**< Overview />**

The intention of this document is to make team members familiar with the Angular framework: What it is, how to use it, and the supporting technologies it leverages.

What this document doesn’t do is provide a comprehensive, in-depth tutorial of Angular because there are many fine tutorials on the web.

A working, fully-functional and documented Angular 5-based project will be used along with the different topics in this document and to demonstrate functionality. A link to this project’s git repository is in the Addendum at the end of this document should anyone wish to clone it.

1. **What you should know (*at some level*) beforehand**

I’ve been unable to install Angular on my *personal* Windows 10 Ubuntu subsystem in the bash shell (which is still in beta). If you have a Mac or a Linux-based machine then you’re 95% of the way there! However, if you don’t have a Mac, I feel the best approach is to install Ubuntu (or another flavor of Linux) in a separate partition on your *personal* Windows laptop and set up a dual boot.

* 1. Some understanding of what a web framework is and what is does
     1. Angular and Sails are both web frameworks.
     2. Angular is a frontend framework and Sails is a backend framework (although it can be used as a frontend framework as well).
  2. How to use a Command Line Interface (CLI)
     1. In my experience, Angular and its accompanying technologies rely heavily on being installed and managed via the CLI (e.g. npm)
  3. HTML, CSS, and JavaScript
     1. A working knowledge – at some level – of all three is needed.
  4. TypeScript
     1. An understanding of TypeScript which is a superset of JavaScript.
     2. Angular is written in TypeScript.
  5. How to use the Node Package Manager (npm)
     1. The libraries used by Angular are managed by npm and Angular is installed using npm

**<** **System Stack />**

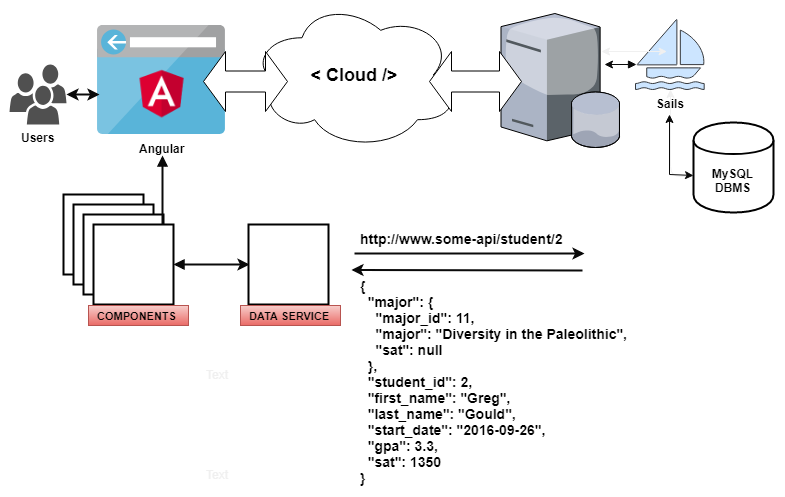
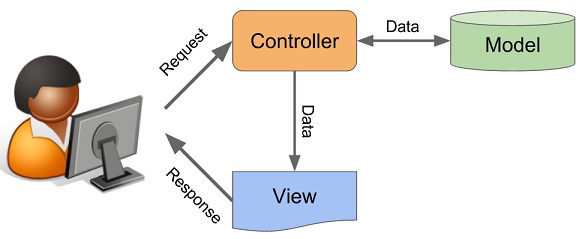
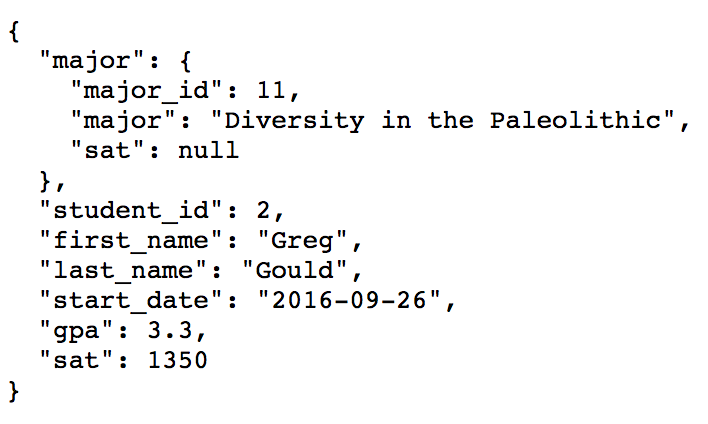


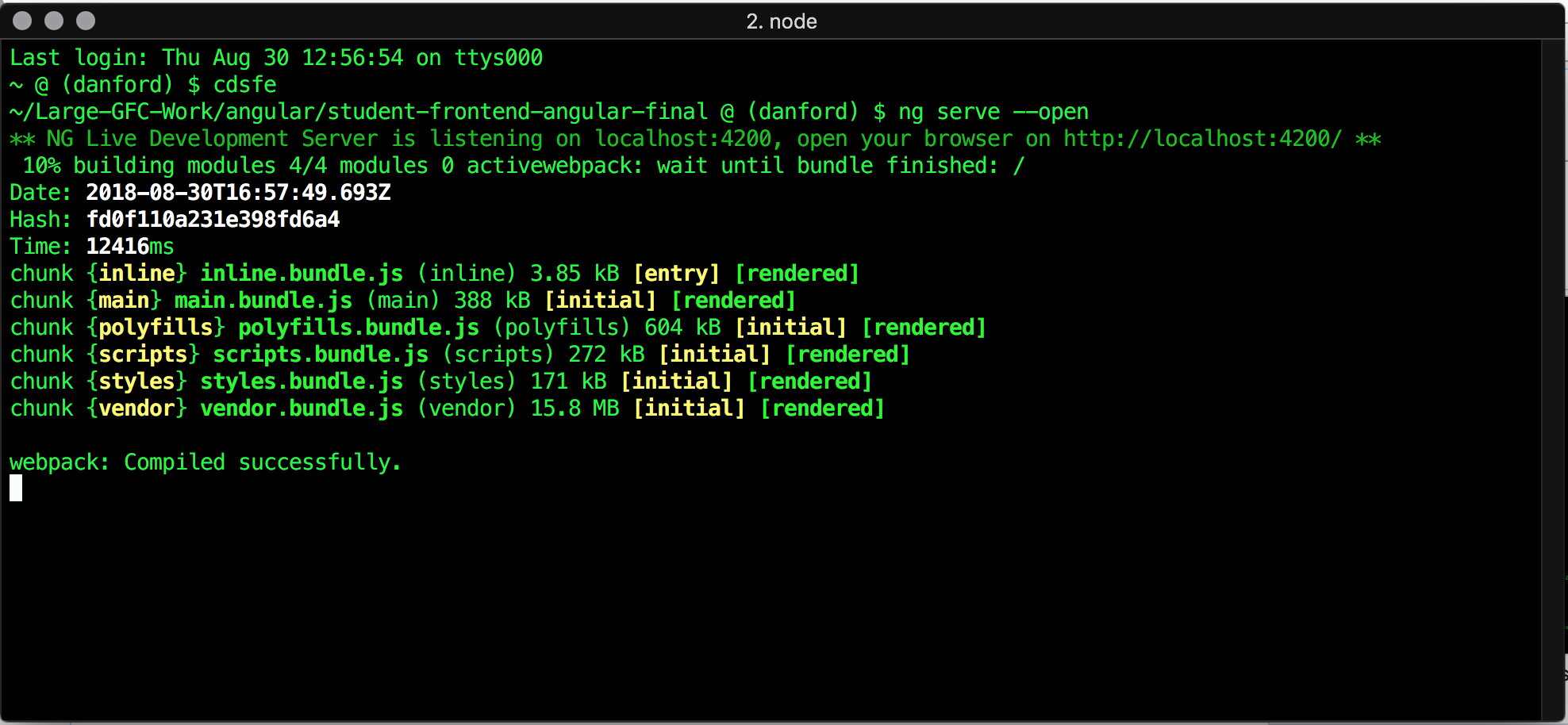
Figure 1 - High-level view demonstrating the Angular / MVC separation of concerns

1. **What is a** [**Model-View-Controller**](https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture)**?**
   1. Model View Controller (MVC) is a software architecture pattern used to implement user interfaces: it a popular choice for architecting web apps. In general, it separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse.  
        
      
   2. View (*Angular* *HTML templates*)
      1. Angular HTML **component** that binds to model data (e.g. student.component.hrml also called a 'template')
   3. Controller (*Angular* *Routing & Data Capture*)
      1. Angular TypeScript, Routing, and Services (e.g. student.component.ts, routing.module.ts, and data.service.ts)
      2. Called API Endpoint: <http://localhost:1337/student/2>
      3. JSON response from endpoint:  
           
         
   4. Model (*Data*)
      1. A Web API that returns JSON data to the data.service.ts
         * I am using a Sails API framework that serves up data from a MySQL database
         * This can be any model if the frontend is updated to handle the data
2. **Frontend Styling (*View*)**
   1. **What is** [**CSS**](https://www.w3schools.com/css/)**?**
      1. Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript
      2. Each Angular component has its own CSS file and it also has an system-wide CSS file named styles.css
   2. **What is** [**Bootstrap 4**](https://getbootstrap.com/)**?**
      1. Bootstrap is a frontend presentation framework. It is a free, open-source library for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions
      2. Bootstrap allows users to quickly create very professional-looking web pages. The use of their templates enforces a consistent look-and-feel across an entire website.
3. **Backend Setup (*Model*)**
   1. **What is** [**Sails**](https://sailsjs.com/)**?**
      1. It’s important to note that while I’m using a Sails/MySQL backend to serve up data, the Angular frontend of the demonstration project can send and receive data to and from any endpoint if it’s been modified to do so
      2. Sails is a "web framework" meaning the "back-end web". This evokes concepts like REST, or HTTP, or WebSockets; and technologies like Java, or Ruby, or Node.js. A "back-end web" framework helps you do things like build APIs, serve HTML files, and handle hundreds of thousands of simultaneous users
      3. Sails is the most popular MVC framework for Node.js, designed to emulate the familiar MVC pattern of frameworks
      4. Why not just use Sails instead of Angular as your frontend? That can be done as well!
         * Links to the Sails frontend and backend git repositories are included in the Addendum at the end of this document
   2. **What is** [**MySQL**](https://www.mysql.com/)**?**
      1. MySQL is an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX, and Windows.

[**Back to Top**](#SystemStack)

**< Technology Stack />**

1. **What is a** [**CLI**](https://en.wikipedia.org/wiki/Command-line_interface)**?**
   1. From the web: A command-line interface or command language interpreter (CLI) is a means of interacting with a computer program where the [user](https://en.wikipedia.org/wiki/User_(computing)) (or client) issues commands to the program in the form of successive lines of text (command lines).



1. **What is** [**npm**](https://www.npmjs.com/)**?**
   1. npm (Node Package Manager) is a package manager for Node.js with hundreds of thousands of packages
   2. It automates dependency and package management
   3. All project package dependencies are contained in the package.json file
      1. Executing the "npm install" command will use the package.json file to (re)install all the project dependencies
   4. All the node packages are contained in the project "node\_modules" folder.
2. **What is** [**Node.js**](https://nodejs.org/en/)**?**
3. Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser
4. Node is effectively server-side JavaScript
5. **What is** [**TypeScript**](https://www.typescriptlang.org/)**?**
6. Angular is written in TypeScript
7. TypeScript is JavaScript for application-scale development
8. TypeScript is an open-source programming language developed and maintained by Microsoft. **It is a strict syntactical superset of JavaScript and adds optional static typing to the language**. Meaning you can enforce typing. For example, plain JavaScript does not enforce typing so a string can be defined as:  
     
   var successMsg;  
     
   Whereas using TypeScript to enforce the String type:  
     
   var successMsg: string;
9. Here is a good [TypeScript tutorial](https://www.tutorialspoint.com/typescript/typescript_overview.htm)
10. **What is** [**RxJS**](https://angular.io/guide/rx-library)**?**
11. RxJS or *Reactive Extensions for JavaScript* is a library for transforming, composing, and querying streams of data ([Learn RxJS](https://www.learnrxjs.io/))
12. Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change ([Wikipedia](https://en.wikipedia.org/wiki/Reactive_programming))
13. RxJS provides an implementation of the "*Observable*" type, which is needed until the type becomes part of the language and until browsers support it

[**Back to top**](#SystemStack)

**< Angular />**

Before starting…

Angular is a very large and complex framework and it can be challenging for newcomers. With that said, there are still parts of Angular I don’t fully understand.

In GoForCode, we were told that many frontend developers have used Angular for some time and still don’t fully grasp how it works. In that respect, I feel that learning how to correctly assemble the pieces to achieve a deliverable result is more important than, say, understanding how Observables really work beneath the covers. It’s enough to understand, for example, how to create an Observable.

Think of it as plugging a new graphics card into a PCI-E port: You could care less how it does what it does. You simply want it to render your UI graphics correctly. Same thing.

1. **What is** [**Angular**](https://angular.io/)**?**
   1. Angular is a platform and a framework for building client-side applications in HTML and TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps. It is built entirely with static files meaning the entire application is loaded into the user’s browser (meaning its fast)
   2. You can use the online Angular [QuickStart](https://angular.io/guide/quickstart) to bootstrap a project
   3. Angular CLI Commands  
      * + Uninstall Angular
          - npm uninstall -g angular-cli
          - npm uninstall --save-dev angular-cli
        + Install Angular Globally
          - npm cache clean
          - npm install -g @angular/cli@latest
        + Create a new Angular Project using angular CLI
          - ng new my-application
          - cd to my-application
          - ng serve –open
        + Create New Angular Component, Service, & Route
          - ng generate component *my-component*
          - ng generate service *my-service*
          - ng generate module app-routing --flat --module=app

**--flat** puts the file in src/app instead of its own folder

**--module=app** tells the CLI to register it in the imports array of the app.module

* 1. A word or two about the **node\_modules** folder
     + - This folder is in the Angular project folder
       - Packages are dropped into the node\_modules folder. When installing locally, this means that you can *require("packagename")* to load its main module, or *require("packagename/lib/path/to/sub/module")* to load other modules.

1. **Why should we use Angular?**

There are lots of reasons to use Angular for frontend web development. Not in any order, here are a few:

* 1. Angular provides true separation of Model (data), View (UI), and Control (business logic). It doesn’t replace the entire DOM every time something in the data changes. It only replaces what’s changed based on what changed in the data
  2. We don’t have to write code to synchronize views with data because Angular does that for us
  3. Because of the “separation of concerns”, Angular projects are much more maintainable
  4. Angular projects were meant to be tested – both unit and end-to-end using tools such as Jasmine
  5. Angular provides both one and two-way data-binding.
  6. Angular doesn’t generate HTML and then pass it to the browser for parsing, rather, it generates DOM data structures directly which is very fast as opposed to, for example, using JQuery

1. **Using Angular**
   * 1. **Angular Project Directory Structure (high-level)**

src/

app/

navigation/

home/ (**landing page component**)

Animations/

...

student/

student.compinent.css **(styles)**

student.component.html **(template)**

student.component.spec.ts **(for testing)**

student.component.ts **(manages the template)**

...

routing/

routing.module.ts

*Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application*

app.module.ts (**system bootstrap**)

app.component.ts (**system bootstrap**)

data.service.ts (**used by components to obtain data**)

assets/

images/

Images used by home.component.html

js/

sortTable.js

index.html

main.ts

styles.css

README.md

package.json

package-lock.json

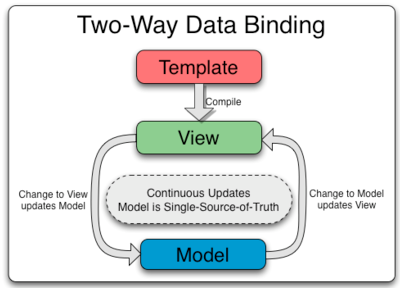
* + - * **index.html** – This is the base of the HTML view for all component HTML files. It pulls in CSS files like bootstrap and script files such as jquery. The “<app-root>” is in the middle of the body and is loaded with views
      * **main.ts** – This is the Angular bootstrapper app (more on it later)
      * **styles.css** - The “styles.css” file is used by Angular for system-wide styling of all component views
      * **package.json** – this file contains all the packages and dependencies used by your angular frontend project
      * **package-lock.json** – automatically generated for any operations where npm modifies either the node\_modules tree, or package.json . It describes the exact tree that was generated, such that subsequent installs can generate identical trees, regardless of intermediate dependency updates
      * **README.md** – Used to generate the html summary you see at the bottom of projects. Github has their own flavor of Markdown language. Check out [the README.md for this project](https://github.com/techlozenge/student-frontend-angular-final). Scroll down below the folder and files list to see its content

|  |  |
| --- | --- |
| **Folder/File** | **Purpose** |
| src/app/ | Angular application files go here. |
| e2e/ | *End-to-end* (e2e) tests of the application, written in Jasmine and run by the [protractor](http://www.protractortest.org/) e2e test runner.  Initialized with an e2e test for the "Hello Angular" sample. |
| node\_modules/ | The *npm* packages installed with the npm install command. |
| .editorconfig .git/ .gitignore | Tooling configuration files and folders. Ignore them until you have a compelling reason to do otherwise. |
| CHANGELOG.md | The history of changes. |
| favicon.ico | The application icon that appears in the browser tab. |
| index.html | The application host page. It loads a few essential scripts in a prescribed order. Then it boots the application, placing the root AppComponent in the custom <my-app> body tag.  The same index.html satisfies all documentation application samples. |
| karma.conf.js | Configuration for the [karma](https://karma-runner.github.io/1.0/index.html) test runner described in the [Testing](https://angular.io/guide/testing) guide. |
| karma-test-shim.js | Script to run [karma](https://karma-runner.github.io/1.0/index.html) with SystemJS as described in the [Testing](https://angular.io/guide/testing) guide. |
| non-essential-files.txt | A list of files that you can delete if you want to purge your setup of the original QuickStart Seed testing and git maintenance artifacts. See instructions in the optional [*Deleting non-essential files*](https://angular.io/guide/setup#non-essential) section. *Do this only in the beginning to avoid accidentally deleting your own tests and git setup!* |
| LICENSE | The open source MIT license to use this setup code in your application. |
| package.json | Identifies npm package dependencies for the project.  Contains command scripts for running the application, running tests, and more. |
| protractor.config.js | Configuration for the [protractor](http://www.protractortest.org/) *end-to-end* (e2e) test runner. |
| README.md | Instruction for using this git repository in your project. |
| styles.css | Global styles for the application. Initialized with an <h1> style for the QuickStart demo. |
| tsconfig.json | Tells the TypeScript compiler how to transpile TypeScript source files into JavaScript files that run in all modern browsers. |
| tslint.json | The npm installed TypeScript linter inspects your TypeScript code and complains when you violate one of its rules.  This file defines linting rules favored by the [Angular style guide](https://angular.io/guide/styleguide) and by the authors of the documentation. |

1. **What is Data Binding?**
   1. Data-binding in Angular apps is the automatic synchronization of data between the model and view components that’s mediated by controllers e.g. data services
   2. Angular supports *two-way data binding*, a mechanism for coordinating the parts of a template with the parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.

|  |  |  |
| --- | --- | --- |
| **Data direction** | **Syntax** | **Type** |
| One-way  from data source  to view target | {{expression}}  [target]="expression"  bind-target="expression" | Interpolation  Property  Attribute  Class  Style |
| One-way  from view target  to data source | (target)="statement"  on-target="statement" | Event |
| Two-way | [(target)]="expression"  bindon-target="expression" | Two-way |





* + 1. The \*ngFor directive tells Angular to iterate over a list
    2. The *{{grade.grade\_id}}, {{grade.grade}}*, and *(click)="deleteGrade(grade.grade\_id, grade.grade)"* bind program data to and from the DOM, responding to user input

<table class="table table-hover" id="dataTable">

<thead class="thead-dark">

<tr>

<th>ID</th>

<th onclick="sortTable(1)"> <i class="fas fa-sort"></i> <u>Grade</u></th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let grade of grades">

<td>{{grade.grade\_id}}</td>

<td>{{grade.grade}}</td>

<td class="text-center">

<a class="btn btn-light" [routerLink]="['/grade/edit/', grade.grade\_id]"><i class="fas fa-pencil-alt"></i></a>&nbsp;

<button (click)="deleteGrade(grade.grade\_id, grade.grade)" class="btn btn-light"><i class="fas fa-trash-alt"></i></button>

</td>

</tr>

</tbody>

</table>

1. **What are Decorators?**

**@Component**({

selector: 'app-student',

templateUrl: './student.component.html',

styleUrls: ['./student.component.css'],

animations: [fadeInAnimation]

})

export class StudentComponent implements OnInit {

...

**@Component**({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css'],

})

export class AppComponent {

**@Input()** errorMessage: string;

}

...

@Component({

selector: 'app-delete-confirm',

templateUrl: './delete-confirm.component.html',

styleUrls: ['./delete-confirm.component.css']

})

export class DeleteConfirmComponent implements OnInit {

constructor**(@Inject(MAT\_DIALOG\_DATA)** public data: any) {}

...

* 1. When used as a design pattern…
     1. They dynamically attach additional responsibilities to an object
     2. They’re a flexible alternative to sub-classing for extending functionality
     3. They will probably become a part of JavaScript ES7
     4. Decorators are part of TypeScript now so we can use them
  2. When used in TypeScript…
     1. They are functions that provide a way to add meta-data to a class, method, accessor, property, or parameter
  3. Decorators are always prefixed by a ‘**@**’ e.g. @Component
  4. Class decorators always appear ***immediately*** before a class
  5. Here are the types of decorators

1. Class decorators, e.g. @Component and @NgModule
   * For example, this is the list of class decorators
     + - @[Component()](https://angular.io/api/core/Component)
       - @[Directive()](https://angular.io/api/core/Directive)
       - @[Pipe()](https://angular.io/api/core/Pipe)
       - @[Injectable()](https://angular.io/api/core/Injectable)
       - @[NgModule()](https://angular.io/api/core/NgModule)
2. Property decorators for properties inside classes e.g. @Input
3. Method decorators for methods inside classes, e.g. @HostListener
4. Parameter decorators for parameters inside class constructors, e.g. @Inject
   1. See [this link](https://toddmotto.com/angular-decorators) for a good article exploring Angular decorators
5. **What are** [**Components**](https://angular.io/api/core/Component)**?**
6. A component defines *views*, which are sets of screen elements
7. A component provides configuration metadata that determines how the component should be processed, instantiated, and used at runtime
8. A singular Component is one view and this view is normally called the 'host' view but its HTML part is also called a template
9. A component must belong to an NgModule for it to be available to another component or application
10. A component consists of an HTML template and a “backend” TypeScript (.ts) class which is the control used to populate the template.
11. You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods

**grade.component.html**

<section @fadeInAnimation>

<br/>

<div [ngClass]="(errorMessage)?'alert alert-danger':'alert alert-success'" role="alert" \*ngIf="errorMessage || successMessage" id="deletemsg">

{{errorMessage}} {{successMessage}}

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<br><br><br>

<h2>Grades</h2>

<a class="btn btn-primary" routerLink="/grade/add">Add Grade</a><br><br>

<table class="table table-hover" id="dataTable">

<thead class="thead-dark">

<tr>

<th>ID</th>

<th onclick="sortTable(1)"> <i class="fas fa-sort"></i> <u>Grade</u></th>

</tr>

</thead>

<tbody>

**<tr \*ngFor="let grade of grades">**

***NOTE: This is an example of one-way data binding***

**<td>{{grade.grade\_id}}</td>**

**<td>{{grade.grade}}</td>**

**<td class="text-center">**

**<a class="btn btn-light" [routerLink]="['/grade/edit/', grade.grade\_id]"><i class="fas fa-pencil-alt"></i></a>&nbsp;**

**FYI: This is known as “**[**Event Binding**](https://angular.io/guide/template-syntax#event-binding)**”**

**<button (click)="deleteGrade(grade.grade\_id, grade.grade)" class="btn btn-light"><i class="fas fa-trash-alt"></i></button>**

**</td>**

**</tr>**

</tbody>

</table>

</section>

**grade.component.ts**

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

import { Component, OnInit,Input } from '@angular/core';

import { MatDialog, MatDialogRef } from '@angular/material';

import { fadeInAnimation } from '../animations/fade-in.animation';

import { DataService } from '../data.service'

import { DeleteConfirmComponent } from '../delete-confirm/delete-confirm.component'

***The @Component decorator identifies the class immediately below it as a component class, and specifies its metadata. The metadata for a component tells Angular where to get the major building blocks that it needs to create and present the component and its view***

**@Component**({

selector: 'app-grade',

templateUrl: './grade.component.html',

styleUrls: ['./grade.component.css'],

animations: [fadeInAnimation]

})

export class GradeComponent implements OnInit {

errorMessage: string;

successMessage: string;

grades: any[];

mode = 'Observable';

constructor (private dataService: DataService, public dialog: MatDialog) {}

ngOnInit() { this.getGrades(); }

**getGrades() {**

**this.dataService.getRecords("grade")**

***NOTE: This is an example of subscribing to an Observer***

**.subscribe(**

**grades => this.grades = grades,**

**error => this.errorMessage = <any>error);**

**}**

deleteGrade(id:number, str:string) {

let dialogRef = this.dialog.open(DeleteConfirmComponent, {

data: {

dataKey: id,

value: str

}

});

dialogRef.afterClosed().subscribe(result => {

if(result){

this.dataService.deleteRecord("grade", id)

.subscribe(

grade => {this.successMessage = "Record(s) deleted succesfully"; this.getGrades(); },

error => this.errorMessage = <any>error);

}

});

} // end deleteGrade

} // end class GradeComponent

1. **What are** [**Observables**](https://angular.io/guide/observables)**?**
2. Observables provide support for passing messages between publishers and subscribers
3. They define a function for publishing values that isn't executed until a consumer subscribes to it
4. They can deliver multiple values of any type and the API for receiving values is the same whether the values are delivered synchronously or asynchronously

**student.component.ts**

import { Component, OnInit,Input } from '@angular/core';

import { MatDialog, MatDialogRef } from '@angular/material';

import { DataService } from '../data.service'

import { DeleteConfirmComponent } from '../delete-confirm/delete-confirm.component'

import { fadeInAnimation } from '../animations/fade-in.animation';

import { Subject } from 'rxjs/Rx';

import 'rxjs/add/operator/map';

***An RxJS Subject is a special type of Observable that allows values to be multicasted to many Observers. While plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable), Subjects are multicast***

**+---------------  
 | => => => => Subscriber  
Sender -----------------------+ +-----------   
Message => => => => => => => => => => => Subscriber  
 -----------------------+ +-----------  
 | => => => => Subscriber  
 +---------------**

@Component({

* + - 1. selector: 'app-student',
      2. templateUrl: './student.component.html',
      3. styleUrls: ['./student.component.css'],
      4. animations: [fadeInAnimation]

})

export class StudentComponent implements OnInit {

errorMessage: string;

successMessage: string;

students: any[];

mode = 'Observable';

constructor (private dataService: DataService, public dialog: MatDialog) {}

ngOnInit(): void {

this.getStudents();

}

**getStudents() {**

**this.dataService.getRecords("student")**

***NOTE: Another example of subscribing to an Observable!***

**.subscribe(**

**students => this.students = students,**

**error => this.errorMessage = <any>error);**

**}**

...

1. **What Are** [**Services**](https://angular.io/tutorial/toh-pt4)**?**
2. Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.
3. Typically, data services perform all the heavy lifting for a group of components. They contain Observables which are subscribed to by the components and once the subscription is made, they send and receive data from an endpoint.

**data-service.ts**

***The data service uses HTTP calls (GET, PUT, POST, DELETE) to web API endpoints and returns the data back to the publisher.***

import { Injectable } from '@angular/core';

import { Http, Response } from '@angular/http';

**import { Observable } from 'rxjs/Observable';**

**import 'rxjs/add/operator/catch';**

**import 'rxjs/add/observable/throw';**

**import 'rxjs/add/operator/map';**

@Injectable()

export class DataService {

private baseUrl = '<http://localhost:1337/>'

constructor (private http: Http) {}

...

***NOTE: The Observable at work!***  
**getRecords(endpoint: string): Observable<any[]> {**

**let apiUrl = this.baseUrl+endpoint;**

**return this.http.get(apiUrl)**

**.map(this.extractData)**

**.catch(this.handleError);**

**}**

...

***The extractData and handleError functions perform the calls and handle any results.***

private **extractData(res: Response)** {

let results = res.json();

return results || [];

}

private **handleError(error: Response | any)** {

let errMsg: string;

if(typeof error.\_body === "string") {

errMsg = error.\_body

} else {

if (error instanceof Response) {

if(error.status === 0) {

errMsg = "Error connecting to API"

} else {

const errorJSON = error.json();

errMsg = errorJSON.message;

}

}

}

return Observable.throw(errMsg);

}  
  
...

1. **What are** [**@NgModules**](https://angular.io/guide/ngmodules)**?**

The basic building blocks of an Angular application are **NgModules**.

NgModules configure the injector and the compiler and help organize related things together. An NgModule is a class marked by the @NgModule **decorator** (identifiers prefixed by an @ are called decorators e.g. [@Component](https://angular.io/api/core/Component), [@Injectable](https://angular.io/guide/dependency-injection), [@NgModule](https://angular.io/guide/ngmodules) and decorators sit immediately above a class definition). @NgModule takes a metadata object that describes how to compile a component's template and how to create an injector at runtime.

1. The @NgModule is metadata e.g. **app.module.ts** (full listing: <https://angular.io/api/core/NgModule>)
2. **declarations**
3. The [components](https://angular.io/guide/architecture-components), *directives*, and *pipes* that belong to this NgModule
4. *declarations: [AppComponent, StudentComponent, HomeComponent, … ]*
5. **imports**
6. Other modules whose exported classes are needed by component templates declared in *this* NgModule
7. *imports: [ BrowserModule, AppRoutingModule, HttpModule, MatDialogModule, BrowserAnimationsModule, FormsModule]*
8. **providers**
9. Creators of [services](https://angular.io/guide/architecture-services) that this NgModule contributes to the global collection of services; they become accessible in all parts of the app
10. *providers: [DataService]*
11. **entryComponents**
12. The set of components to compile when this NgModule is defined, so that they can be dynamically loaded into the view
13. *entryComponents: [DeleteConfirmComponent]*
14. **bootstrap**
15. The main application view, called the ***root component***, which hosts all other app views. Only the *root NgModule* should set the [bootstrap](https://angular.io/api/core/NgModule#bootstrap) property.

*bootstrap: [AppComponent]* Example:   
  
**main.ts**

***This is the main entry point the frontend app. It compiles the application using the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser***  
  
import { enableProdMode } from '@angular/core';

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

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

import { environment } from './environments/environment';

if (environment.production) {

enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule);  
   
**app.module.ts**  
  
***This tells Angular how to assemble the application. By default, it declares only the AppComponent***

...

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

import { BrowserModule } from '@angular/platform-browser';

...

@NgModule({

imports: [ BrowserModule ],

providers: [ Logger ],

declarations: [ AppComponent ],

exports: [ AppComponent ],

bootstrap: [ AppComponent ]

})

...

export class AppModule { }

[**Back to top**](#SystemStack)

1. **What is** [**Routing**](https://angular.io/guide/router)**?**

To understand routing, we need to touch on the bootstrap Angular process:

main🡪app.module🡪app.component

main.ts performs platformBrowserDynamic().bootstrapModule(AppModule); Where AppModule is the entry point for our application. The AppModule imports our AppRoutingModule which is an exported class build by the routing.module that defines where all our components are located and the paths needed to navigate to each of them.

**routing.module.ts**

**Perform imports for our application…**

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

import { RouterModule, Routes } from '@angular/router';

import { StudentComponent } from '../student/student.component';

import { StudentFormComponent } from '../student-form/student-form.component';

import { GradeComponent } from '../grade/grade.component';

import { GradeFormComponent } from '../grade-form/grade-form.component';

import { ClassComponent } from '../class/class.component';

import { ClassFormComponent } from '../class-form/class-form.component';

import { AssignmentComponent } from '../assignment/assignment.component';

import { AssignmentFormComponent } from '../assignment-form/assignment-form.component';

import { InstructorComponent } from '../instructor/instructor.component';

import { InstructorFormComponent } from '../instructor-form/instructor-form.component';

import { MajorComponent } from '../major/major.component';

import { MajorFormComponent } from '../major-form/major-form.component';

import { StudentClassComponent } from '../student-class/student-class.component';

import { StudentClassFormComponent } from '../student-class-form/student-class-form.component';

import { MajorClassComponent } from '../major-class/major-class.component';

import { MajorClassFormComponent } from '../major-class-form/major-class-form.component';

import { HomeComponent } from '../home/home.component';

import { DeleteConfirmComponent } from '../delete-confirm/delete-confirm.component';

**Create the routes table used for our application…**

**const routes: Routes** = [

{ path: '', redirectTo: '/home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent },

{ path: 'student', component: StudentComponent },

{ path: 'student/edit/:id', component: StudentFormComponent },

{ path: 'student/add', component: StudentFormComponent },

{ path: 'grade', component: GradeComponent },

{ path: 'grade/edit/:id', component: GradeFormComponent },

{ path: 'grade/add', component: GradeFormComponent },

{ path: 'class', component: ClassComponent },

{ path: 'class/edit/:id', component: ClassFormComponent },

{ path: 'class/add', component: ClassFormComponent },

{ path: 'assignment', component: AssignmentComponent },

{ path: 'assignment/edit/:id', component: AssignmentFormComponent },

{ path: 'assignment/add', component: AssignmentFormComponent },

{ path: 'instructor', component: InstructorComponent },

{ path: 'instructor/edit/:id', component: InstructorFormComponent },

{ path: 'instructor/add', component: InstructorFormComponent },

{ path: 'major', component: MajorComponent },

{ path: 'major/edit/:id', component: MajorFormComponent },

{ path: 'major/add', component: MajorFormComponent },

{ path: 'student-class', component: StudentClassComponent },

{ path: 'student-class/edit/:id', component: StudentClassFormComponent },

{ path: 'student-class/add', component: StudentClassFormComponent },

{ path: 'major-class', component: MajorClassComponent },

{ path: 'major-class/edit/:id', component: MajorClassFormComponent },

{ path: 'major-class/add', component: MajorClassFormComponent },

{ path: 'delete-confirm', component: DeleteConfirmComponent}

];

**Setup our NgModule and load our routes…**

@NgModule({

imports: [ RouterModule.forRoot(routes) ],

exports: [ RouterModule ]

})

export class AppRoutingModule {}

**navigation.component.html**

The **routerLinkActive** defines a class that can be used when the link is active. This Angular link provides a demonstration of this capability: <https://angular.io/api/router/RouterLinkActive>

The **routerLink** tells Angular the location of the component.

**Notice the use of Bootstrap classes used in the <nav> tag? Bootstrap saves us a lot of time we’d otherwise spend fiddling with CSS.**

<**nav** style="height: 55px;" class="**navbar navbar-collapse navbar-toggleable-md navbar-dark bg-dark fixed-top navbar-expand-sm**" role="navigation">

<div class="collapse navbar-collapse">

<ul class="navbar-nav mr-auto">

<li class="w-16 p-2" style="color: red;">{{today | date}}</li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/home">home</a></li>

<li class="w-16 p-2" **routerLinkActive**="active"><a **routerLink**="/student" >students </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/grade" >grades </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/class" >classes </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/assignment" >assignments </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/instructor" >instructors </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/major" >majors </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/major-class" >major-class </a></li>

<li class="w-16 p-2" routerLinkActive="active"><a routerLink="/student-class" >student-class </a></li>

</ul>

<div class="form-control-sm form-inline ml-auto srch-form">

<form (ngSubmit)="doWebSearch(srchtext)" #searchForm="ngForm">

<div class="input-group">

<input [(ngModel)]="srchtext" id="srchText" name="srchtext" type="text" [(ngModel)]="srchtext" autofocus="autofocus" placeholder="Search the Web...">

<span class="input-group-append">

<button id="srchButton" type="submit" class="btn btn-block btn-info"><i class="fas fa-search"></i></button>

</span>

</div>

</form>

</div>

</div>

</nav>

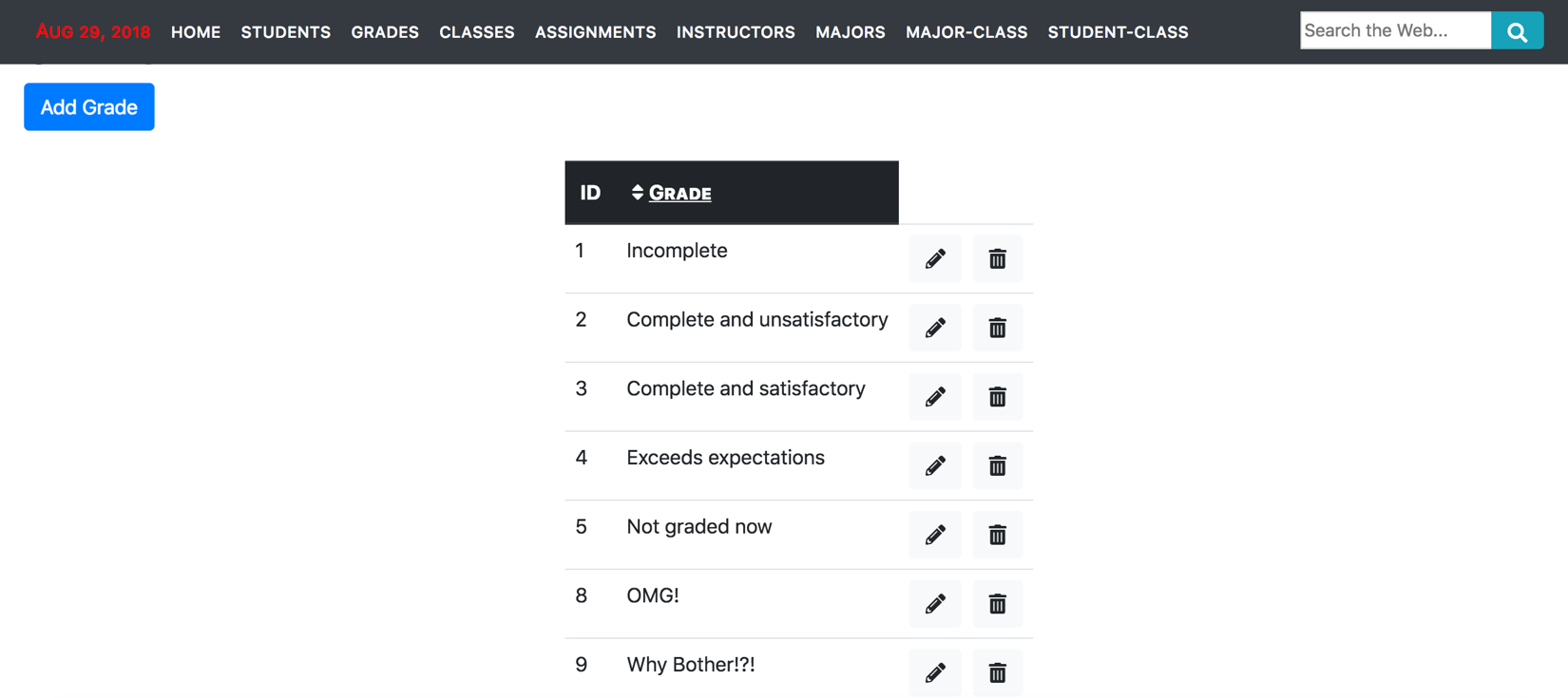
1. **Injecting Data Directly into a Component**

Sometimes components need to communicate data between one another. The goal of this exercise is to demonstrate how this can be done.

My original project had a very simple yes/no dialog box that asked the user if they wanted to delete a record from the system. I wanted to pass some data directly to this component that would provide record content to the user to help them ensure they are deleting the right record.

Initially, a user is presented with a list of records used by the system (image below). Should they wish to remove a record, they click the trash can icon at the end of that records table row and then they’re asked to confirm the deletion.

To begin, the user clicks a trash can and a series of events is initiated that we’ll walk through one step at a time.



**grade.component.html**

***The user clicks a trash can icon and the “deleteGrade” function in the grade.component.ts is performed which takes as arguments the grade’s id and description.***

<table class="table table-hover" id="dataTable">

<thead class="thead-dark">

<tr>

<th>ID</th>

<th onclick="sortTable(1)"> <i class="fas fa-sort"></i> <u>Grade</u></th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let grade of grades">

<td>{{grade.grade\_id}}</td>

<td>{{grade.grade}}</td>

<td class="text-center">

<a class="btn btn-light" [routerLink]="['/grade/edit/', grade.grade\_id]"><i class="fas fa-pencil-alt"></i></a>&nbsp;

<button **(click)="deleteGrade(grade.grade\_id, grade.grade)"** class="btn btn-light"><i class="fas fa-trash-alt"></i></button>

</td>

</tr>

</tbody>

</table>

**grade.component.ts**

***The deleteGrade function opens the DeleteConfirmComponent class and passes it a JSON object containing the grade id and description.***

import { Component, OnInit,Input } from '@angular/core';

import { MatDialog, MatDialogRef } from '@angular/material';

import { fadeInAnimation } from '../animations/fade-in.animation';

import { DataService } from '../data.service'

import { DeleteConfirmComponent } from '../delete-confirm/delete-confirm.component'

@Component({

selector: 'app-grade',

templateUrl: './grade.component.html',

styleUrls: ['./grade.component.css'],

animations: [fadeInAnimation]

})

export class GradeComponent implements OnInit {

errorMessage: string;

successMessage: string;

grades: any[];

mode = 'Observable';

constructor (private dataService: DataService, public dialog: MatDialog) {}

ngOnInit() { this.getGrades(); }

getGrades() {

this.dataService.getRecords("grade")

.subscribe(

grades => this.grades = grades,

error => this.errorMessage = <any>error);

}

deleteGrade(**id:number, str:string**) {

let dialogRef = this.dialog.open(DeleteConfirmComponent, {

**data: {**

**dataKey: id,**

**value: str**

**}**

});

dialogRef.afterClosed().subscribe(result => {

if(result){

this.dataService.deleteRecord("grade", id)

.subscribe(

grade => {this.successMessage = "Record(s) deleted succesfully"; this.getGrades(); },

error => this.errorMessage = <any>error);

}

});

}

}

**delete-confirm-component.ts**

***To receive the data from its HTML template, the delete-confirm.component.ts needs to import the MAT\_DIALOG\_DATA (which is part of the MatDialog library). It also needs to “inject” this data into the DeleteConfirmComponent class constructor to make it available for use.***

import { Component, OnInit} from '@angular/core';

import {**MAT\_DIALOG\_DATA**} from '@angular/material';

import { Inject } from '@angular/core';

@Component({

selector: 'app-delete-confirm',

templateUrl: './delete-confirm.component.html',

styleUrls: ['./delete-confirm.component.css']

})

export class DeleteConfirmComponent implements OnInit {

***Note: @Inject is defined as “ a constructor parameter decorator that specifies a custom provider of a dependency”.***

constructor**(@Inject(MAT\_DIALOG\_DATA) public data: any**) {}

ngOnInit() {

console.log(this.data);

}

}

**delete-confirm.component.html**

***At this point, it is a straight-forward exercise to use the injected data in the delete-confirm.component.html (notice the one-way binding to the data in the delete-confirm.component.ts).***

<body>

<div class="ui-dialog">

<mat-dialog-content class="ui-dialog-content">Are you sure you want to delete record #**{{data.dataKey}}** / **{{data.value}}**?</mat-dialog-content>

<mat-dialog-actions align="end">

<button class="btn" mat-button mat-dialog-close>CANCEL</button>

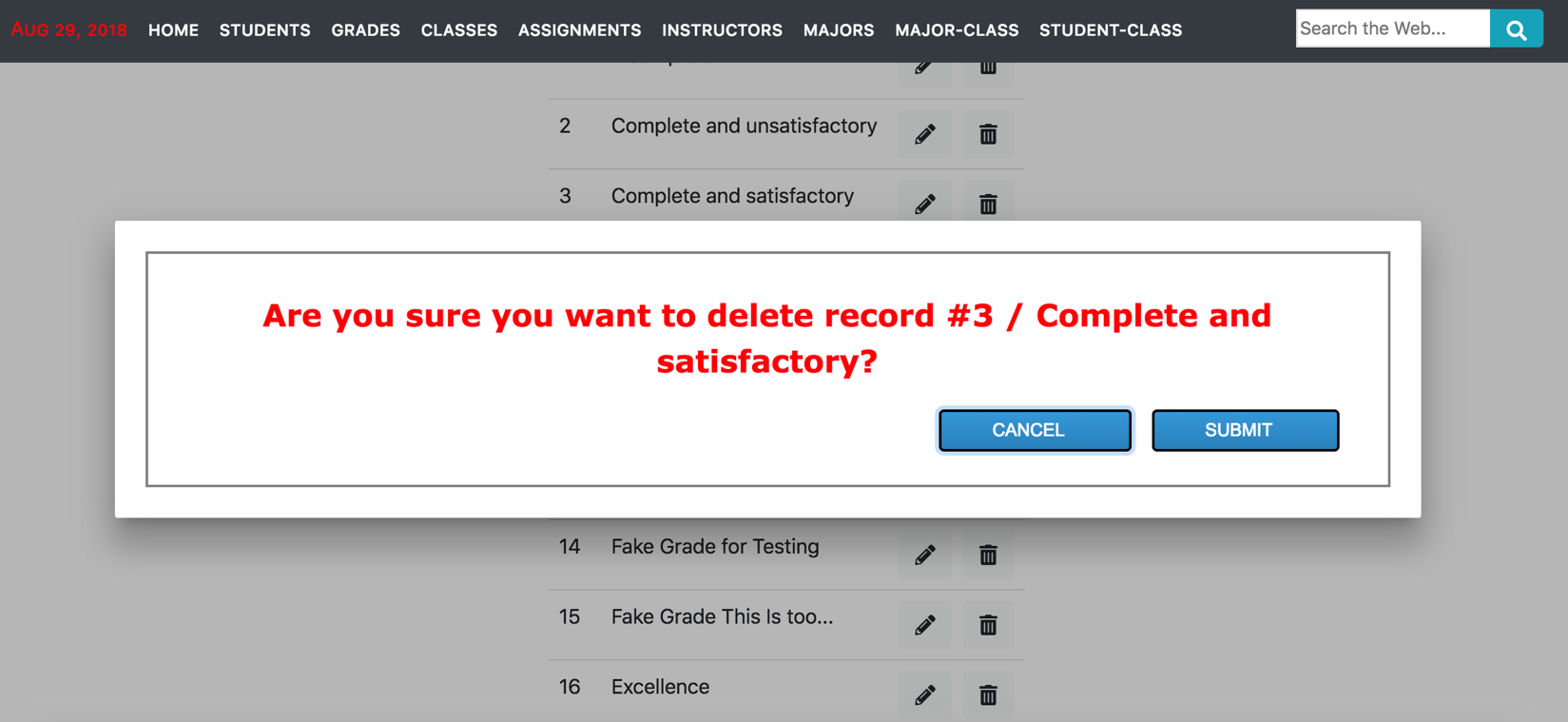
<button class="btn" mat-button [mat-dialog-close]="true">SUBMIT</button>

</mat-dialog-actions>

</div>

</body>

In the image below, you can see that the record id and a string containing the grades description was injected into the delete-confirm dialog box.



**< Addendum />**

1. **Links**
2. The Angular 5 frontend git repository used by this document
   1. <https://github.com/techlozenge/student-frontend-angular-final>
3. Sails Frontend and Backend git repositories
   1. Backend – Listens on port 1337
      1. <https://github.com/techlozenge/sails-student-frontend-v2>
   2. Frontend – Listens on port 1338
      1. <https://github.com/techlozenge/sails-student-api>
4. Draw.io is a free service which is the equivalent of MS Visio
   1. <https://www.draw.io/>
5. Iterm2 is a terminal emulator for MacOS
   1. <https://www.iterm2.com/>
6. **Terminology**
   1. **Interpolation** - You use interpolation to weave calculated strings into the text between HTML element tags and within attribute assignments. e.g. {{currentCustomer.name}} or <p>The sum of 1 + 1 is {{1 + 1}}</p>
   2. **Idempotence -** An [idempotent](https://en.wikipedia.org/wiki/Idempotence) expression is ideal because it is free of side effects and improves Angular's change detection performance. In Angular terms, an idempotent expression always returns *exactly the same thing* until one of its dependent values changes.
   3. **Safe Navigation Operator ?.** e.g. {{currentCustomer**?.**name}} - A convenient way to guard against null and undefined values in property paths. If the safe navigation operator weren't use the TypeScript would throw an error and in many cases the host view (i.e. your component) wouldn't display.

[**Back to Top**](#SystemStack)