**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)
  + Build powerful SPA apps
  + 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:**

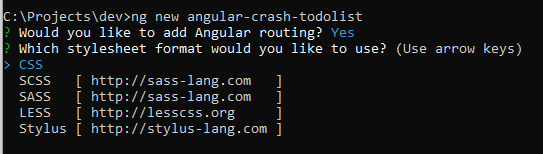
* Node.JS (we need npm to install Angular CLI)
* Data from <http://jsonplaceholder.typicode.com/todos>

**Installing Angular:**

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

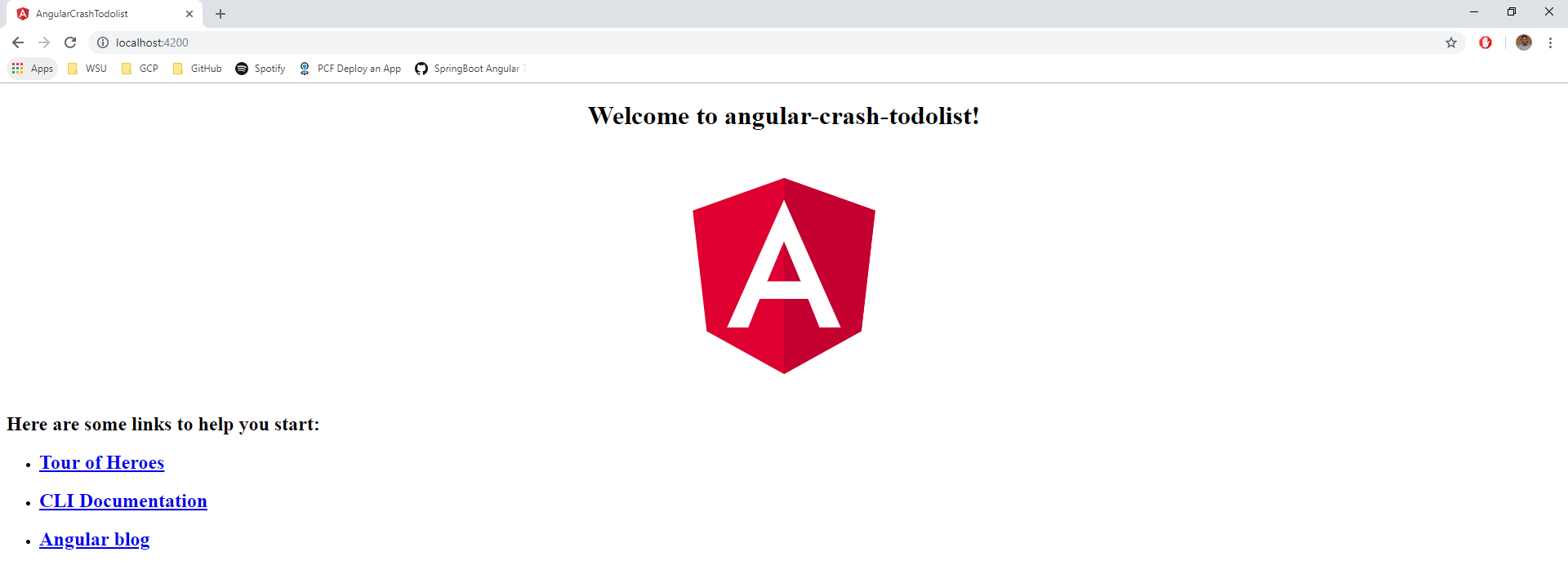
**Generation a New Application:**

* 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.

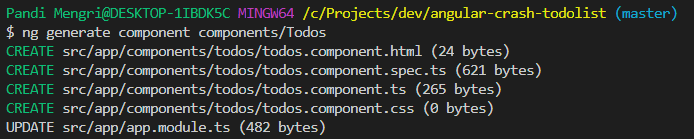


**About the Files:**

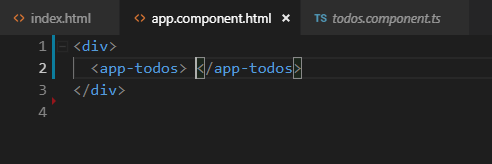
* Package.json
  + This shows all the dependencies and packages included in the project.
  + This also has “Scripts” which are angular commands for tasks such as running and building the app.
* Src/index.html
  + This is the main page that loads in the browser.
  + <app-root></app-root> is what embeds 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 using 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

**Making a New 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.

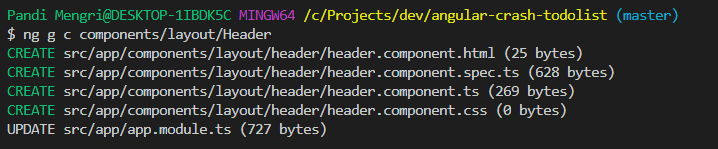


* If you go into the **todos.component.ts** file you will notice it looks similar to the **app.module.ts** except for the **ngOnInit** method
  + ngOnInit is a lifecycle method. It works like the constructor by running right away. **Note:** you want to use this instead of this constructor. You mainly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



**Adding a Header Component**

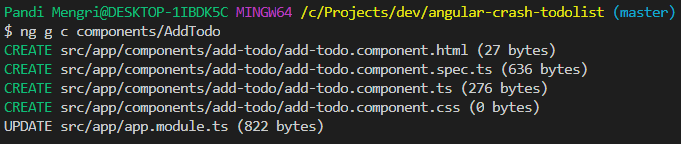
* Run command “ng g c components/layout/Header” this will make the Header component in the layout folder.



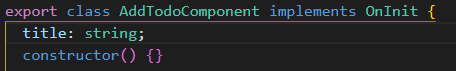
* Update the html to have “TodoList” as a header and add this component to the main **app.component.html**

**Add Todos Component**

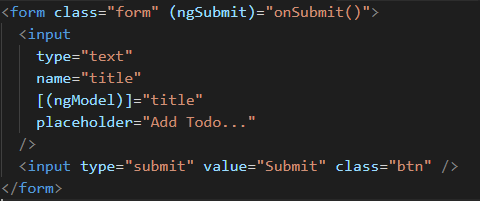
* Run command “ng g c components/AddTodo” to make a new **AddTodo** component



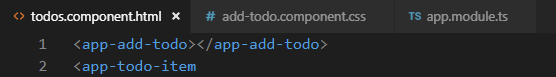
* This is basically going to be a basic title input
* In the **add-todo.component.ts** and add the variable title as string into the class



* 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.
  + NOTE!: in order to use ngModel we need to uses the forms module. Therefore, we need to import that into the **app.module.ts**

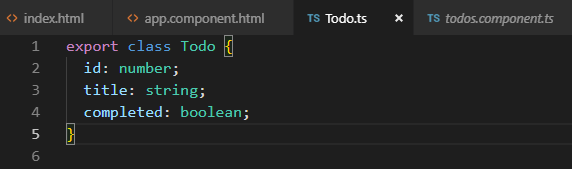


* Embed the add-todos In the **todos.component.html** at the top of the page.



**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.

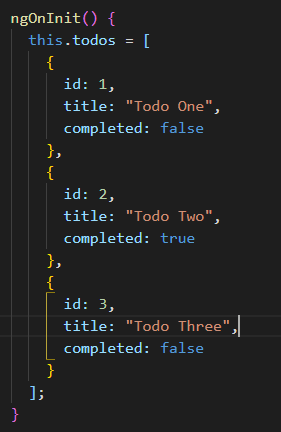


* Import this class and add it to the TodosComponent class in the **todos.component.ts**



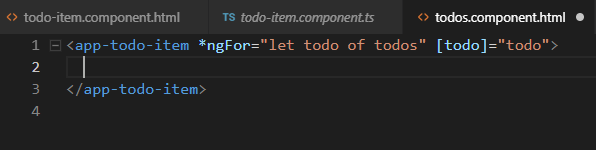
**Adding Todos to an Array:**

* Add the array of objects (todos) in the **ngOnInit**



**Outputting the Todos on the App:**

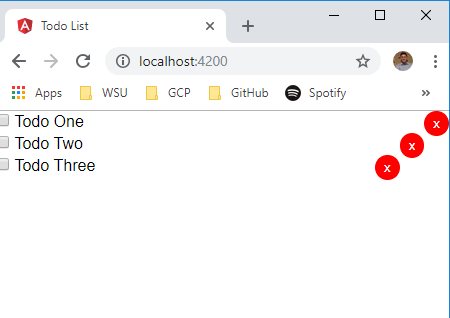
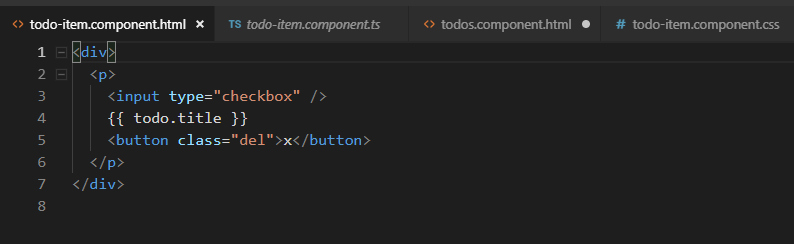
* Run command “ng generate component/components/TodoItem”
* In the **todo-item.component.html** add {{ todo.title }}
* In the **todos.component.html** page, add **<app-todo-item></app-todo-item>**. Loop over the list of todo items by using **\*ngFor = “let todo of todos” [todo] = “todo”**
  + We are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



* 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.

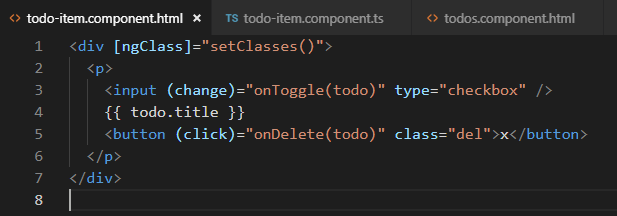


* Use interpolation in the todo-item html to add the todo.title and add a checkbox and delete button.

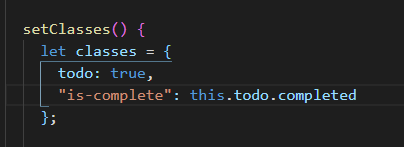


**Line Through An “Is Complete” Todo Item:**

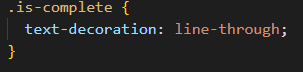
* First, use the Ng classes directive in the **todo-item.component.html**. Set the div to [ngClass]= “setClasses()”.
* In the checkbox add a “change” condition = “onToggle(todo)”
* In the button add a “click” condition = “onDelete(todo)”



* Next, go to your **todo-item.component.ts** file and add a new class called “setClasses” and create a class that sets the todo to “completed”



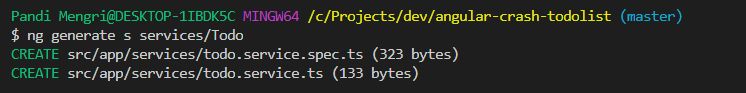
* Finally, style the todo list component in the **todo-item.component.css**. Target it with .is-complete as shown below.



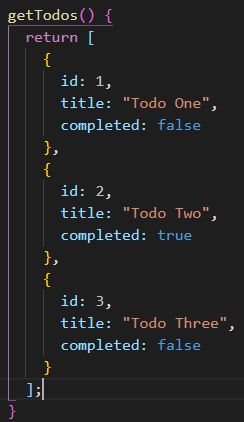
* Stopped taking notes around 44

**(Understanding the service layer) Moving Data to a Service:**

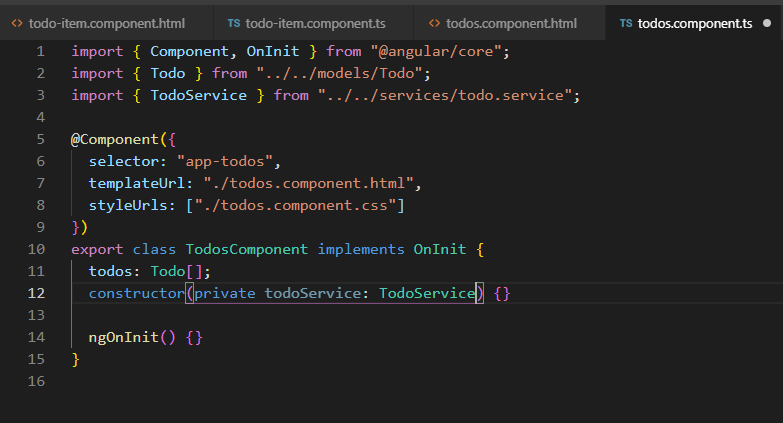
* Run command “ng generate s services/Todo” to generate a new service called “Todo” stored in the services folder.



* 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.
* Create a new function named **getTodos()** that returns the ToDo data.



* In the **todos.component.ts**  import the todos service and initialize the service in the constructor.



**Getting Data from an API Backend (jsonplaceholder API):**

* The intention here is to move away from hard coding our data and move into getting data from an API.
* 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.

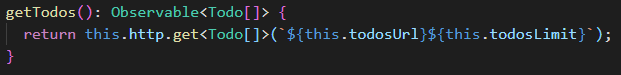




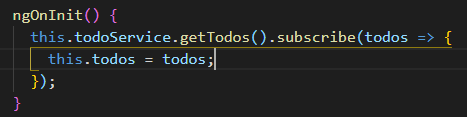
* Next, we need to import **HttpClient** and **HttpHeaders** into our **todo.service.ts** and initiate **HttpClient** in our constructor.
* Import **Todo** from our models folder
* Make a variable called todosURL that contains the URL to the jsonplaceholder API.
* Make a variable called todosLimit that contains a limit of 5 todos.



* 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.

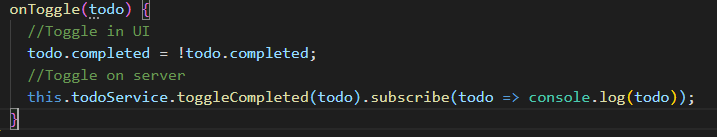


* In the **todos.component.ts** we need to subscribe to the **getTodos** observable. Set todos to this.todos

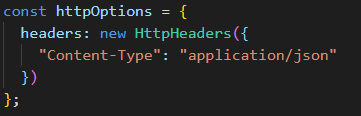


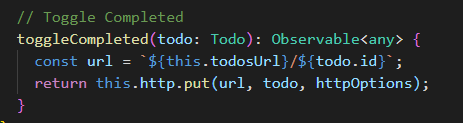
**PUT Request for Crossed Todo:**

* First, in the **todo-item.component.ts** bring in **TodoService** and inject it into the constructor.
* Next, in the onToggle function, add a line of code to toggle 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.



* 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.



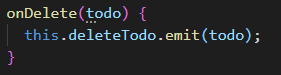


* Now when a todo item is market completed we the JSON will be the following

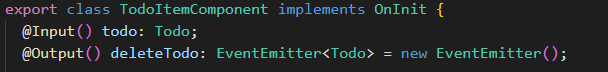


**DELETE Request:**

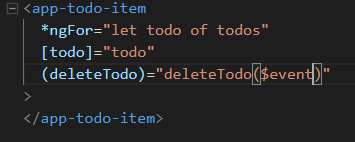
* To delete we need to access the todos in the UI (which are in the todos component where we are embedding the todo item)
* To do this we first need to bring in **EventEmitter** and **Output** into **todo-item.component.ts**
* Next we need the **onDelete** function to emit the todo.



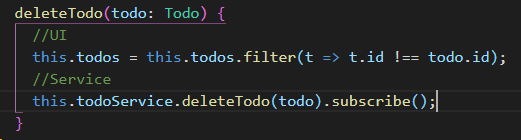
* Now that we are emitting something out we need an **@Output()** and we need to call the method **deleteTodo** and set it to **EventEmitter<Todo>**



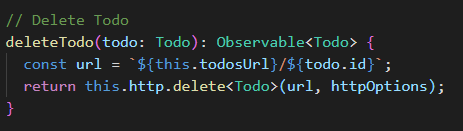
* We need to catch **deleteTodo** 
  + First, in the **todos.component.html**  add a deleteTodo parameter as shown



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



* + 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.



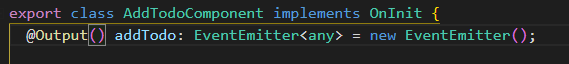
**Add Todos (POST Request)**

* 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.

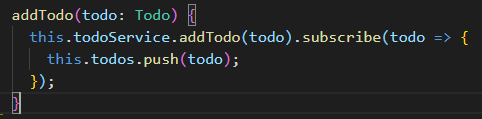


* In the TS file create the onSubmit function.
  + In this function add the title and completed, ID is not needed because JSON will do it for us.
  + We need to emit this upwards because we need access to the direct Todos in the todo component. Therefore import EventEmitter and Output
  + In the onSubmit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

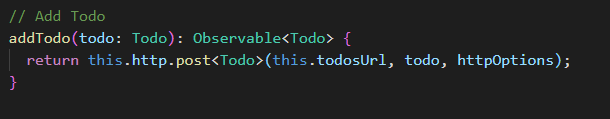




* + We need to catch this emit so in the HTML we need to go to the addto and add “(addTodo)” in the **todos.component.html.** This basically sets it to do run a method in our todos component called “addTodo”
  + Add the method **addTodo**  in the **todos.component.ts**
    - This will make a post request to the server through the service.
    - Once we get the response back we will add it to our UI

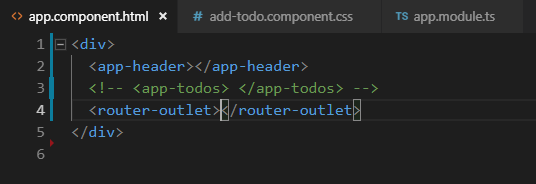


* + In the service write the **addTodo** method that returns a post request that takes the URL, todo, httpOptions



**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
  + Add the path to the Routes
* For the router to work, we need to have the router in the **app.component.html**



* If you want to route an about page, first run “ng g c components/pages/About” to make an about component
* Next add the AboutComponet to the routing as shown below.



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

