**About Angular:**

* Built and maintained by Google
* Angular 2 was completely rewritten and 2 – 7 have basically been enhancements
* Why use Angular?
  + Organized front – structure (all UI pieces are components)
  + It is powerful and full featured
  + All in one solution (routing, HTTP, RxJs (reactive extensions)
  + Uses MVC design pattern
  + Uses Typescript (uses everything awesome about JS6 plus static typing, classes, arrow functions, etc.)
  + Fantastic CLI
  + Uses “services” to share data/functionality between components
  + Concept of “modules” comprised of root, forms, http modules for making REST API requests.
  + Uses Observables for async operations

**What You Need:**

* Node.JS (we need npm to install Angular CLI)
* VSCode (or preferred IDE)
* Data from <http://jsonplaceholder.typicode.com/todos>

**Commands**

|  |  |
| --- | --- |
| **Command** | **Description** |
| ng serve | Runs and Builds the application and makes it live/responsive. So if you are editing your project, whenever you save the project will recompile and all you have to do is refresh your browser to see changes |
| ng lint | Does a code smell by checking our code for coding standard violations (the rules are defined in tslint.json) |
| ng build | This is a build command for our Angular app. This will make specific files for us inside of **“dist”**, these are the files we need for our project to run in a web server. These are the files we will end up deploying to production. |
| ng new | This is used when you first create the angular project. |
| ng test | Runs our unit tests that are written for our Angular app. Typically the unit tests are written in the Jasmine framework and Karma will run the unit test. This will open a browser and run all the unit tests. They are defined in the following files: **karma.conf.js,** **app.component.spec.ts, test.ts** |
| ng e2e | This runs our End To End test. E2E launches the entire angular app and tests it. This uses Selenium automation testing and downloads the browser drivers needed. The tests are written in a framework called “Protractor” which is apart of Selenium. |
| ng generate | Generates a component for us. Example: “ng generate component welcome” or “ng generate service Authentication” |

**Documents**

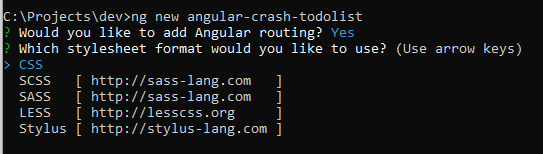
|  |  |
| --- | --- |
| **Document Name** | **Description** |
| index.html  main.ts  test.ts | Just like any index html page. This holds our web app title. Index.html, main.ts, and test.ts are responsible for bootstrapping the application. |
| app.commponent.html | This is where the actual content of the home page is coming from |
| tslint.json | This is where the ng lint rules are defined for code smells |
| karma.conf.js | This is the configuration of the test runner “Karma” |
| app.component.spec.ts | Any file with “spec.ts” is most likely a test |
| protractor.conf.js | Configuration file for Protractor found in the “e2e” folder |
| angular.json | Defines what should happen when we run specific commands |
| src/app | This is where all our angular code goes into. All components and modules are stored here. |
| /assets | This is where all our images are stored |
| /environments | This is where our configuration file for our angular app are. |
| tsconfig.json | This helps translate typescript to JavaScript so our browser can render it |
| package.json | Where all external frameworks and tools are managed for the project. This helps us run our commands. |
| /node\_moduels | Where our dependencies (frameworks and tools) are stored after being downloaded from our package.json file. We did not use npm directly but when we use “ng new” the “npm install” command is automatically executed. |
| polyfills.ts | Helps us manage different browser rendering and browser standards. |
| test.ts | Responsible for running the unit tests |
| app-routing.module.ts | This file helps us route our pages from view to view. Example: route me to welcome component from login component if the login was good. |

**Installing Angular:**

* Run command “npm install -g @angular/cli”

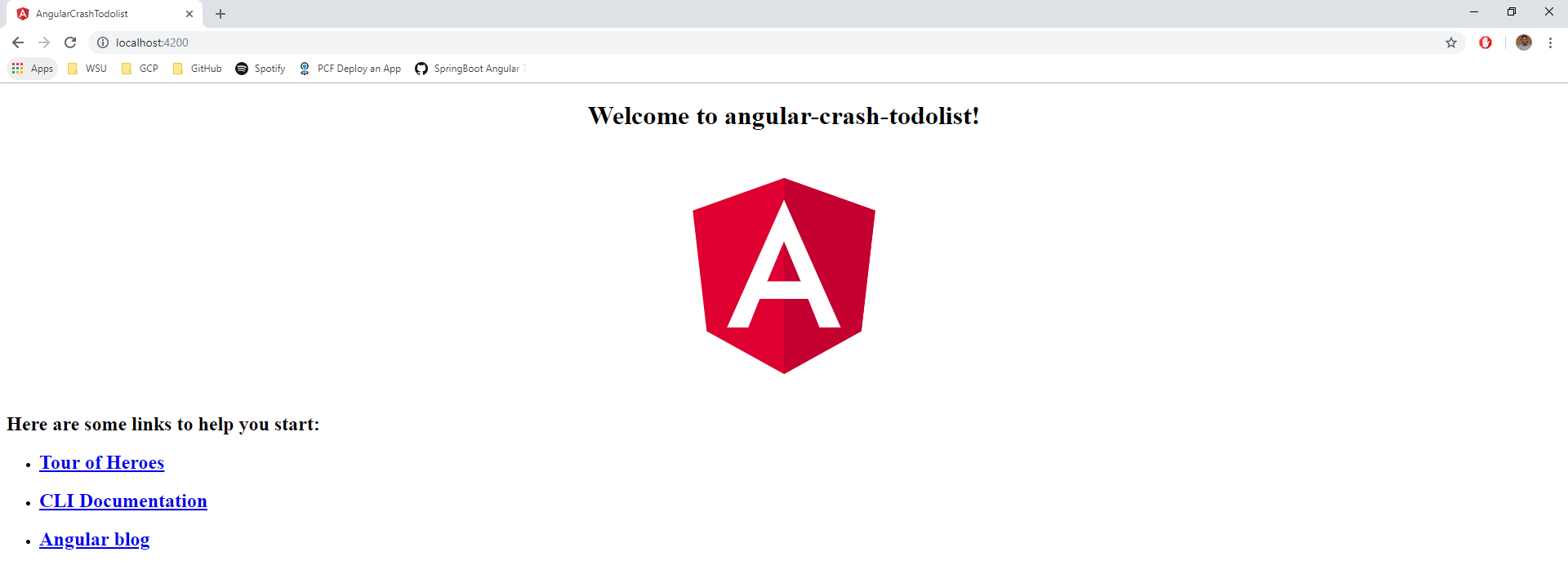
**Generation a New Application:**

* Run command “ng new angular-crash-todolist” to make a new angular app.
* Confirm “Yes” for routing.
* Use CSS for styling.



**Running the App:**

* Run command “ng serve” to run the app. Go to localhost:4200 to see the app running.

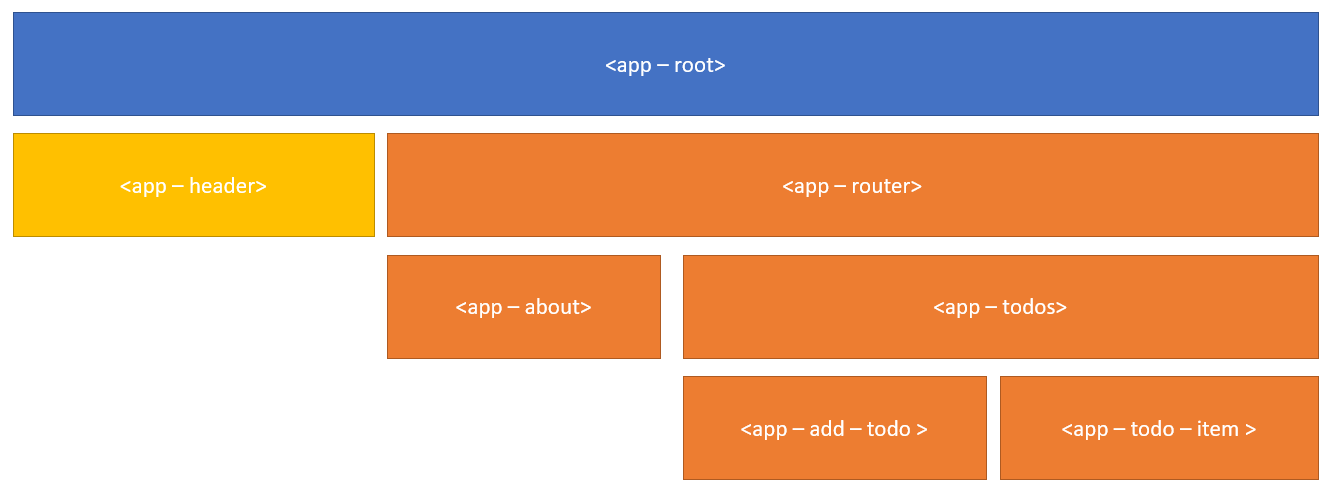


**About the Files:**

* Package.json
  + This shows all the dependencies and packages included in the project.
  + This also has “Scripts” which are angular commands for tasks such as running and building the app.
* Src/index.html
  + This is the main page that loads in the browser.
  + **<app-root></app-root>** is what embeds the root app component.
* Angular.json
  + Where we can import local files (such as styles.css) and frameworks (such as Bootstrap)
* App
  + This is where we will be creating all of our components and services
  + **App.module.ts** is the entry point to angular and the meeting place for all of our components.
    - Declarations is where we will be including our components (the CLI will do that for us)
    - When bringing in imports they need to be imported and listed on this file
  + **App.component.ts**
    - This is where we will be programming the Typescript logic
    - The @Component is a decorator, it includes meta data for the component
      * Selector – what will be used in the html element
      * templateUrl – points to html templet
      * styleUrls - points to styles sheet
  + **App.component.html**
    - Html for that component
  + **App.component.css** 
    - CSS for that component

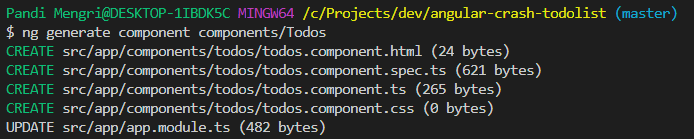
**App Components:**





**Making Todos Component:**

* Run command “ng generate component components/Todos”
  + This will also make a folder named components for us
* Notice, it will create the html, spec.ts, ts, and css files and update the app.module.ts for you.



* If you go into the **todos.component.ts** file you will notice it looks similar to the **app.module.ts** except for the **ngOnInit** method
  + ngOnInit is a lifecycle method. It works like the constructor by running right away. **Note:** you want to use this instead of this constructor. You mainly only want to use the constructor for importing services.

**Adding a Header Component**

* Run command “ng g c components/layout/Header” this will make the Header component in the layout folder.
* Create a basic header with HTML & CSS
* Add this component to the main **app.component.html**

HTML (header.component.html)

<header class="header">

<h1>TodoList</h1>

<nav><a routerLink="/">Home</a> | <a routerLink="/about">About</a></nav>

</header>

CSS (header.component.css)

.header {

background: #333;

color: #fff;

text-align: center;

padding: 10px;

}

.header a {

color: #fff;

text-decoration: none;

}

* Insert the header component into the app by adding the component selector (from TS file) into the **app.component.html** file.

**Making Add Todos Component**

* Run command “ng g c components/AddTodo” to make a new **AddTodo** component
* This is going to be a basic input field with HTML & CSS
* In the **add-todo.component.html** create a form with a text input and a name of “title” to bind the title input to the title property in the TS file, we need to use **[(ngModel)]=”title”**. Also add a button submit.

HTML (add-todo.component.html)

<form class="form" (ngSubmit)="onSubmit()">

<input

type="text"

name="title"

[(ngModel)]="title"

placeholder="Add Todo..."

/>

<input type="submit" value="Submit" class="btn" />

</form>

CSS (add-todo.component.css)

.form {

display: flex;

}

.form input[type="text"] {

flex: 10;

padding: 5px;

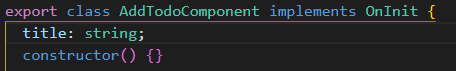
}

.form input[type="submit"] {

flex: 2;

}

* In the **add-todo.component.ts** add the variable title as a string into the class. This will bind it to the TS class.



**Creating Todo Model:**

* First, make a “models” folder inside of app. Then make a file called **Todos.ts** this will be a class with fields we want of Todo.

export class Todo {

id: number;

title: string;

completed: boolean;

}

* Import this class and add it to the TodosComponent class in the **todos.component.ts**

import { Todo } from "../../models/Todos";

export class TodosComponent implements OnInit {

todos: Todo[];

**Outputting the Todos on the App:**

* Run command “ng g c components/TodoItem”
* In the **todo-item.component.html** add {{ todo.title }}

{{ todo.title }}

* In the **todos.component.html** page, add **<app-todo-item></app-todo-item>**. Loop over the list of todo items by using **\*ngFor = “let todo of todos” [todo] = “todo”**
  + We are basically passing in the property “todo” into the <app-todo-item>

<app-todo-item

\*ngFor="let todo of todos"

[todo]="todo"

>

</app-todo-item>

* When we pass in something like “[todo] = “todo”” we need to add it as an @Input in our TS file.
* In the **todo-item.component.ts,** add “Input” to the import of ‘@angular/core’ and in the class bring in an @Input property of todo.

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

export class TodoItemComponent implements OnInit {

@Input() todo: Todo;

* With HTML & CSS build the todo-item component.
* Use interpolation in the todo-item html to add the todo.title and add a checkbox and delete button.

(todo-item.component.html)

<div>

<p>

<input type="checkbox" />

{{ todo.title }}

<button class="del">x</button>

</p>

</div>

(todo-item.component.css)

.del {

background: #ff0000;

color: #fff;

border: none;

padding: 5px 9px;

border-radius: 50%;

cursor: pointer;

float: right;

}

.todo {

background: #f4f4f4;

padding: 10px;

border-bottom: 1px #ccc dotted;

}

**Line Through An “Is Complete” Todo Item:**

* First, use the Ng classes directive in the **todo-item.component.html**. Set the div to [ngClass]= “setClasses()”.
* In the checkbox add a “change” condition = “onToggle(todo)”
* In the button add a “click” condition = “onDelete(todo)”

<div [ngClass]="setClasses()">

<p>

<input (change)="onToggle(todo)" type="checkbox" />

{{ todo.title }}

<button (click)="onDelete(todo)" class="del">x</button>

</p>

</div>

* Next, go to your **todo-item.component.ts** file and add a new class called “setClasses” that is responsible for setting dynamic classes.

setClasses() {

let classes = {

todo: true,

"is-complete": this.todo.completed

};

return classes;

}

* Finally, style the todo list component in the **todo-item.component.css**. Target it with .is-complete as shown below.

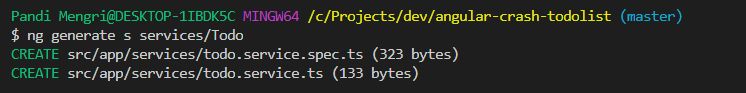
.is-complete {

text-decoration: line-through;

}

**Generating the Service Layer:**

* Run command “ng generate s services/Todo” to generate a new service called “Todo” stored in the services folder.



* Go to **todo.service.ts** the first thing you will notice is that it has created an injectable so that we can inject this service in a constructor of a component.
* In the **todos.component.ts**  import the todos service and initialize the service in the constructor.

import { TodoService } from "../../services/todo.service";

constructor(private todoService: TodoService) {}

**Getting Data from an API Backend (jsonplaceholder API) + GET REQUEST:**

* First, we need to implement the HTTP module that lets us do REST API requests. This is done in **app.module.ts** we need to import **HttpClientModule** and add it to our imports.

import { HttpClientModule } from "@angular/common/http";

imports: [BrowserModule, AppRoutingModule, HttpClientModule, FormsModule],

* Next, we need to import **HttpClient** and **HttpHeaders** into our **todo.service.ts** and initiate **HttpClient** in our constructor.

import { HttpClient, HttpHeaders } from "@angular/common/http";

constructor(private http: HttpClient) {}

* Import **Todo** from our models folder

import { Todo } from "../models/Todo";

* Make a variable called todosURL that contains the URL to the jsonplaceholder API.
* Make a variable called todosLimit that contains a limit of 5 todos.

todosUrl: string = "https://jsonplaceholder.typicode.com/todos";

todosLimit = "?\_limit=5";

* Have the **getTodos()** function return the API GET request.
* **getTodos()** is going to return a todo Observable so we need to declare it as that type.
  + Note: Angular should auto import this library for you. If not it is “import { Observable } from “rxjs”
* In the GET method pass in the todo format (in an array) by passing in the todo model **<Todo[]>**, pass in the todosUrl and todosLimit here.

// Get Todos

getTodos(): Observable<Todo[]> {

return this.http.get<Todo[]>(`${this.todosUrl}${this.todosLimit}`);

}

* In the **todos.component.ts** we need to subscribe to the **getTodos** observable. Set todos to this.todos

ngOnInit() {

this.todoService.getTodos().subscribe(todos => {

this.todos = todos;

});

}

**PUT Request for Crossed Todo:**

* First, in the **todo-item.component.ts** bring in **TodoService** and inject it into the constructor.

import { TodoService } from "../../services/todo.service";

* Next, add a function called onToggle that toggles the service side PUT Request.
  + We will call a method inside of the service to toggle the todo to completed/not completed.
  + This will return an observable so we need to use .subscribe to get the todo back.

onToggle(todo) {

// Toggle in UI

todo.completed = !todo.completed;

// Toggle on server

this.todoService.toggleCompleted(todo).subscribe(todo => console.log(todo));

}

* Next, in the **todo.service.ts** we will create the toggleCompleted function, pass in todo, and have it return an observable with the format as <any> because it is not going to be formatted as an exact todo. Finally return the put request, todo, httpOptions (this includes the header of content type. Since we’re sending data we have to send the content type of application JSON). Create the httpOptions variable that contains this information.

const httpOptions = {

headers: new HttpHeaders({

"Content-Type": "application/json"

})

};

// Toggle Completed

toggleCompleted(todo: Todo): Observable<any> {

const url = `${this.todosUrl}/${todo.id}`;

return this.http.put(url, todo, httpOptions);

}

* Now when a todo item is market completed the JSON will be the following



**DELETE Request:**

* To delete we need to access the todos in the UI (which are in the todos component where we are embedding the todo item)
* To do this we first need to bring in **EventEmitter** and **Output** into **todo-item.component.ts**

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

* Next we need the **onDelete** function to emit the todo.

onDelete(todo) {

this.deleteTodo.emit(todo);

}

* Now that we are emitting something out we need an **@Output()** and we need to call the method **deleteTodo** and set it to **EventEmitter<Todo>**

export class TodoItemComponent implements OnInit {

@Input() todo: Todo;

@Output() deleteTodo: EventEmitter<Todo> = new EventEmitter();

* We need to catch **deleteTodo** 
  + First, in the **todos.component.html**  add a deleteTodo parameter in **<app-todo-item>**

(deleteTodo)="deleteTodo($event)"

* + Next, in the **todos.component.ts** add a new function called **deleteTodo** 
    - This function will filter out the todos on the UI that match the id of the todo that is being deleted.
    - This function will also delete it on the server side

deleteTodo(todo: Todo) {

//UI

this.todos = this.todos.filter(t => t.id !== todo.id);

//Service

this.todoService.deleteTodo(todo).subscribe();

}

* + Next, create the **deleteTodo** in the service
    - Our URL will need the ID so we know what we are deleting.
    - We will need to pass in the httpOptions.

// Delete Todo

deleteTodo(todo: Todo): Observable<Todo> {

const url = `${this.todosUrl}/${todo.id}`;

return this.http.delete<Todo>(url, httpOptions);

}

**Add Todos (POST Request)**

* Here we want to add a Todo to the UI and a POST request to go to the server
* First add (ngSubmit)=”onSubmit()” to the **add-todo** form.

<form class="form" (ngSubmit)="onSubmit()">

* In the TS file create the **onSubmit** function.
  + In this function add the title and completed, ID is not needed because JSON will do it for us.
  + We need to emit this upwards because we need access to the direct Todos in the todo component. Therefore import EventEmitter and Output
  + In the onSubmit add this.addTodo.emit(todo) and in order to do this we first need to declare our output in the AddTodoComponent Class

export class AddTodoComponent implements OnInit {

@Output() addTodo: EventEmitter<any> = new EventEmitter();

onSubmit() {

const todo = {

//ID is not needed bc backend will create for us.

title: this.title,

completed: false

};

this.addTodo.emit(todo);

}

* + We need to catch this emit so in the HTML we need to go to the <app-add-todo> and add “(addTodo)” in the **todos.component.html.** This basically sets it to run a method in our todos component called “addTodo”

<app-add-todo (addTodo)="addTodo($event)"></app-add-todo>

* + Add the method **addTodo**  in the **todos.component.ts**
    - This will make a post request to the server through the service.
    - Once we get the response back we will add it to our UI

addTodo(todo: Todo) {

this.todoService.addTodo(todo).subscribe(todo => {

this.todos.push(todo);

});

}

* + In the service write the **addTodo** method that returns a post request that takes the URL, todo, httpOptions

// Add Todo

addTodo(todo: Todo): Observable<Todo> {

return this.http.post<Todo>(this.todosUrl, todo, httpOptions);

}

**How to Use the Router:**

* Routing is done in **app-routing.module.ts**
* Anything we want to add as a route we need to bring in.
  + Import the TodosComponent

import { TodosComponent } from "./components/todos/todos.component";

* + Add the path to the Routes

const routes: Routes = [

{ path: "", component: TodosComponent },

* For the router to work, we need to have the router in the **app.component.html**

<div>

<app-header></app-header>

<router-outlet></router-outlet>

</div>

* If you want to route an about page, first run “ng g c components/pages/About”
* Make an about component with basic HTML

<div>

<h1>About</h1>

<p>This is the TodoList app v1.0.0. It is part of a Angular crash course</p>

</div>

* Next import and add the AboutComponet to the routing as shown below.

import { AboutComponent } from "./components/pages/about/about.component";

const routes: Routes = [

{ path: "", component: TodosComponent },

{ path: "about", component: AboutComponent }

];

* Finally, to add navigation use routerLink. For example, if you wanted to link the header with Home and About then to this following below

<header class="header">

<h1>TodoList</h1>

<nav><a routerLink="/">Home</a> | <a routerLink="/about">About</a></nav>

</header>