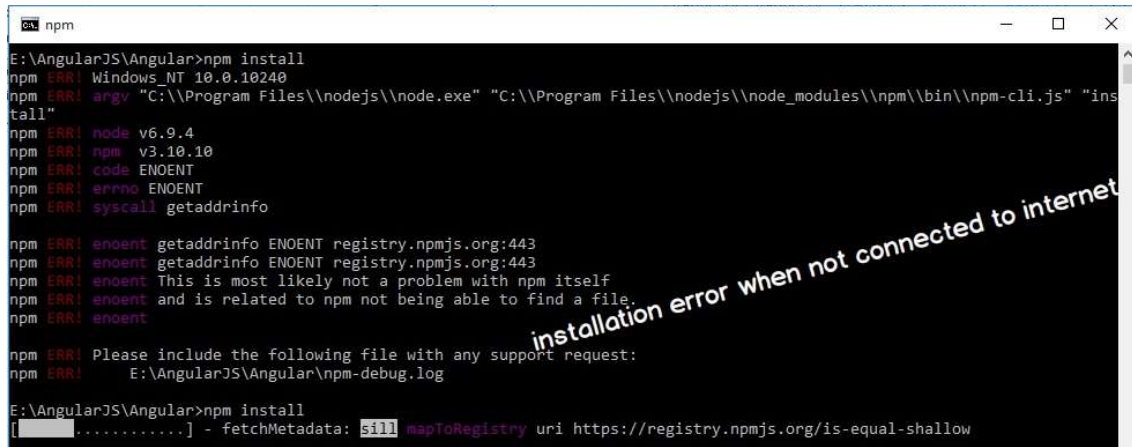
Below goes the explanation of those 4 JSON files:-

JSON File Name	What is the use of the file?
package.json	This file has the references of all the Angular 2 modules. Node will use this file to download all the Angular modular files. If you are new to node please go through the pre-requisite video section.
tsconfig.json	This is a configuration file for typescript and will be used by typescript to define how transpiling process takes place in typescript. If you are not aware of typescript please go through the pre-requisite videos.
typings.json	Typescript is a new language some of the old JS frameworks cannot be consumed in typescript. For those frameworks we need to define the typings.
systemjs.config	SystemJS is a module loader. This configuration file defines the configuration for SystemJS module loader. We will be talking more about it in the further coming steps.

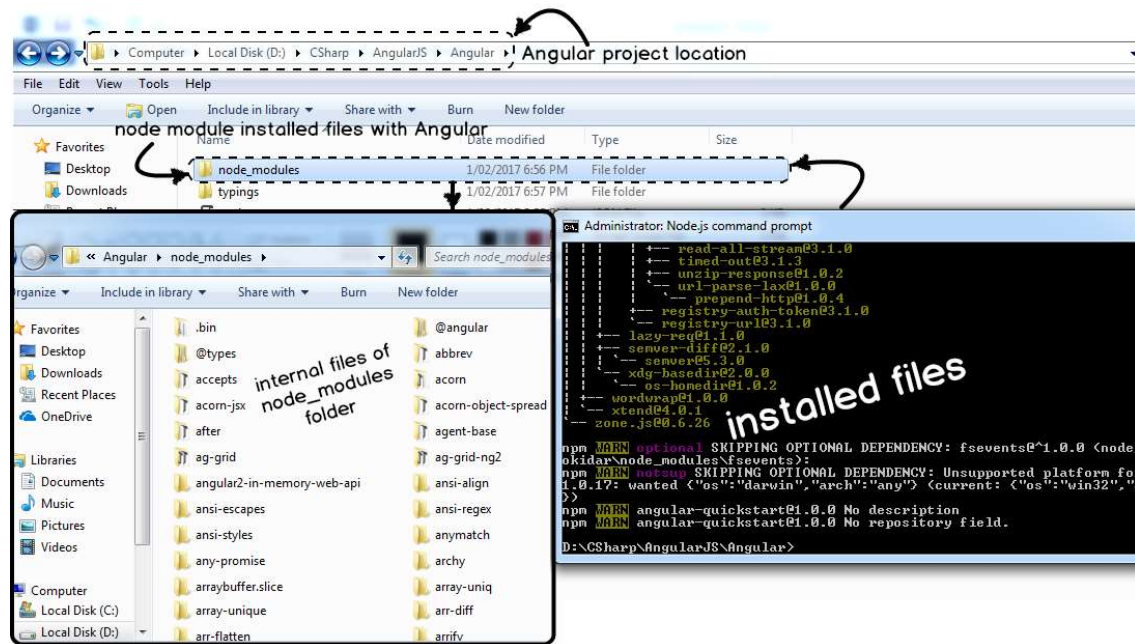
So now lets open "node" command prompt and type "npm install" in the command line. This command tries to get all the files which are mentioned in the package.json file.



You need to be connected to internet to get these files. If you are not connected to internet you would land up in to some kind of an error as shown in the below figure.



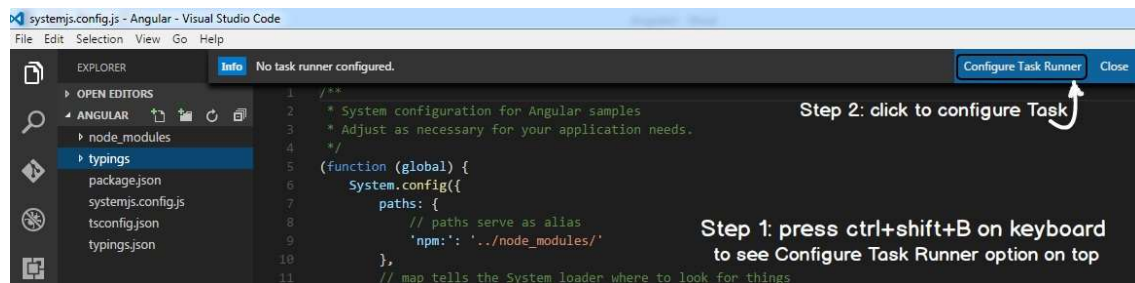
After the Angular installation is done successfully you will see "node_modules" folder created. In this folder all the JS files of Angular 2 has been downloaded.



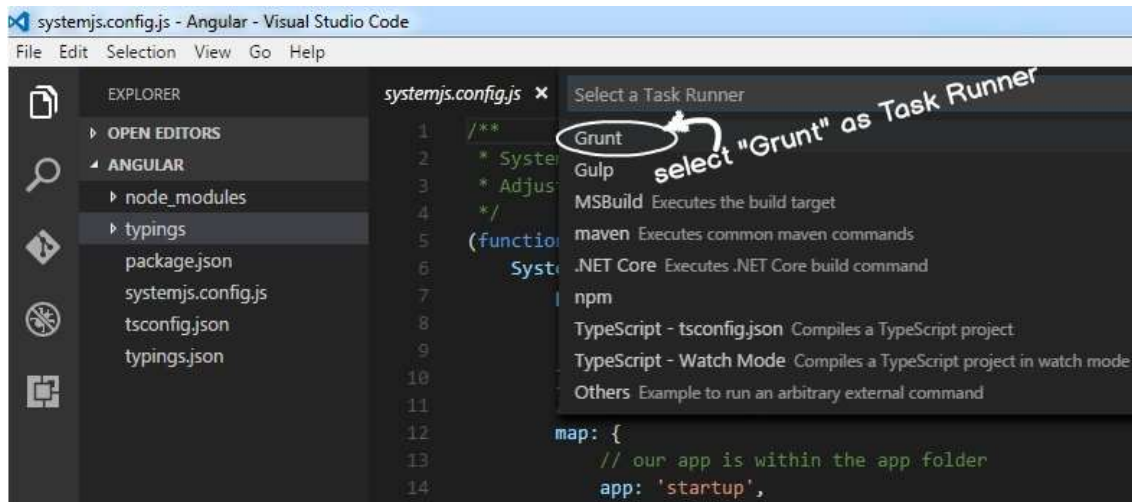
Step 3:- Configuring the task runner

VS Code is just a code editor. It has no idea how to compile typescript code, how to run TSC and so on. So we need to create a GRUNT task which will run TSC command and transpile typescript code to JavaScript.

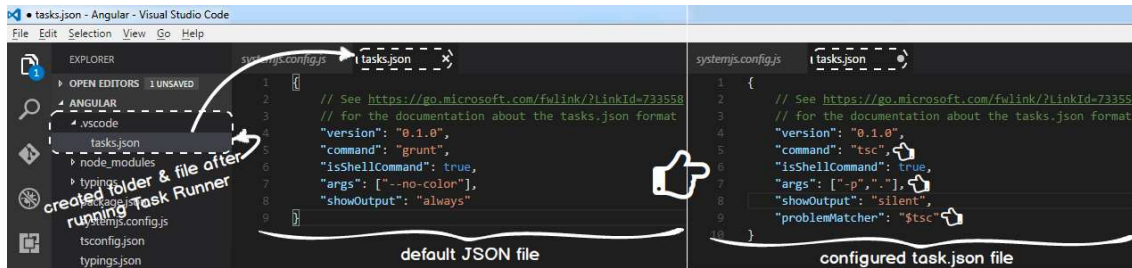
Press CONTROL + SHIFT + B and click on configure task runner as shown in the below figure.



VS Code then pops up lot of task runner options saying what kind of task is it, is it a GRUNT task, GULP task, MSBUILD, MAVEN etc. and so on.



Let's select GRUNT Task and paste the below JSON code. Once you paste it you will see it has created a file called as "tasks.json" file inside ".vscode" folder.



Below is the paste of "tasks.json" file. You can see in the command attribute we have given "tsc" as the command line, the "isShellCommand" specifies that this has to be executed in command line and 'args' specifies the argument.

When typescript will execute it will run using the configuration specified in the tasks.json file.

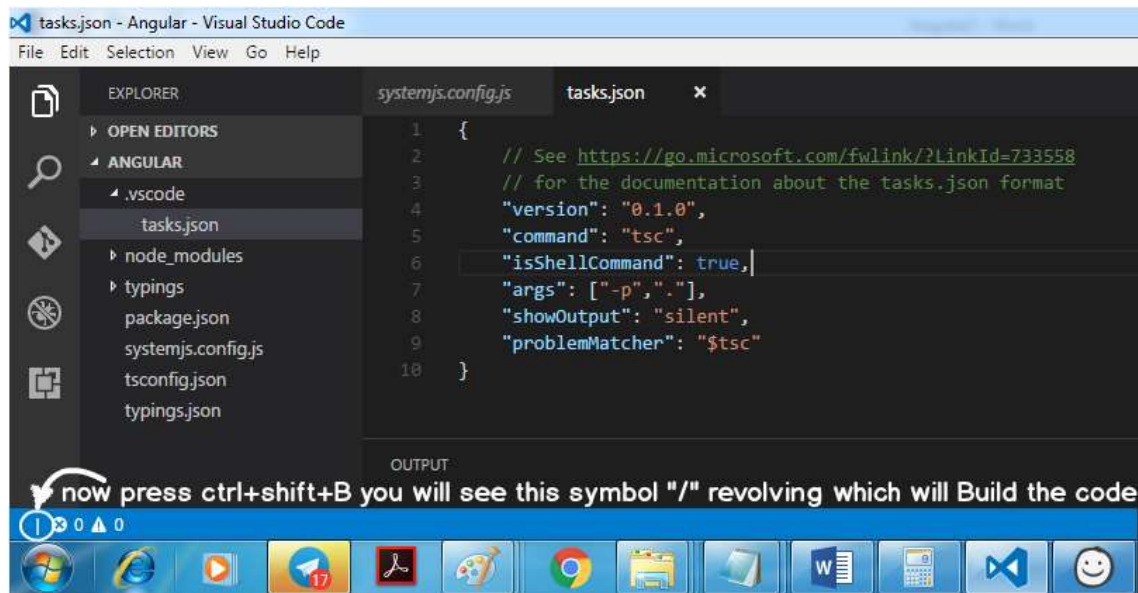
```

{

    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "0.1.0",
    "command": "tsc",
    "isShellCommand": true,
    "args": ["-p", "."],
    "showOutput": "silent",
    "problemMatcher": "$tsc"
}

```

So if you now press CONTROL + B it will build the TS files to JS file.

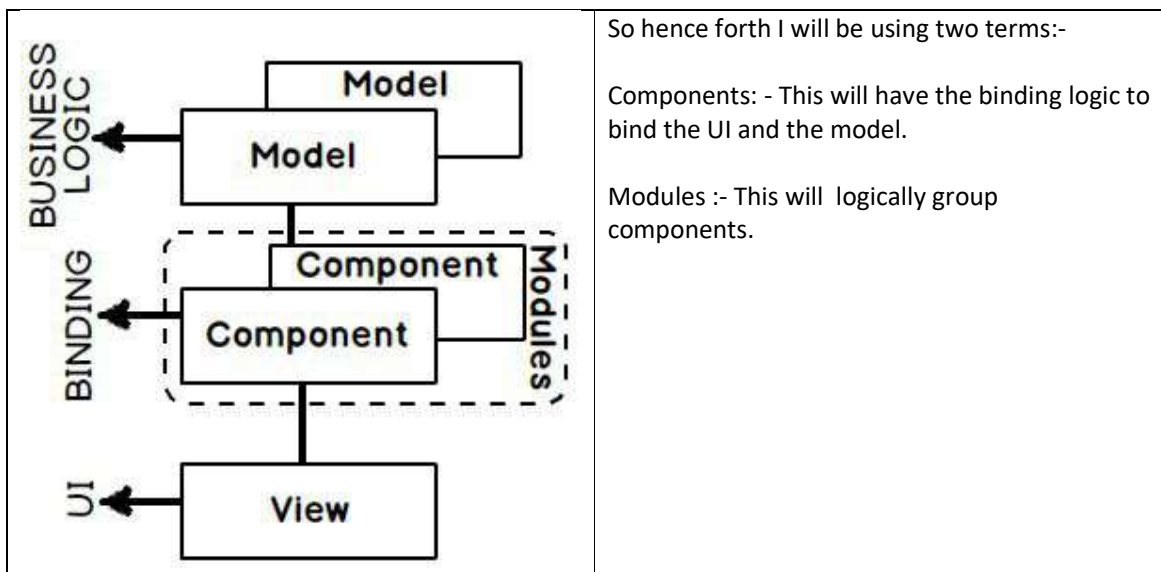


Understanding Angular 2 Component and module architecture

As we said in the previous section that the whole goal of Angular 2 is binding the model and the view. In Angular 2 the binding code is officially termed as “Component”. So hence forth we will use the word “Component” for the binding code.

In enterprise projects you can have lot of components. With many components it can become very difficult to handle the project.

So you can group components logically in to modules.

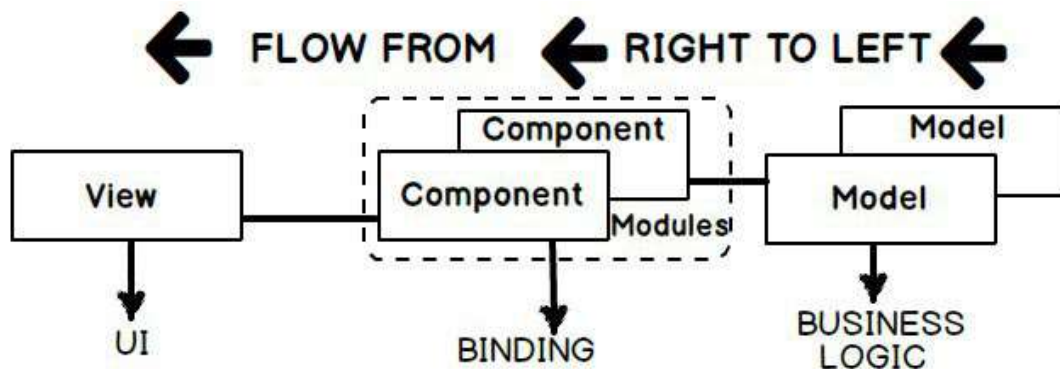


Step 4:- Following MVW Step by Step – Creating the folders

Before we start coding let's visualize the steps of coding. As we have said Angular 2 is a binding framework. It follows MVW architecture. It binds HTML UI with the JavaScript code (model).

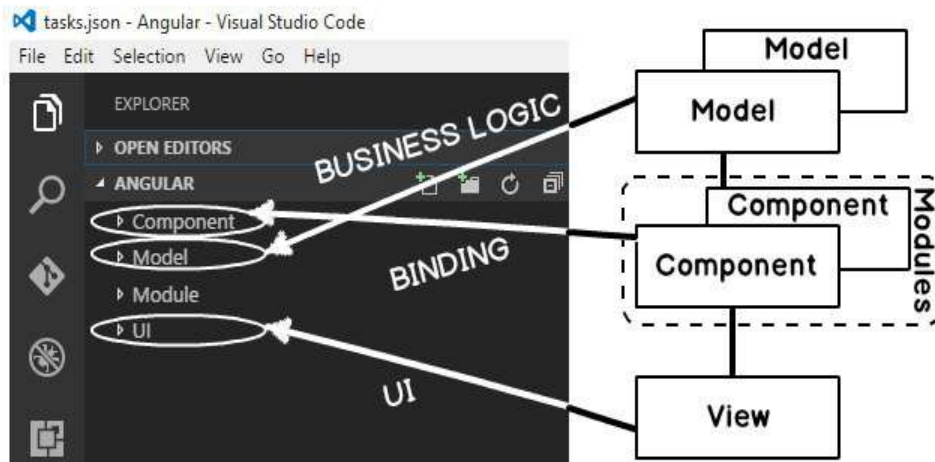
So if we visualize it will look something as shown in the image below. So let's move from right to left. So let's do the coding in the following sequence:-

1. Create the model.
2. Create the Component.
3. Create the module.
4. Create the HTML UI.

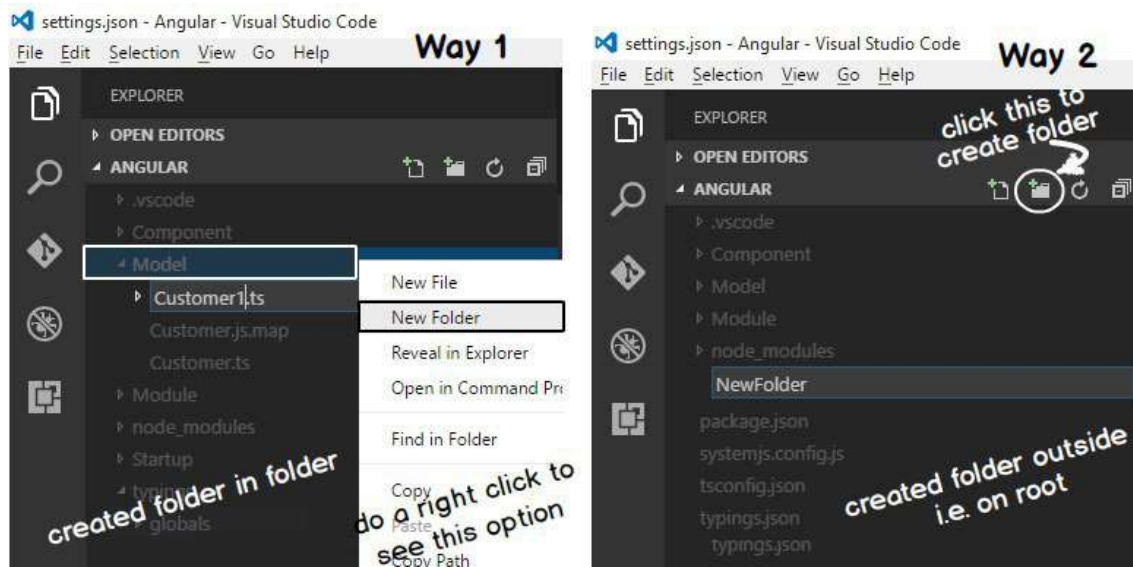


So let's first create four folders in our project:-

- View folder: - This folder will contain the HTML UI.
- Model folder: - This folder will have the main business typescript classes.
- Component folder: - This folder will have the binding code which binds the HTML UI and Model.
- Module: - This folder will have code which will logically group the components.



In order to create a folder in VS code you can use the "New folder" icon or you can right click and also create a folder.



Step 5:- Creating the model

Model > New File

do a right click to see this option

A model is nothing but a class with properties and behavior. So let us first create the customer model with three properties "CustomerName", "CustomerCode" and "CustomerAmount".

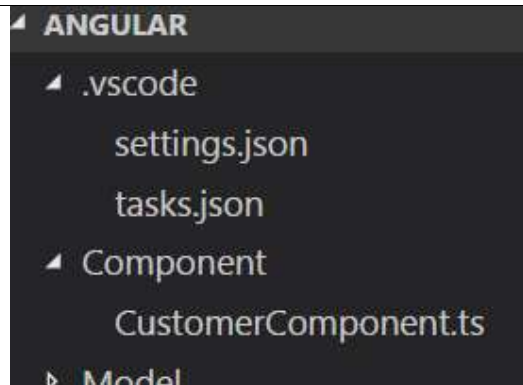
So right click on the "Model" folder and add a new file "Customer.ts". Keep the extension of this file as ".ts" as it's a typescript file.

While compiling the typescript command identifies only files with the extension ".ts".

In the “Customer.ts” let’s create a “Customer” class with three properties. In this book we will not be going through the basics of typescript , please do go through this [1 hour training video of typescript](#) which explains typescript in more detail.

```
export class Customer {  
    CustomerName: string = "";  
    CustomerCode: string = "";  
    CustomerAmount: number = 0;  
}
```

Step 6:- Creating the Component



The next thing we need to code is the binding code. Binding code in Angular 2 is represented by something termed as “COMPONENTS”. Angular 2 components has the logic which helps to bind the UI with the model.

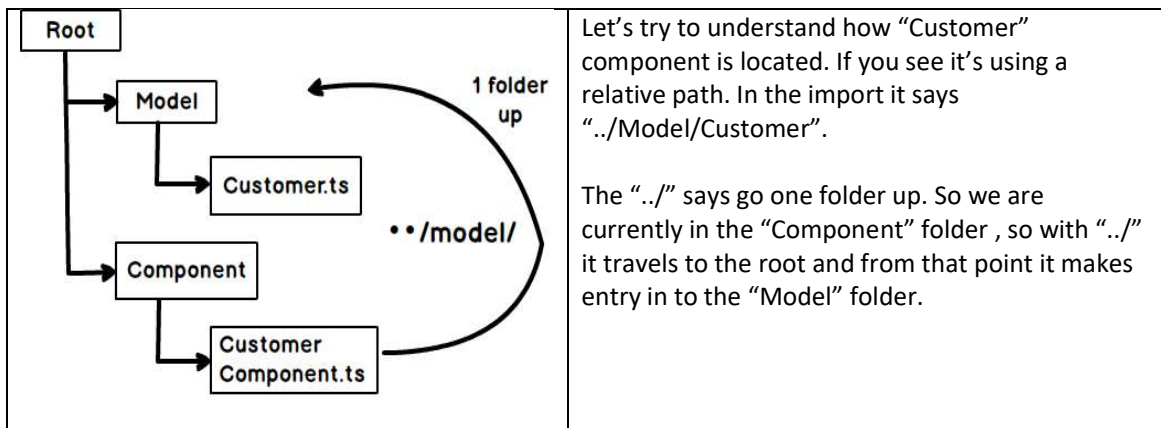
So right click on the component folder and add “CustomerComponent.ts” file as shown in the figure at the left.

In the component we need to import two things the Angular 2 core and our Customer model. Please note “import” is a typescript syntax and not JavaScript. So in case you are not following the code , please see this [Learn Typescript in 1 hour](#) video before moving ahead.

```
import {Customer} from '../Model/Customer'  
import {Component} from "@angular/core"
```

The first line imports the “Customer” class in to the “CustomerComponent.ts”. This import is only possible because we have written “export” in “Customer.ts” file. The import and export generate code which follows CommonJs , AMD or UMD specifications. In case you are new to these specifications please see this [CommonJs video](#) which explains the protocol in more detail.

```
import {Customer} from '../Model/Customer'
```



The next import command imports angular core components. In this we have not given any relative path using `"../"` etc. So how does typescript locate the angular core components ?.

```
import {Component} from "@angular/core"
```

If you remember we had used node to load Angular 2 and node loads the JS files in the `"node_modules"` folder. So how does typescript compiler automatically knows that it has to load the Angular 2 components from `"node_modules"` folder.

Typescript compiler uses the configuration from `"tsconfig.json"` file. In the configuration we have one property termed as `"moduleResolution"`. It has two values:-

- Classic :- In this mode typescript relies on `"./"` and `"../"` to locate folders.
- Node :- In this mode typescript first tries to locate components in `"node_modules"` folder and if not found then follows the `"../"` convention to traverse to the folders.

In our `tsconfig.json` we have defined the mode as `"node"` this makes typescript hunt modules automatically in `"node_modules"` folder. That makes the `"import"` of angular 2 components work.

```
{
  {
    ....
    ....
    "moduleResolution": "node",
    ....
    ....
  }
}
```

So now that both the import statements are applied let us create the `"CustomerComponent"` and from that let's expose `"Customer"` object to the UI with the object name `"CurrentCustomer"`.

```
export class CustomerComponent {
```

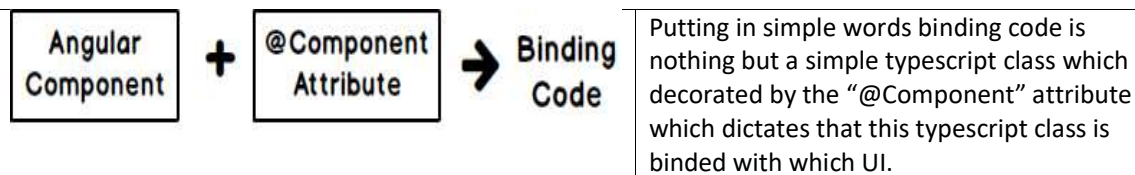
```
CurrentCustomer:Customer = new Customer();  
}
```

As we said previously that component connects / binds the model to the HTML UI. So there should be some code which tells that “CustomerComponent” is bounded with HTML UI. That’s done by something termed as “Component MetaData Attribute”. A component metadata attribute starts with “@Component” which has a “templateUrl” property which specifies the HTML UI with which the component class is tied up with.

```
@Component({  
  selector: "customer-ui",  
  templateUrl: "../UI/Customer.html"  
})
```

This attribute is then decorated on the top of the component. Below goes the full code.

```
@Component({  
  selector: "customer-ui",  
  templateUrl: "../UI/Customer.html"  
})  
export class CustomerComponent {  
  CurrentCustomer:Customer = new Customer();  
}
```



Below goes the full code of the Angular component.

```
// Import statements  
import {Component} from "@angular/core"  
import {Customer} from '../Model/Customer'  
  
// Attribute metadata  
@Component({  
  selector: "customer-ui",  
  templateUrl: "../UI/Customer.html"  
})  
  
// Customer component class exposing the customer model
```

```
export class CustomerComponent {
    CurrentCustomer:Customer = new Customer();
}
```

Step 7:- Creating the Customer HTML UI – Directives and Interpolation

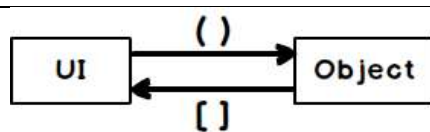
Now from the “CustomerComponent” , “Customer” is exposed via the “CurrentCustomer” object to UI. So in the HTML UI we need to refer this object while binding.

In the HTML UI the object is binded by using “Directives”. Directives are tags which direct how to bind with the UI.

For instance if we want to bind “CustomerName” with HTML textbox code goes something as shown below:-

- “[ngModel)” is a directive which will help us send data from the object to UI and vice versa.
- Look at the way binding is applied to the object. It’s referring the property as “CurrentCustomer.CustomerName” and not just “CustomerName”. Why ???. Because if you remember the object exposed from the “CustomerComponent” is “CurrentCustomer” object. So you need to qualify “CurrentCustomer.CustomerCode”.

```
<input type="text" [(ngModel)]="CurrentCustomer.CustomerName">
```



- Round brackets indicate data sent from UI to object.
- Square brackets indicate data is sent from object to UI.
- If both are present then it’s a two way binding.

There would be times when we would like to display object data on the browser. By using “{{” braces we can display object data with HTML tags. In the below HTML we are displaying “CustomerName” mixed with HTML BR tag. These braces are termed as “INTERPOLATION”. If you see the dictionary meaning of interpolation it means inserting something of different nature in to something else.

In the below code we are inserting object data within HTML.

```
{{CurrentCustomer.CustomerName}}<br />
```

Below goes the full HTML UI code with binding directives and interpolation.

```
<div>
Name:
<input type="text" [(ngModel)]="CurrentCustomer.CustomerName"><br /><br />
Code:
<input type="text" [(ngModel)]="CurrentCustomer.CustomerCode"><br /><br />
Amount:
```

```

<input type="text" [(ngModel)]="CurrentCustomer.CustomerAmount"><br /><br />
</div>
{{CurrentCustomer.CustomerName}}<br /><br />
{{CurrentCustomer.CustomerCode}}<br /><br />
{{CurrentCustomer.CustomerAmount}}<br /><br />

```

Step 8:- Creating the Module

Module is a container or you can say it's a logical grouping of components and other services.

So the first import in this module is the "CustomerComponent" component.

```
import { CustomerComponent } from '../Component/CustomerComponent';
```

We also need to import "BrowserModule" and "FormsModule" from core angular.

"BrowserModule" has components by which we can write IF conditions and FOR loop.

"FormsModule" provides directive functionality like "ngModel", expressions and so on.

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from "@angular/forms"
```

We also need to create a typescript class "MainModuleLibrary". At this moment this class does not have any code but it can have code which will provide component level logic like caching , initialization code for those group of components and so on.

```
export class MainModuleLibrary { }
```

To create a module we need to use import "NgModule" from angular core. This helps us to define module directives.

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

"NgModule" has three properties:-

- Imports: - If this module is utilizing other modules we define the modules in this section.
- Declarations: - In this section we define the components of the modules. For now we only have one component 'CustomerComponent'.
- Bootstrap: - This section defines the first component which will run. For example we can have "HomeComponent", "CustomerComponent" and so on. But the first component which will run is the "HomeComponent" so that we need to define in this section.

```
@NgModule({
  imports: [BrowserModule,
            FormsModule],
```

```

    declarations: [CustomerComponent],
    bootstrap: [CustomerComponent]
  })

```

Below goes the full code of Angular module which we discussed in this section.

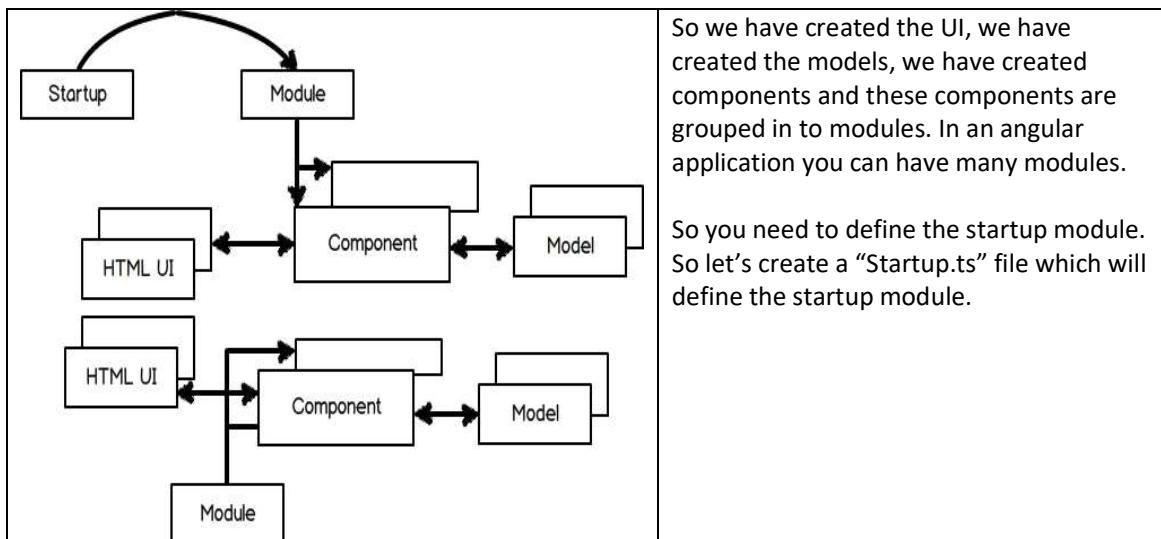
```

import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {FormsModule} from "@angular/forms"
import { CustomerComponent }  from '../Component/CustomerComponent';

@NgModule({
  imports: [BrowserModule,
            FormsModule],
  declarations: [CustomerComponent],
  bootstrap: [CustomerComponent]
})
export class MainModuleLibrary { }

```

Step 9:- Creating the “Startup.ts” file



Below goes the “Startup.ts” file in which we have defined which module will be bootstrapped.

```

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { MainModuleLibrary } from '../Module/MainModuleLibrary';

```

```
const platform = platformBrowserDynamic();
platform.bootstrapModule(MainModuleLibrary);
```

Step 10:- Invoking “Startup.ts” file using main angular page

So let us create a startup HTML page which will invoke the “Startup.ts”. Now in this page we will need to import four JavaScript framework files Shim , Zone , Meta-data and System JS as shown in the below code.

```
<script src="../../node_modules/core-js/client/shim.min.js"></script>
<script src="../../node_modules/zone.js/dist/zone.js"></script>
<script src="../../node_modules/reflect-metadata/Reflect.js"></script>
<script src="../../node_modules/systemjs/dist/system.src.js"></script>
```

Below are the use of JS files :-

Shim.min.js	This framework ensures that ES 6 javascript can run in old browsers.
Zone.js	This framework ensures us to treat group of Async activities as one zone.
Reflect.js	Helps us to apply meta-data on Javascript classes. We are currently using @NgModule and @NgComponent as attributes.
System.js	This module will helps to load JS files using module protocols like commonjs , AMD or UMD.

In this HTML page we will be calling the “systemjs.config.js” file. This file will tell system JS which files to be loaded in the browser.

```
<script src="../../systemjs.config.js"></script>
<script>
    System.config({
        "defaultJSExtensions": true
    });

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

In the “import” we need to specify “startup” which will invoke “startup.js” file.

```
System.import('startup').catch(function (err) { console.error(err); });
```

Our customer screen in with the name “Customer.html”. So to load in to this screen we need to define a place holder. So in this place holder our Customer HTML page will load.

```
<customer-ui></customer-ui>
```

If you remember when we created the component class we had said to load the HTML page in a selector. So that selector is nothing but a tag (placeholder) to load our Customer page.

```
@Component({  
  selector: "customer-ui",  
  templateUrl: "../UI/Customer.html"  
})
```

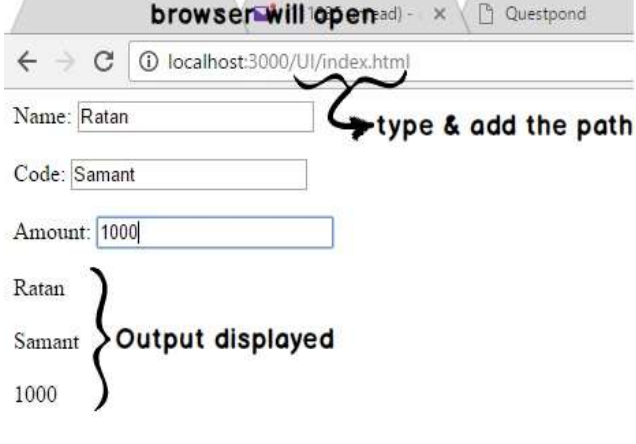
Below goes the full HTML page with all scripts and the place holder tag.

```
<!DOCTYPE html>  
<html>  
<head>  
  <title></title>  
  <meta charset="utf-8" />  
</head>  
<!-- 1. Load libraries -->  
<!-- Polyfill(s) for older browsers -->  
<script src="../../node_modules/core-js/client/shim.min.js"></script>  
<script src="../../node_modules/zone.js/dist/zone.js"></script>  
<script src="../../node_modules/reflect-metadata/Reflect.js"></script>  
<script src="../../node_modules/systemjs/dist/system.src.js"></script>  
<!-- 2. Configure SystemJS -->  
<script src="../../systemjs.config.js"></script>  
<script>  
  System.config({  
    "defaultJSExtensions": true  
  });  
  
  System.import('startup').catch(function (err) { console.error(err); });  
</script>  
<body>  
  <customer-ui></customer-ui>  
</body>  
</html>
```

Step 11:- Installing http-server and running the application

In order to run the application we need a web server. So go to integrated terminal and type “npm install http-server”. “http-server” is a simple, zero-configuration command-line http server which we can use for testing, local development and learning. For more details visit <https://www.npmjs.com/package/http-server>

<pre>C:\Users\user\Downloads\Telegram Desktop\Angular>http-server Starting up http-server, serving ./ Available on: http://192.168.1.4:8080 http://192.168.1.7:8080 http://192.168.15.1:8080 http://192.168.56.1:8080 http://10.71.34.1:8080 http://127.0.0.1:8080 Hit CTRL-C to stop the server</pre>	<p>To run this server we need to type “http” in the VS code integrated terminal as shown in the figure.</p> <p>In case your 80 port is blocked you can run this server on a specific port using “http-server -p 99”. This will run this server over 99 port.</p>
---	--

<p>browser will open</p>  <p>type & add the path</p> <p>Output displayed</p>	<p>So once the web server is running you can now browse to the main angular HTML page.</p> <p>Main angular page means the page in which we have put the scripts, put the place holder , systemjs and so on.</p> <p>Please note Customer.html is not the main page. This page will be loaded in the placeholder of main angular page.</p> <p>Once the sites are running type in one of the textboxes and see the automation of binding output in expression.</p>
--	---

How to the run the source code?

The source code that is attached in this book is without “node_modules” folder. So to run the code you need to open the folder using VS code and then do a NPM using the integrated terminal on the folder where you have “package.json” file. Please read step 2 again to understand how node works.

Lab 2 :- Implementing SPA using Angular 2 routing

Fundamental of Single page application

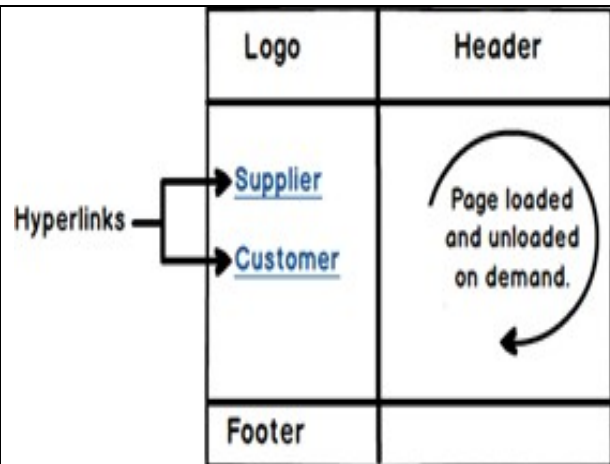
Now a days Single page application (SPA) has become the style of creating websites. In SPA we load only things which we need and nothing more than that.

At the right-hand side is a simple website where we have Logo and header at the top, Left menu links and footer at the bottom.

So the first time when user comes to the site all the sections of the master page will be loaded.

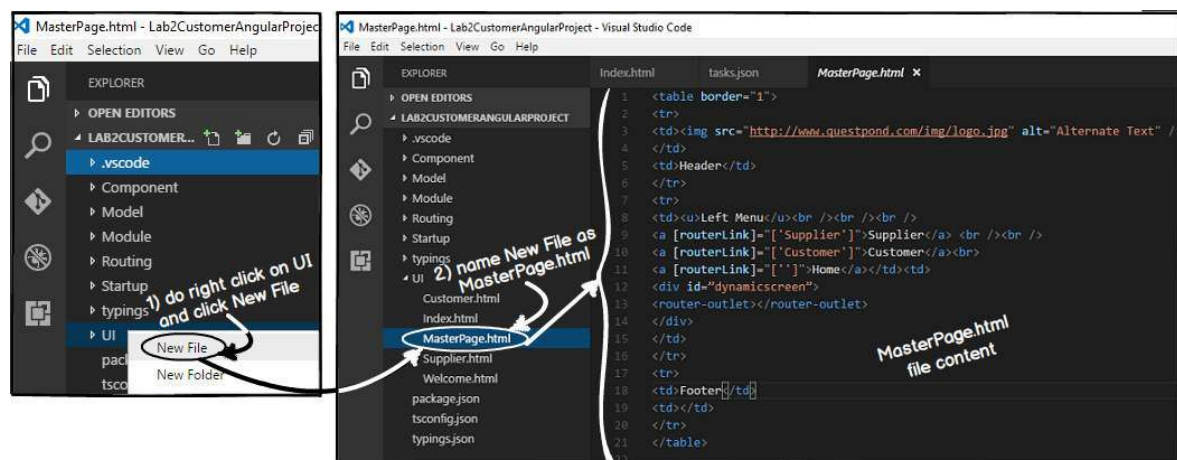
But when user clicks on Supplier link only Supplier page will load and not logo, header and footer again. When user clicks on Customer link only Customer page will be loaded and not all other sections.

Angular 2 routing helps us to achieve the same.



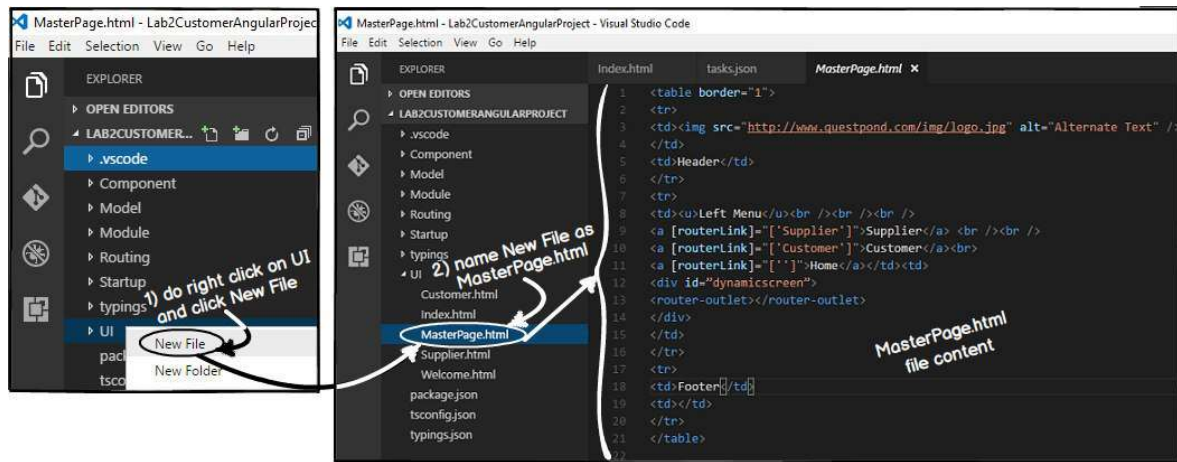
Step 1 :- Creating the Master Page

As everything revolves around the Master Page so the first logical step would be is to create the "MasterPage".



In this master page we will create placeholders for logo , header , menu , footer , copyright and so on. These sections will be loaded only once when the user browses the website first time. And in the later times only pages which are needed will be loaded on demand.

Below is the sample HTML code which has all the placeholder sections. You can also see in this code we have kept a “DIV” tag section in which we would loading the pages on-demand.



Below is the overall main sections of “MasterPage”. Please note the “DIV” section with name “dynamicscreen” where we intend to load screens dynamically. We will fill these sections later.

```
<table border="1">
<tr>
<td>Logo</td>
<td>Header</td>
</tr>
<tr>
<td>Left Menu</td>
<td>
<div id="dynamicscreen">
    Dynamic screen will be loaded here
</div>
</td>
</tr>
<tr>
<td>Footer</td>
<td>Copyright</td>
</tr>
</table>
```

Step 2:- Creating the Supplier page and welcome page

Let's create two more HTML UI one "Supplier" page and "Welcome" page. In both these HTML pages we are not doing much, we have just greeting messages.

Below is the supplier pages text.

```
This is the Suppliers page
```

Below is welcome pages text.

```
Welcome to the website
```

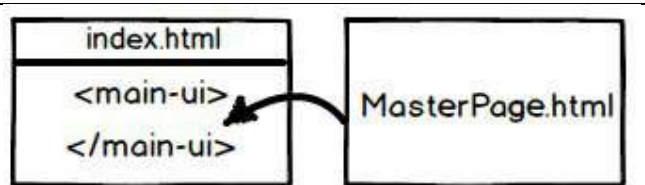
Step 3:- Renaming placeholder in Index.html

As explained in Part 1 "Index.html" is the startup page and it bootstraps all other pages using systemjs. In the previous lesson inside "Index.html", "Customer.html" page was loading. But now that we have master page so inside index page "MasterPage.html" will load.

So to make it more meaningful let's rename "customer-ui" tag to "main-ui". In this "main-ui" section we will load the master page and when end user clicks on the master page left menu links supplier, customer and welcome pages will load.

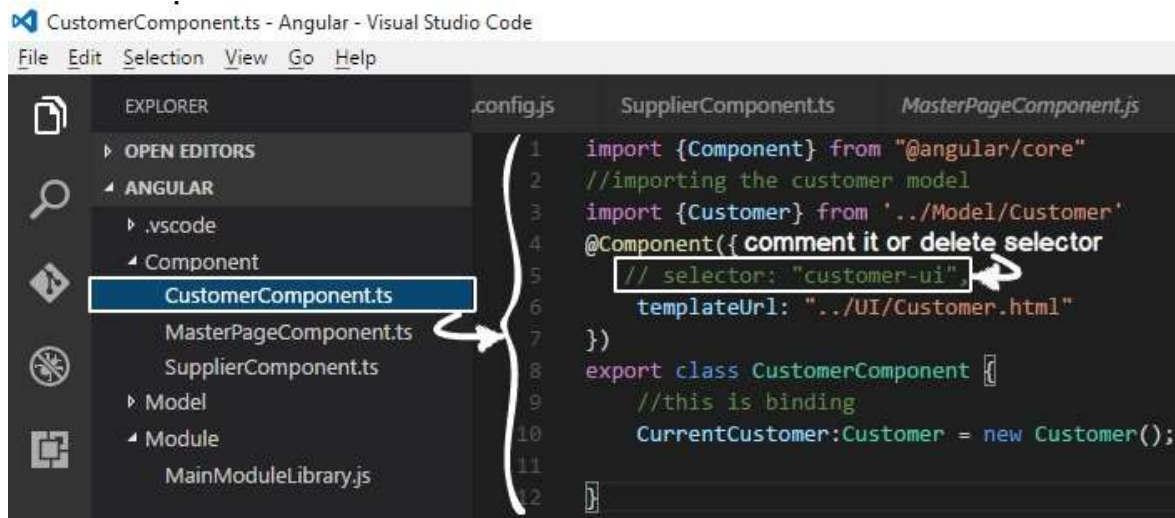
```
<body>
  <main-ui></main-ui>
</body>
```

So if look at the flow, first index.html will get loaded and then inside the "main-ui", "masterpage.html" gets loaded.



Step 4:- Removing selector from CustomerComponent

Now the first page to load in the index.html will be Masterpage and not Customer page. So we need to remove the selector from "CustomerComponent.ts". This selector will be moved to masterpage component in the later sections.



The final code of “CustomerComponent.ts” would look something as show below.

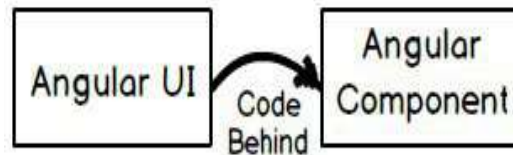
```
import {Component} from "@angular/core"
//importing the customer model
import {Customer} from '../Model/Customer'
@Component({
    templateUrl: "../UI/Customer.html"
})
export class CustomerComponent {
    //this is binding
    CurrentCustomer:Customer = new Customer();
}
```

Step 5:- Creating Components for Master , Supplier and Welcome page

Every UI which is Angular enabled should have component code file.
We have created 3 user interfaces so we need three component code files for the same.

In the component folder, we will create three component TS files “MasterPageComponent.ts” , “SupplierComponent.ts” and “WelcomeComponent.ts”.

You can visualize component code files as code behind for Angular UI.



So first let's start with "MasterPage.html" component which we have named as "MasterPageComponent.ts". This master page will get loaded in "Index.html" in the initial bootstrapping process. You can see in this component we have put the selector and this will be the only component which will have the selector.

```
import {Component} from "@angular/core"

@Component({
  selector: "main-ui",
  templateUrl: "../UI/MasterPage.html"
})
export class MasterPageComponent {
}
```

Below is the component code for "Supplier.html".

```
import {Component} from "@angular/core"

@Component({
  templateUrl: "../UI/Supplier.html"
})
export class SupplierComponent {
}
```

Below is the component code for "Welcome.html". Both Supplier and Welcome component do not have the selector, only the master page component has it as it will be the startup UI which will get loaded in index page.

```
import {Component} from "@angular/core"

@Component({
  templateUrl: "../UI/Welcome.html"
})
export class WelcomeComponent {
}
```

Step 6: - Creating the routing constant collection

Once the master page is loaded in the index page, end user will click on the master page links to browse to supplier page, customer page and so on. Now in order that the user can browse properly

we need to define the navigation paths. These paths will be specified in the “href” tags in the later steps.

When these paths will be browsed, it will invoke the components and components will load the UI. Below is a simple table with three columns. The first column specifies the path pattern, second which component will invoke when these paths are browsed and the final column specifies the UI which will be loaded.

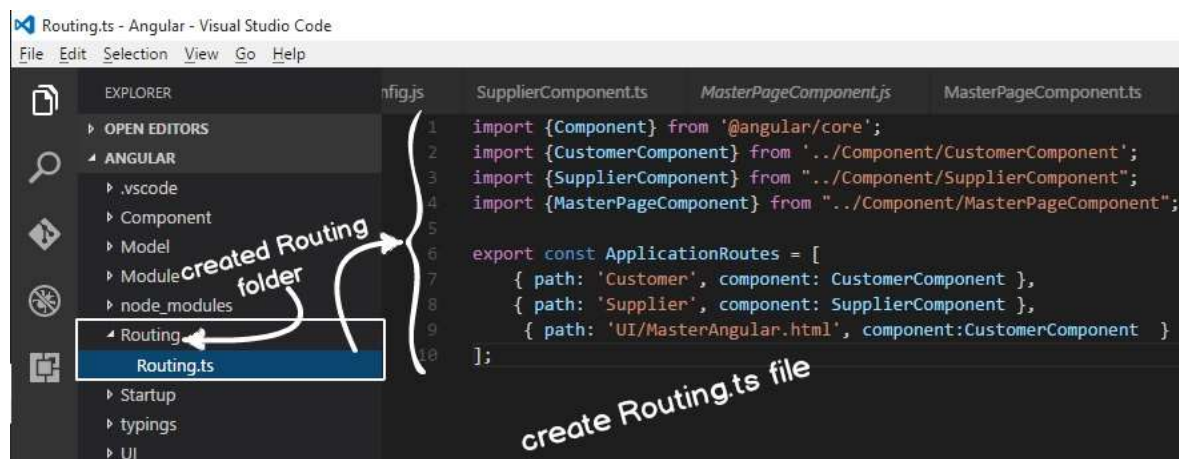
Path/URL	Component	UI which will be loaded
/	WelcomeComponent.ts	Welcome.html
/Customer	CustomerComponent.ts	Customer.html
/Supplier	SupplierComponent.ts	Supplier.html

The paths and component entries needs to be defined in a simple literal collection as shown in the below code. You can see the “ApplicationRoutes” is a simple collection where we have defined path and the component which will be invoked. These entries are made as per the table specified at the top.

```
import {Component} from '@angular/core';
import {CustomerComponent} from '../Component/CustomerComponent';
import {SupplierComponent} from "../Component/SupplierComponent";
import {WelcomeComponent} from "../Component/WelcomeComponent";

export const ApplicationRoutes = [
  { path: 'Customer', component: CustomerComponent },
  { path: 'Supplier', component: SupplierComponent },
  { path: '', component: WelcomeComponent }
];
```

As a good practice all the above code we have defined in a separate folder “routing” and in a separate file “routing.ts”.



Step 7: - Defining routerLink and router-outlet

The navigation (routes) defined in “Step 6” in the collection needs to be referred when we try to navigate inside the website. For example, in the master page we have defined the left menu hyperlinks.

So rather than using the “href” tag of HTML we need to use “[routerLink]”.

```
<a href="Supplier.html">Supplier</a>
```

We need to use “[routerLink]” and the value of “[routerLink]” will be the path specified in the routes collection defined in the previous step. For example in the “ApplicationRoutes” collection we have made one entry for Supplier path we need to specify the path in the anchor tag as shown in the below code.

```
<a [routerLink]="['Supplier']">Supplier</a>
```

When the end user clicks on the left master page links the pages (supplier page, customer page and welcome page) will get loaded inside the “div” tag. For that we need to define “router-outlet” placeholder. Inside this placeholder pages will load and unload dynamically.

```
<div id="dynamicscreen">
<router-outlet></router-outlet>
</div>
```

So if we update the master page defined in “Step 1” with “router-link” and “router-outlet” we would end up with code something as shown below.

```
<table border="1">
<tr>
<td>
</td>
<td>Header</td>
</tr>
<tr>
<td><u>Left Menu</u><br /><br /><br />
<a [routerLink]="['Supplier']">Supplier</a> <br /><br />
<a [routerLink]="['Customer']">Customer</a></td><td>
<div id="dynamicscreen">
<router-outlet></router-outlet>
</div>
</td>
</tr>
```

```

<tr>
<td>Footer</td>
<td></td>
</tr>
</table>

```

Step 8:- Loading the routing in Main modules

In order to enable routing collection paths defined in “ApplicationRoutes” we need to load that in the “MainModuleLibrary” as shown in the below code. “RouterModule.forRoot” helps load the application routes at the module level.

Once loaded at the module level it will be available to all the components for navigation purpose which is loaded inside this module.

```

@NgModule ({
  imports: [RouterModule.forRoot(ApplicationRoutes),
    BrowserModule,
    FormsModule],
  declarations:
[CustomerComponent,MasterPageComponent,SupplierComponent],
  bootstrap: [MasterPageComponent]
})
export class MainModuleLibrary { }

```

The complete code with routes would look something as shown below.

```

import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {FormsModule} from "@angular/forms"
import { CustomerComponent }    from '../Component/CustomerComponent';
import { SupplierComponent }    from '../Component/SupplierComponent';
import { MasterPageComponent }  from '../Component/MasterPageComponent';
import { RouterModule }    from '@angular/router';
import { ApplicationRoutes }  from '../Routing/Routing';

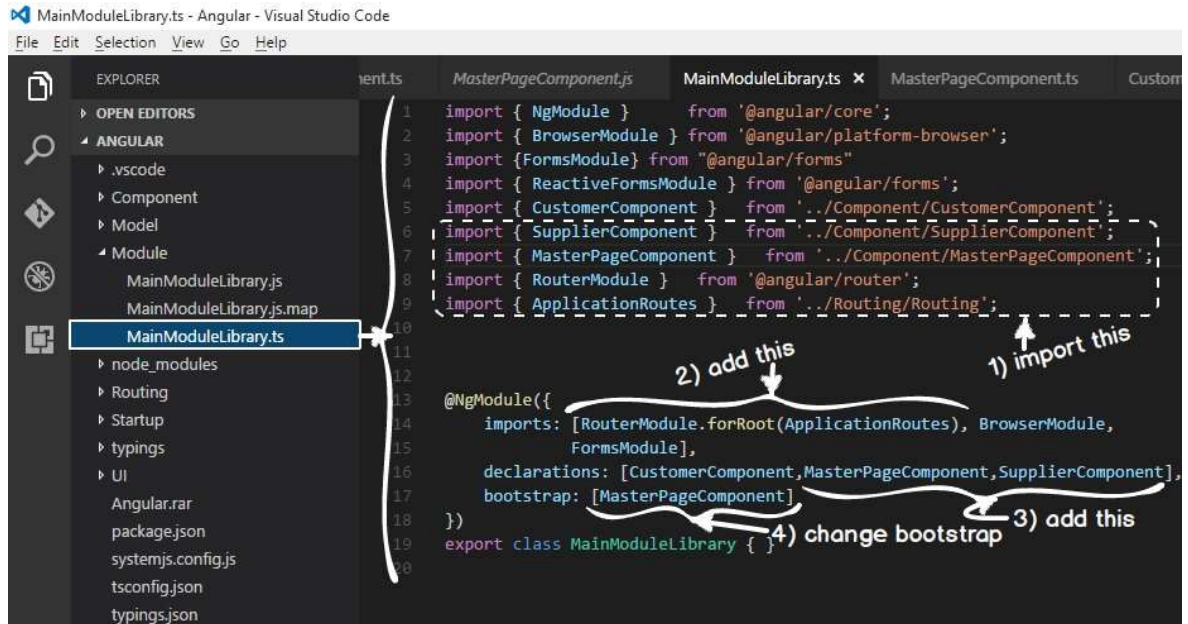
@NgModule({
  imports: [RouterModule.forRoot(ApplicationRoutes),
    BrowserModule,
    FormsModule],
  declarations:
[CustomerComponent,MasterPageComponent,SupplierComponent],

```

```

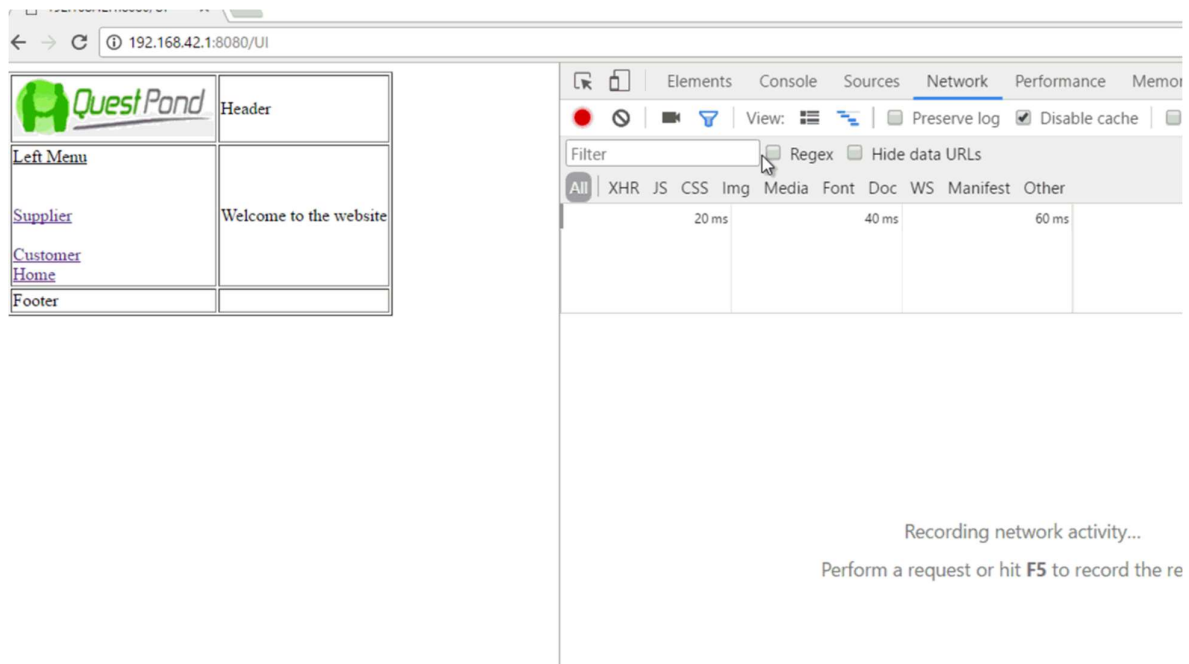
    bootstrap: [MasterPageComponent]
  })
  export class MainModuleLibrary { }

```



Step 9:- Seeing the output

Now run the website and try to browser to UI folder and you should see the below animated video output. You can see that the logo gets loaded only once and later when the user is click on the supplier links , customer links image is not loading again and again.



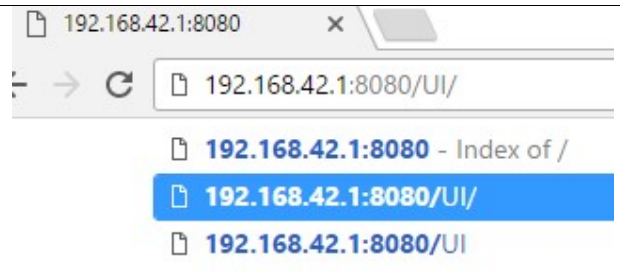
Step 10:- Fixing Cannot match any routes error

If you do a “f12” and check in the console part of the chrome browser , you would see the below error. Can you guess what the error is ?.

```
at zone.js:502
at ZoneDelegate.invokeTask (zone.js:265)
at Object.onInvokeTask (core.umd.js:6233)
at ZoneDelegate.invokeTask (zone.js:264)
✖ ▶ Unhandled Promise rejection: Cannot match any routes: 'UI' ; Zone: angular
; Task: Promise.then ; Value: Error: Cannot match any routes: 'UI'
at ApplyRedirects.noMatchError (router.umd.js:769)
at CatchSubscriber.eval [as selector] (router.umd.js:747)
at CatchSubscriber.error (catch.ts:58)
```

Your current angular application is route enabled. So every URL which is browsed is looked up in to routes collection. So the first URL which you browse is “/UI” and it tries to lookup in to your routes collection and does not find one.

So that’s why it throws up the above error.

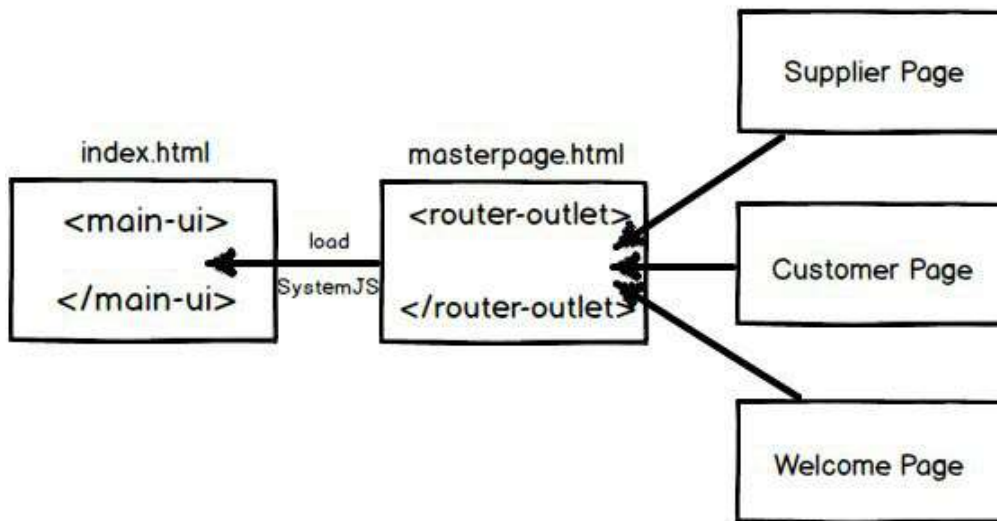


In order to fix the same make one more entry for “UI” path and point it to “WelcomeComponent”.

```
export const ApplicationRoutes = [
  { path: 'Customer', component: CustomerComponent },
  { path: 'Supplier', component: SupplierComponent },
  { path: '', component: WelcomeComponent },
  { path: 'UI', component: WelcomeComponent }
];
```

Understanding the flow

1. End user loads index.html page.
2. Index.html triggers systemjs and loads masterpage.html.
3. When end users click on masterpage links, other pages are loaded in “router-outlet” placeholders.



Lab 3 :- Handling Refresh (HashRouting)

