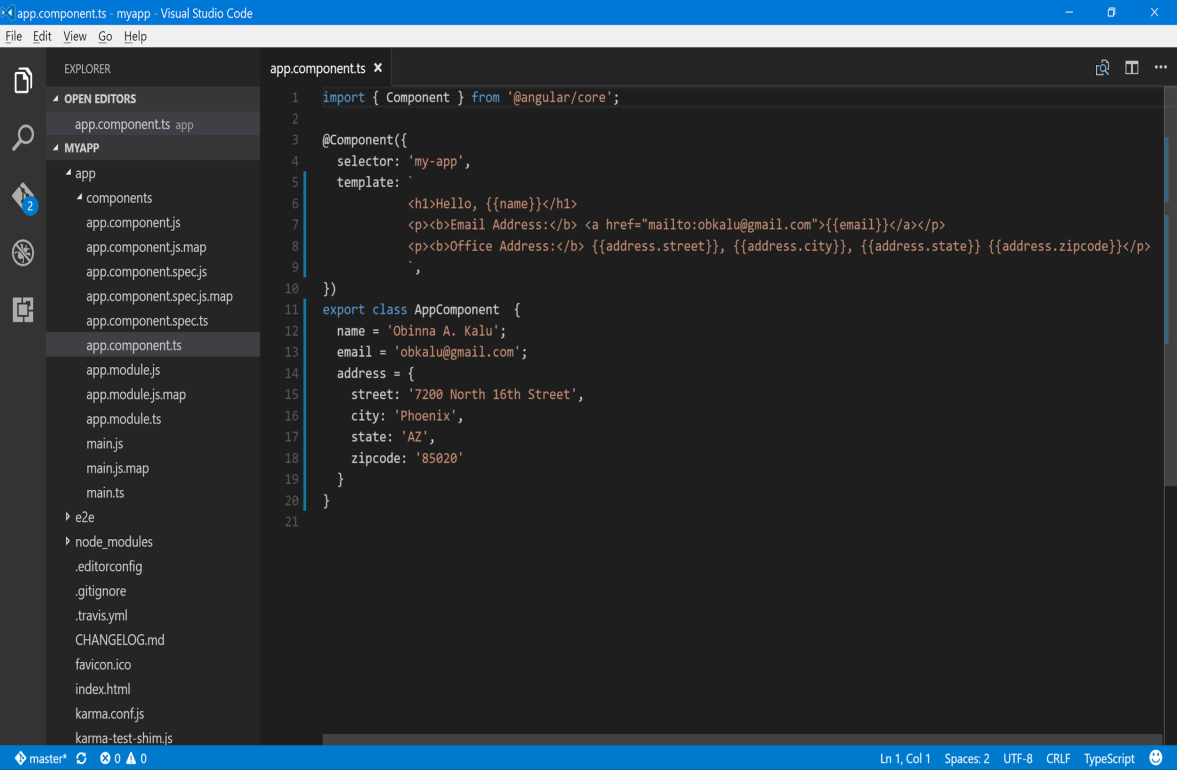
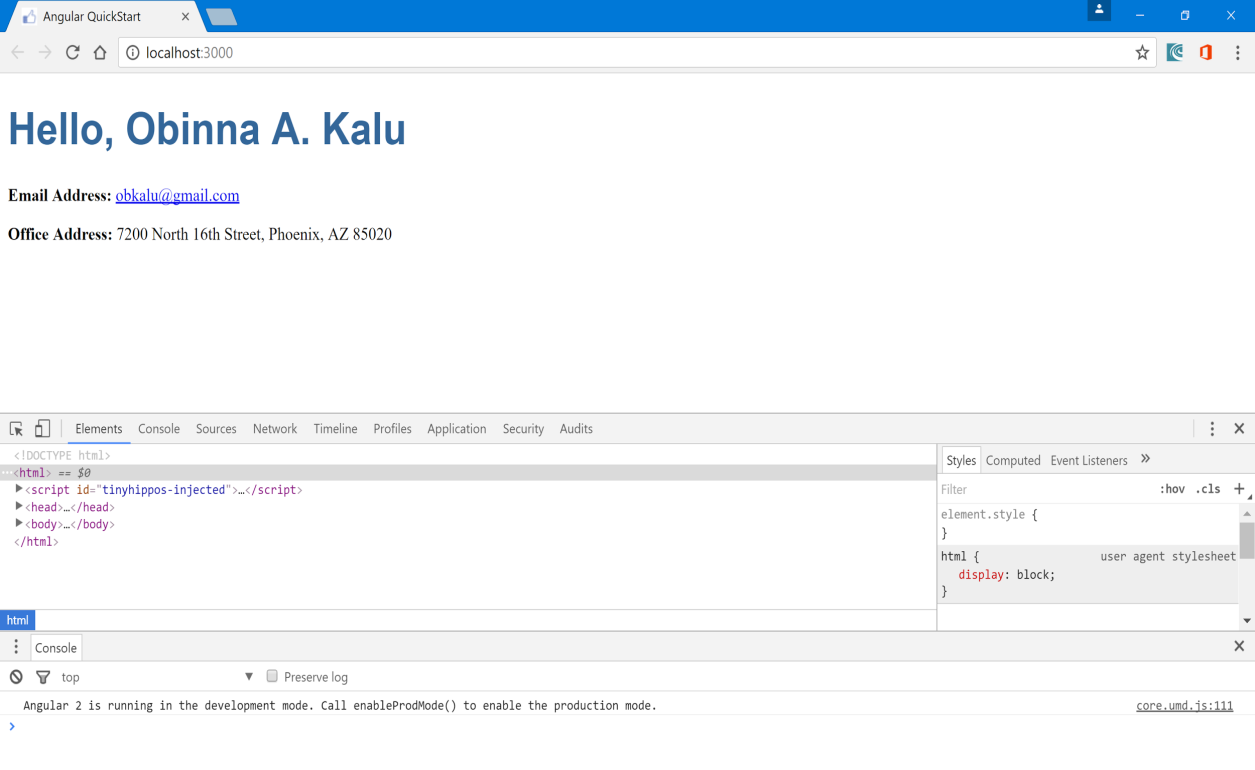
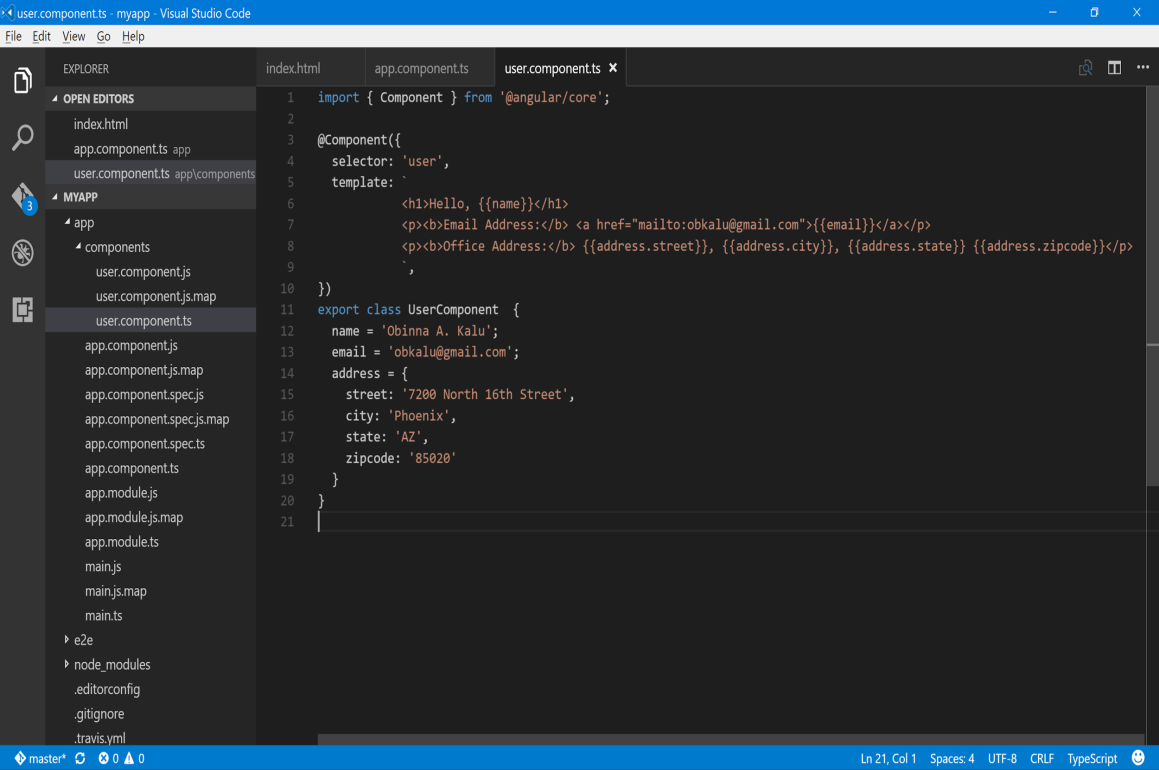
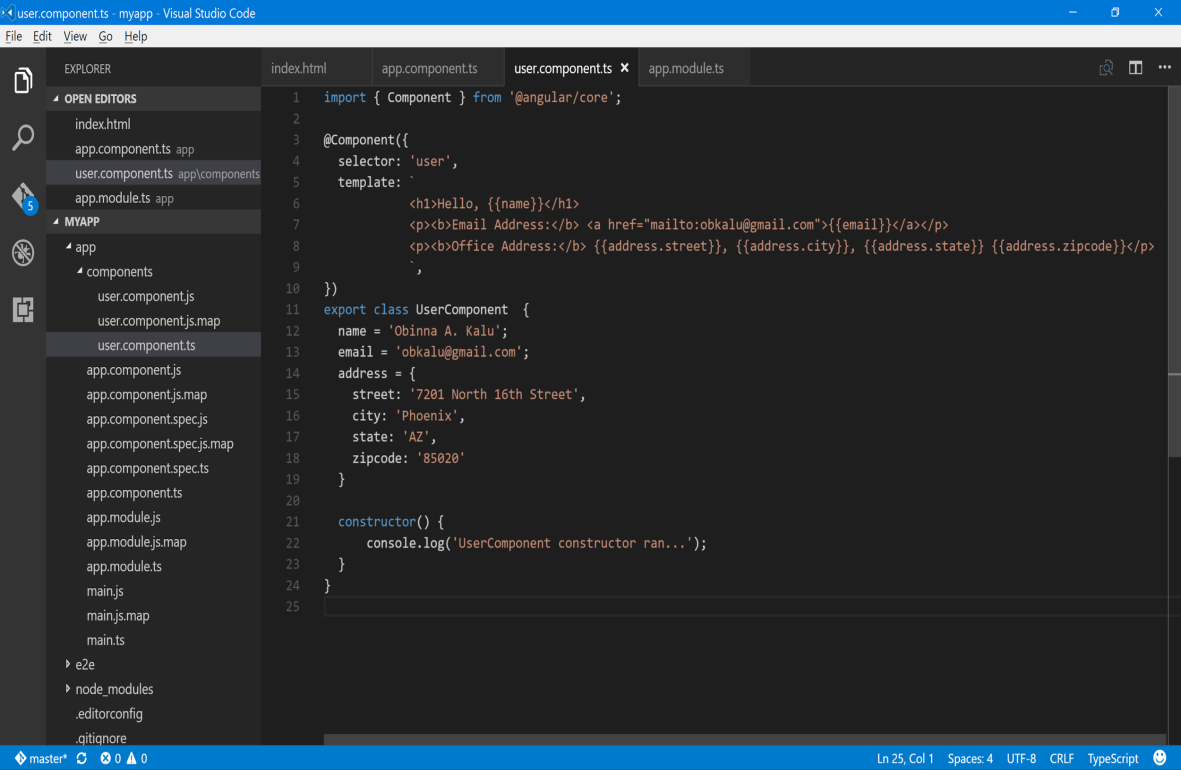
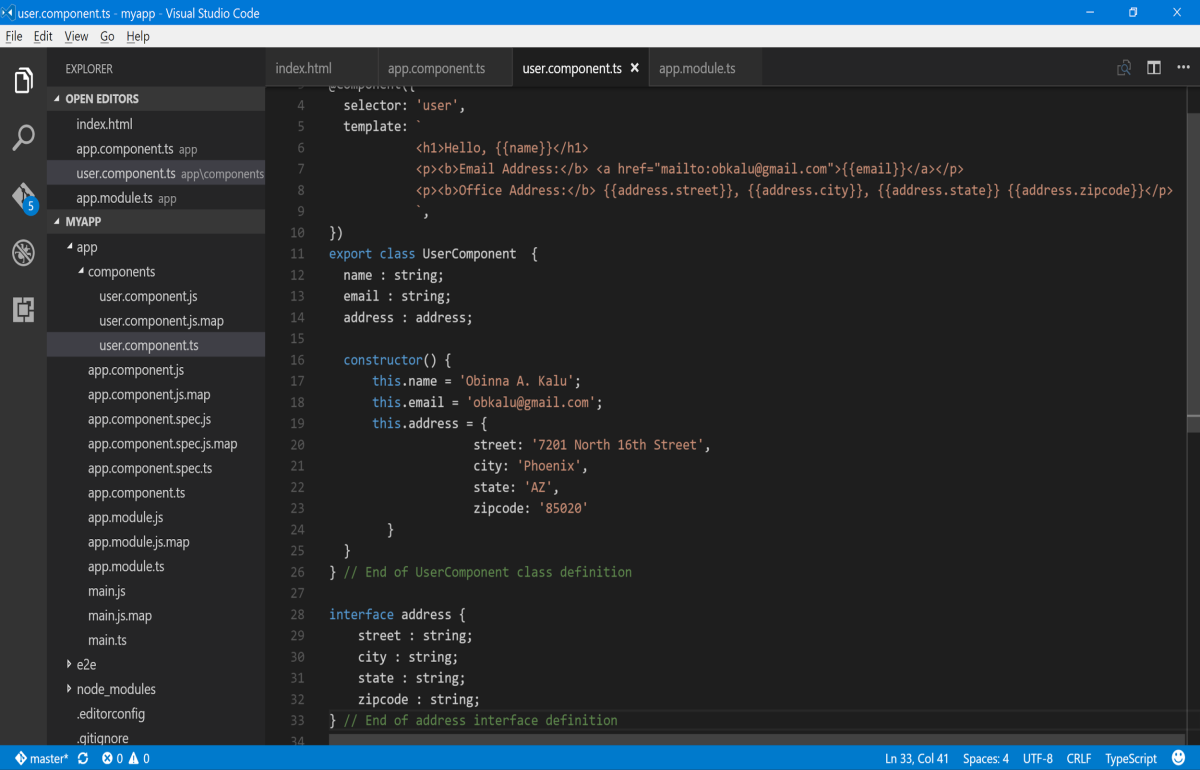
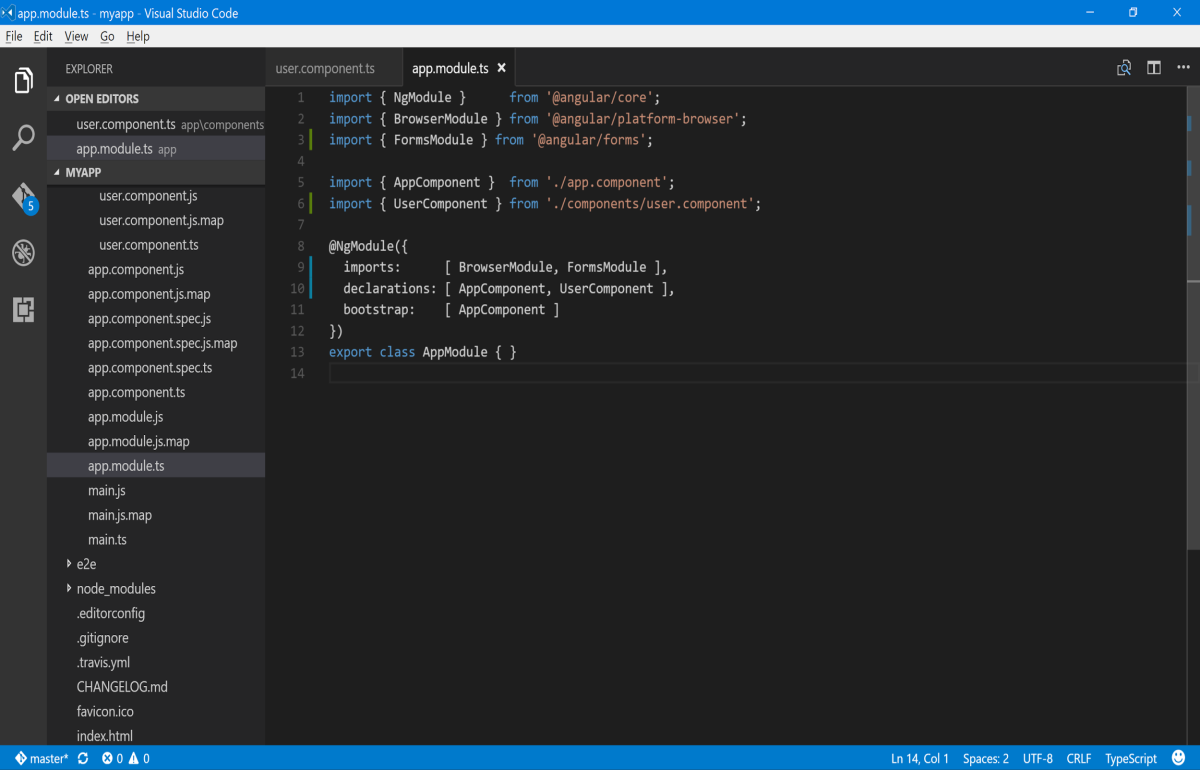
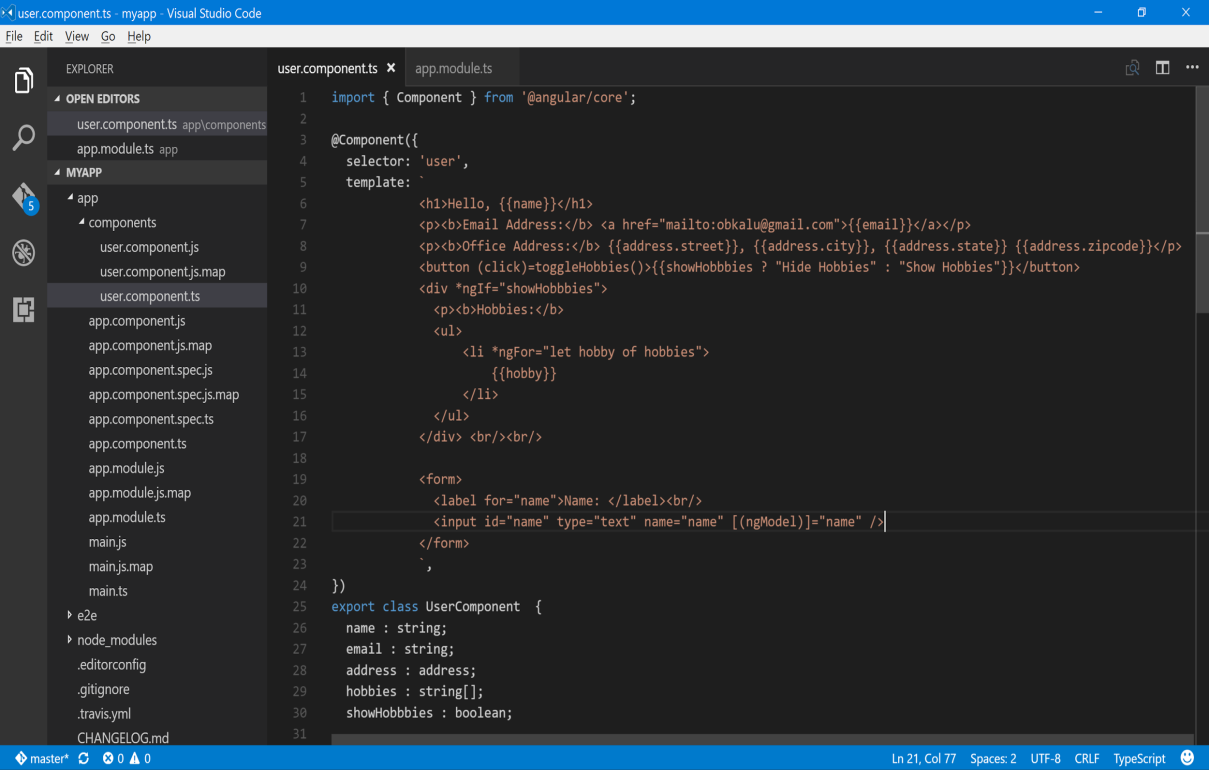
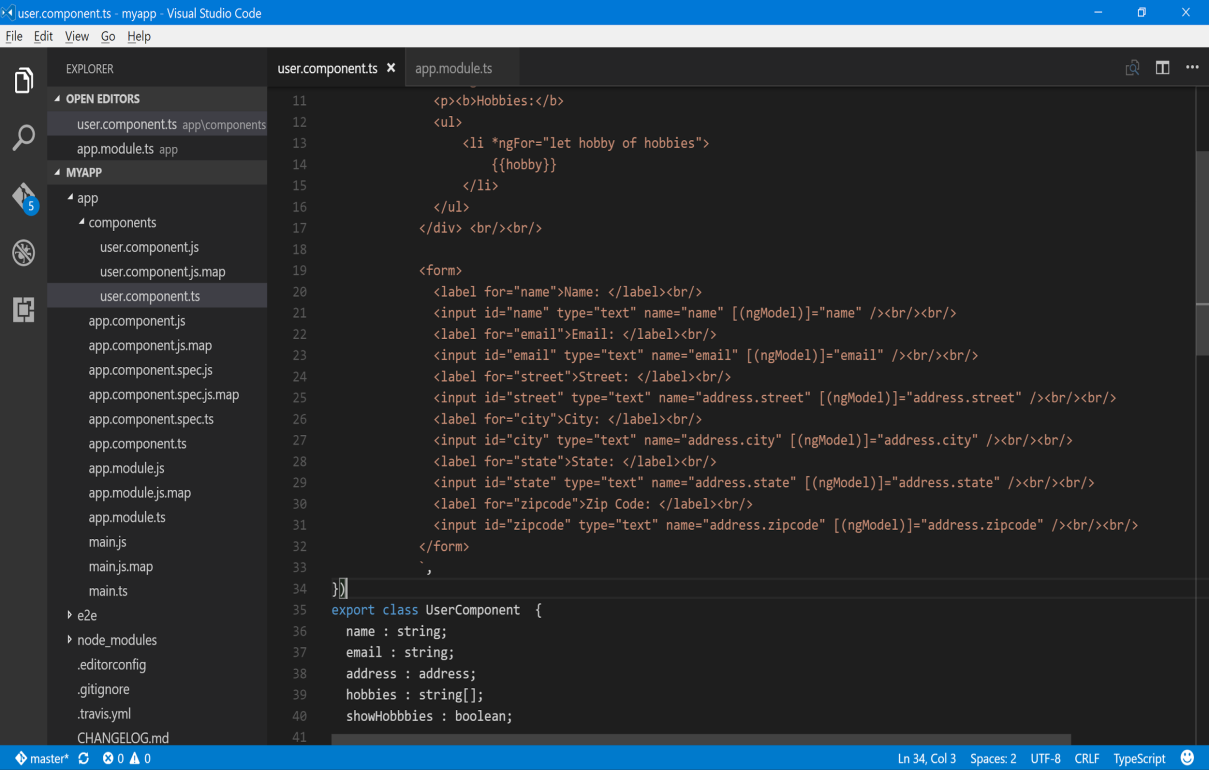
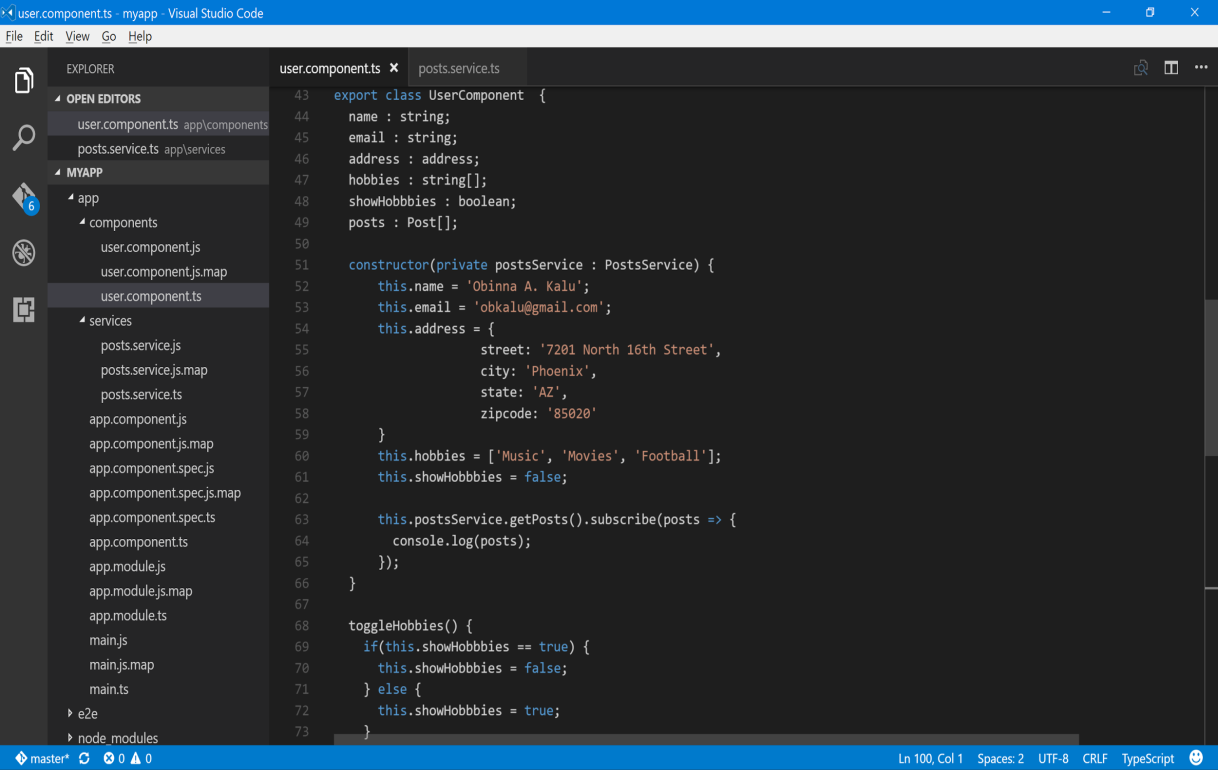
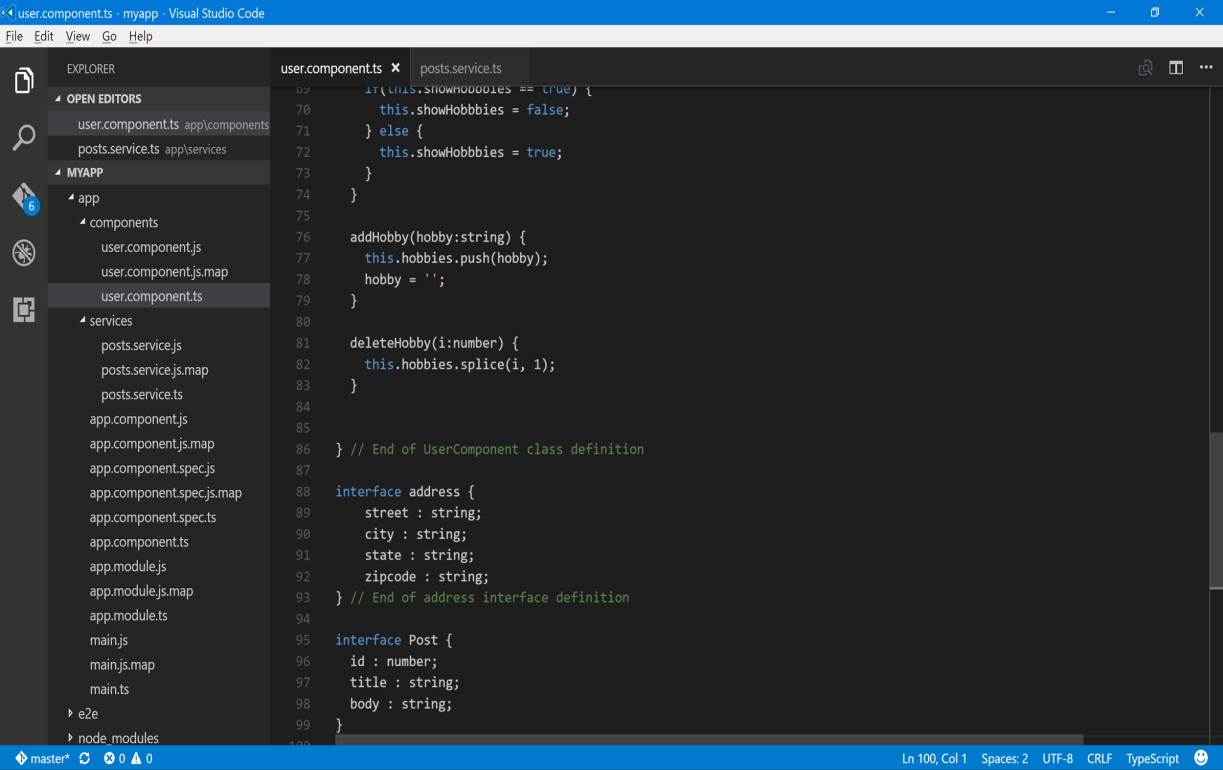
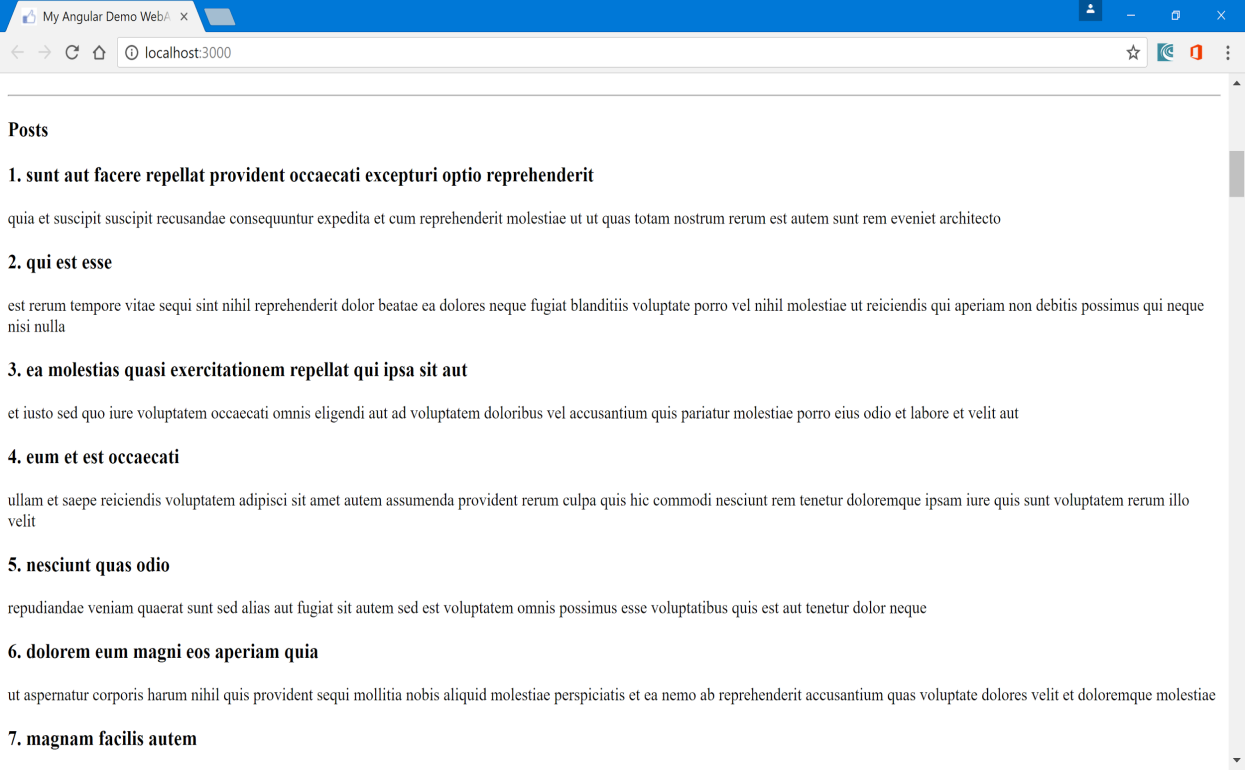
Angular 2 Crash Course

1. What is Angular 2?
   1. It is a client-side Javascript-based framework for developing Web applications.
   2. It supports development in Javascript (ECMAScript 5+), Typescript and Dart.
   3. It is modular and components-based.
2. How to install/setup Angular 2 for development - Options:
   1. From scratch, using npm (node package manager) which comes with Node.js.
   2. Using the QuickStart tutorial found on the Angular documentation pages online at angular.io. The angular quickstart is basically a prepackaged, ready-to-use angular project, which contains angular.js along with a predefined set of associated libraries and tools for developing an angular 2 webapp. You can obtain it by simply cloning the project from github at <https://github.com/angular/quickstart>, using gitbash or simply downloading the zip file version from the github site.
   3. Using the Angular CLI (see my other notes on how to do this): Note: this option also uses npm as in: c:\myangularproject>***npm install -g angular-cli***
3. Developing with Angular 2:
   1. Note: Use of Git is optional. Since we can choose to simply download the zip file version from github.
   2. First, we make a project folder, say, c:\>mkdir angular2project. Then, in Windows Explorer, we navigate to the newly created project folder, say, c:\angular2project, and right-click and select Git Bash Here. This opens a Git Bash command line prompt at the folder. Then we simply run the following git command at the prompt - $ git clone https://github.com/angular/quickstart myapp, which will clone the angular quickstart project from github into a sub-folder named ‘myapp’, inside our c:\angular2project folder. We then $cd into the myapp folder.
   3. Next, we launch open, our Visual Studio Code IDE and select from the menu, File -> Open Folder. And then, we navigate to the ‘myapp’ folder and load it up.
   4. The file named, package.json, found on the root of our myapp folder, contains all the dependencies (angularjs modules, libraries + others) that are defined for the project. We then install them by running the following command at the git bash cmd prompt - $ ***npm install.*** This will create a ‘node\_modules’ folder, inside our ‘myapp’ project folder and populate it with all the dependent packages/libraries.
   5. We can now start our dev server (i.e. lite-server) by running the command - $ ***npm start***. This will transpile (translate-compile) all of the app’s .ts typescript source files (e.g. myapp/app/app.component.ts) into .js javascript executable files, and then automatically launches a browser window and loads the app at url - <http://localhost:3000/>; displaying the text **Hello Angular** on the webpage.
   6. Edit the ‘app.component.ts’ source file to -
      1. import { Component } from '@angular/core';
      2. @Component({
      3. selector: 'my-app',
      4. template: `
      5. <h1>Hello, {{name}}</h1>
      6. <p><b>Email Address:</b> <a href="mailto:obkalu@gmail.com">{{email}}</a></p>
      7. <p><b>Office Address:</b> {{address.street}}, {{address.city}}, {{address.state}} {{address.zipcode}}</p>
      8. `,
      9. })
      10. export class AppComponent {
      11. name = 'Obinna A. Kalu';
      12. email = 'obkalu@gmail.com';
      13. address = {
      14. street: '7200 North 16th Street',
      15. city: 'Phoenix',
      16. state: 'AZ',
      17. zipcode: '85020'
      18. }
      19. }
      20. 
      21. What we have done is, simply define a set of named variables (using typescript code in the app.component.ts file) in the AppComponent class, and assigning them values, and then displaying these values in the component’s view template using interpolation (i.e. the double curly, {{}} syntax).
   7. 
   8. Now, instead of composing the app’s UI elements (components) all within the app.component.ts file, it is better to create sub-components in their own individual .ts files and then assemble them into one root component (hence, we say Angular 2 enables building a webapp’s UI by creating a component tree), the AppComponent, defined in the ‘app.component.ts’ file. To do this, we create a folder inside our app folder, named ‘components’.
   9. Inside the ‘components’ folder, we create a new file named, ‘user.component.ts’.
   10. In the ‘user.component.ts’ file, we simply copy over and paste the content of the ‘app.component.ts’ file and then edit the selector from ‘my-app’ to ‘user’, and change the class name from AppComponent to UserComponent. This should now read as:
       1. import { Component } from '@angular/core';
       2. @Component({
       3. selector: 'user',
       4. template: `
       5. <h1>Hello, {{name}}</h1>
       6. <p><b>Email Address:</b> <a href="mailto:obkalu@gmail.com">{{email}}</a></p>
       7. <p><b>Office Address:</b> {{address.street}}, {{address.city}}, {{address.state}} {{address.zipcode}}</p>
       8. `,
       9. })
       10. export class UserComponent {
       11. name = 'Obinna A. Kalu';
       12. email = 'obkalu@gmail.com';
       13. address = {
       14. street: '7200 North 16th Street',
       15. city: 'Phoenix',
       16. state: 'AZ',
       17. zipcode: '85020'
       18. }
       19. }
       20. 
       21. -
       22. We then update the ‘app.module.ts’ file by adding an import statement for the new ‘UserComponent’ class we have created and also add it to the declarations attribute of the @NgModule decorator. i.e.
       23. import { NgModule } from '@angular/core';
       24. import { BrowserModule } from '@angular/platform-browser';
       25. import { AppComponent } from './app.component';
       26. ***import { UserComponent } from './components/user.component';***
       27. @NgModule({
       28. imports: [ BrowserModule ],
       29. declarations: [ AppComponent***, UserComponent*** ],
       30. bootstrap: [ AppComponent ]
       31. })
       32. export class AppModule { }
       33. Then, we go back to the ‘app.component.ts’ file and simply remove all the html markup from the component’s view template, replacing it with just the element, <user></user>. And also remove the body of the AppComponent class definition. I.e. delete all the variables and values there, to have just an empty class declaration. I.e.
       34. import { Component } from '@angular/core';
       35. @Component({
       36. selector: 'my-app',
       37. template: `
       38. <user></user>
       39. `,
       40. })
       41. export class AppComponent { }
       42. Next, we add a constructor to our ‘UserComponent’ class definition, which will run each time the component is created (i.e. whenever it is loaded/displayed on the browser by/via the AppComponent. This will be:
       43. import { Component } from '@angular/core';
       44. @Component({
       45. selector: 'user',
       46. template: `
       47. <h1>Hello, {{name}}</h1>
       48. <p><b>Email Address:</b> <a href="mailto:obkalu@gmail.com">{{email}}</a></p>
       49. <p><b>Office Address:</b> {{address.street}}, {{address.city}}, {{address.state}} {{address.zipcode}}</p>
       50. `,
       51. })
       52. export class UserComponent {
       53. name = 'Obinna A. Kalu';
       54. email = 'obkalu@gmail.com';
       55. address = {
       56. street: '7201 North 16th Street',
       57. city: 'Phoenix',
       58. state: 'AZ',
       59. zipcode: '85020'
       60. }
       61. ***constructor() {***
       62. ***console.log('UserComponent constructor ran...');***
       63. ***}***
       64. }
       65. 
       66. -
       67. Next, we modify our UserComponent class code, so that variables are declared with their appropriate typescript data types (i.e. strongly-typed variables) and then the values are assigned within the constructor. This is as:
       68. 
       69. -
       70. Next, we add a form with a text input element and add 2-way data-binding to the ‘name : string’ property of the UserComponent class. We do this using, the [(ngModel)] = “name” in the <input... element. And for ngModel to work, we need to add an import for the FormsModule from ‘@angular/forms’ package, inside the app.module.ts file. And also, add FormsModule to the imports attribute of the NgModule decorator of the AppModule class.
       71. 
       72. -
       73. 
       74. -
       75. We now add further form fields for the rest of our UserComponent data properties. I.e. email, address.street, address.city, address.state etc. and bind each to their respective variables using ngModel’s 2-way binding. I.e
       76. -
       77. 
       78. -
       79. Another way of binding data to UI element is such as -
       80. <form (submit)="addHobby(hobby.value)">
       81. <label for="hobby">Add Hobby: </label><br/>
       82. ***<input id="hobby" type="text" #hobby />***
       83. </form>
       84. -
       85. Next, we implement a service and use it to fetch some external data from a remote api e.g. from the free rest api found at <http://jsonplaceholder.typicode.com/>
       86. To do this, we create a new folder named, services inside our app folder. And then create a new file named, posts.service.ts inside the services folder.
       87. In the ‘posts.service.ts’ file, we begin coding by doing the following:
       88. First, we import { Injectable } module from ‘@angular/core’ package.
       89. Next, we import { Http } module from ‘@angular/http’ package.
       90. In doing our json objects/data transfer, we are going to be working with Observables. So, we will need the map function from the reactive extensions js package. I.e. We need to, import ‘rxjs/add/operator/map’.
       91. We then declare our PostsService class as follows:
       92. @Injectable()
       93. export class PostsService {
       94. //
       95. }
       96. Inside the class, we add a constructor, as follows:
       97. @Injectable()
       98. export class PostsService {
       99. //
       100. constructor() {
       101. console.log(‘PostsService is initialized…’);
       102. }
       103. }
       104. In order to use the Http module, we need to inject it as a dependency inside the PostsService’s class constructor, as follows:
       105. @Injectable()
       106. export class PostsService {
       107. //
       108. constructor(private http : Http) {
       109. console.log(‘PostsService is initialized…’);
       110. }
       111. }
       112. Now, to be able to use the service inside the UserComponent, we import it there as - import { PostsService } from '../services/posts.service';
       113. Next, inside the @Component decorator in the UserComponent, we add the PostsService as a provider, by adding a ‘providers’ array attribute and setting its value as [PostsService] i.e.
       114. @Component({
       115. selector: 'user',
       116. template: `
       117. <h1>{{name}}</h1>
       118. <p><b>
       119. …
       120. </form>
       121. `,
       122. ***providers: [PostsService]***
       123. })
       124. export class UserComponent {
       125. …
       126. }
       127. We then add the HttpModule into the ‘app.module.ts’ file, by adding an import statement and also adding it in the imports array. i.e
       128. ...
       129. ***import { HttpModule } from '@angular/http';***
       130. @NgModule({
       131. imports: [ BrowserModule, FormsModule, ***HttpModule*** ],
       132. declarations: [ AppComponent, UserComponent ],
       133. bootstrap: [ AppComponent ]
       134. })
       135. export class AppModule { }
       136. Next, in the PostsService class inside the ‘posts.service.ts’, we then add a getPosts() function as thus:
       138. ***getPosts() {***
       139. ***return this.http.get('http://jsonplaceholder.typicode.com/posts')***
       140. ***.map(res => res.json());***
       141. ***}***
       142. Next, what we do is to call this, getPosts() function from inside our UserComponent class, in ‘user.component.ts’ file, as thus:
       143. First, in order to use the PostsService, we inject it into the constructor of the UserComponent, as thus:
       144. constructor(private postsService : PostsService) {
       145. this.name = 'Obinna A. Kalu';
       146. …
       147. And then, at the end inside the constructor, we add the following call to the getPosts() function of the PostsService:
       148. Note that: this.postsService.getPosts() will give us an Observable json object; so we need to subscribe to it, as thus:
       149. export class UserComponent {
       150. name : string;
       151. ...
       152. constructor(***private postsService : PostsService***) {
       153. this.name = 'Obinna A. Kalu';
       154. this.email = 'obkalu@gmail.com';
       155. this.address = {
       156. street: '7201 North 16th Street',
       157. city: 'Phoenix',
       158. state: 'AZ',
       159. zipcode: '85020'
       160. }
       161. this.hobbies = ['Music', 'Movies', 'Football'];
       162. this.showHobbbies = false;
       163. ***this.postsService.getPosts().subscribe(posts => {***
       164. ***console.log(posts);***
       165. ***});***
       166. }
       168. On saving the ‘user.component.ts’ file, the browser refreshes and in our console, we will see all 100 posts retrieved from the remote API.
       169. Next, in the UserComponent class, we add a variable for the posts; and also define an interface named Post to be the data type, as thus:
       171. posts : Post[ ];
       172. interface Post {
       173. id : number;
       174. title : string;
       175. body : string;
       176. }
       177. -
       178. 
       179. -
       180. 
       181. -
       182. We then modify our call to the getPosts() code inside our constructor, to be:
       183. this.postsService.getPosts().subscribe(posts => {
       184. this.posts = posts;
       185. });
       186. I.e. sets the this.posts variable to the posts data collection returned by the service call.
       187. So, now we are able to access and present/display the posts collection in our view template, in however way we choose to. Such as:
       188. <hr />
       189. <h3>Posts</h3>
       190. <div \*ngFor="let post of posts; let i=index">
       191. <h3>{{i+1}}. {{post.title}}</h3>
       192. <p>{{post.body}}</p>
       193. </div>
       194. Result: screenshot
       195. 
       196. -
       197. Now, our ‘user.component.ts’ file is looking very cluttered, since it contains both typescript code + all the view template html markup. To fix this, we create a new file, named ‘user.component.html’ inside the app/components folder. And then remove all the view template html markup code over to the new file and change the template attribute of the @Component decorator to be, templateUrl, and assign it the value, ‘user.component.html’.
       198. Next, let’s look at routing in Angular. To do this, we first create a new file in components folder, named ‘about.component.ts’.
       199. We then add the following code to it:
       200. import { Component } from '@angular/core';
       201. @Component({
       202. selector: 'about',
       203. template: '<h1>About This App</h1>'
       204. })
       205. export class AboutComponent { }
       206. We then include it in ‘app.module.ts’, by doing:
       207. …
       208. ***import { AboutComponent } from './components/about.component';***
       209. @NgModule({
       210. imports: [ BrowserModule, FormsModule, HttpModule ],
       211. declarations: [ AppComponent, UserComponent, ***AboutComponent*** ],
       212. bootstrap: [ AppComponent ]
       213. })
       214. export class AppModule { }
       215. To do routing in angular, the intent here is to be able to navigate to <http://localhost:3000/about> and have the app take us to the AboutComponent’s page.
       216. To do this, we create a new file in the ‘app’ folder, named ‘app.routing.ts’.
       217. In this file, we start coding by importing the following couple of angular modules:
       218. ***import { ModuleWithProviders } from '@angular/core';***
       219. ***import { Routes, RouterModule } from '@angular/router';***
       220. We also import the classes for the components that are going to be involved in the routing. I.e.
       221. ***import { UserComponent } from './components/user.component';***
       222. ***import { AboutComponent } from './components/about.component';***
       223. -
       224. Add the following lines of code:
       225. const appRoutes : Routes = [
       226. {
       227. path:'',
       228. component: 'UserComponent'
       229. },
       230. {
       231. path:'about',
       232. component: 'AboutComponent'
       233. }
       234. ];
       235. export const routing : ModuleWithProviders = RouterModule.forRoot(appRoutes);
       236. We then add ‘routing’ to the ‘app.module.ts’ file as thus:
       237. import { NgModule } from '@angular/core';
       238. import { BrowserModule } from '@angular/platform-browser';
       239. import { FormsModule } from '@angular/forms';
       240. import { HttpModule } from '@angular/http';
       241. import { AppComponent } from './app.component';
       242. import { UserComponent } from './components/user.component';
       243. import { AboutComponent } from './components/about.component';
       244. ***import { routing } from './app.routing';***
       245. @NgModule({
       246. imports: [ BrowserModule, FormsModule, HttpModule, ***routing*** ],
       247. declarations: [ AppComponent, UserComponent, AboutComponent ],
       248. bootstrap: [ AppComponent ]
       249. })
       250. export class AppModule { }
       251. Next, we add the following markup to the ‘index.html’ file in the app folder, below the <body> tag:
       252. <body>
       253. ***<base href=”/” />***
       254. …
       255. We then, modify the ‘app.component.ts’ file, setting the template attribute from ‘<user></user>’ to now be, ***‘<router-outlet></router-outlet>’***.
       256. Now, component-to-component routing is now setup. We can simply add more routes in the ‘app.routing.ts’ file.
       257. We can also, add links on our component (pages) to aid with navigating from component to component, by doing this, say in the app.component.ts file:
       258. import { Component } from '@angular/core';
       259. @Component({
       260. selector: 'app',
       261. template: `
       262. ***<ul>***
       263. ***<li><a routerLink="/">Home</a></li>***
       264. ***<li><a routerLink="/about">About</a></li>***
       265. ***</ul>***
       266. <router-outlet></router-outlet>
       267. `
       268. })
       269. export class AppComponent { }