Angular Training Course Content

- Introduction
  - Angular Introduction

    Angular is a javascript framework developed by Misko.

Misko Hevery Brad Green

  - Difference between angular 1 and angular 2
    - Can write code in Typescript
    - Increase performance
    - Implement nested routing
    - Two way data binding is not supported
    - Digest cycle follow tree data structure, in angular 1 digest cycle follow cyclic data structure.
  - Difference between angular 2 and 4
    - Exclude animation package
    - Implement AOT
    - Change structure of *ngIf
    - Reduce the size of router
  - Introduction of TypeScript

Data Types:

Boolean, number, string, array [], object {}, undefined

Classes

```
class Foo { foo: number; }
class Bar { bar: string; }

class Baz {
  constructor(foo: Foo, bar: Bar) { }
}

let baz = new Baz(new Foo(), new Bar()); // valid
baz = new Baz(new Bar(), new Foo());     // tsc errors
```

Inheritance is achieved by using extends

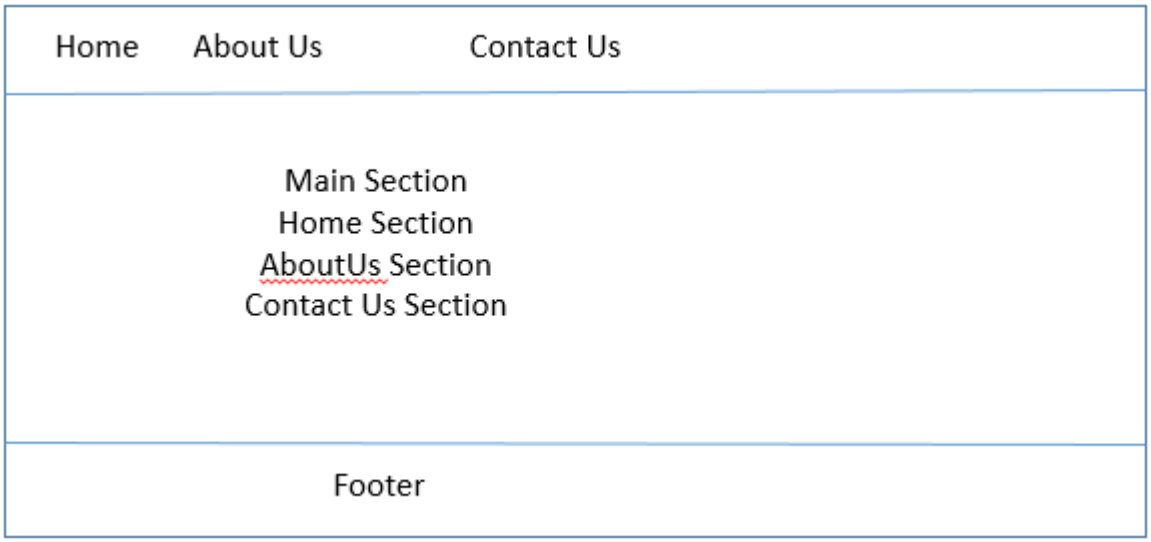
Class Foo extends Bar{}

Class Foo implements <interface>{}

Access modifiers

Private and public

Single Page application SPA



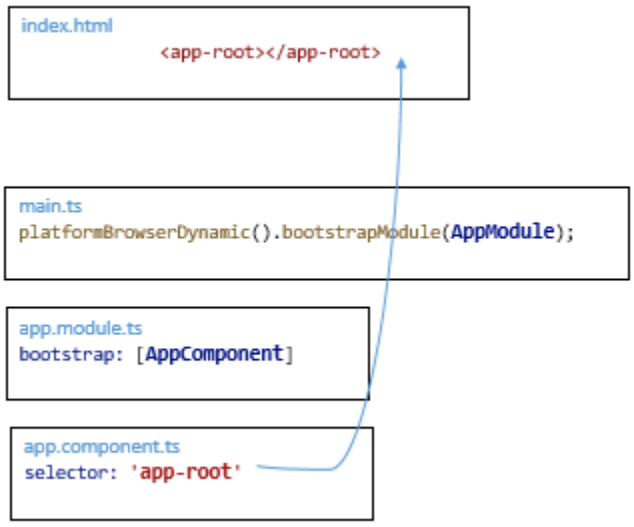
- Introduction to development environment
- Introduction to Node.js and NPM

Environment Setup

- Node.js
  - Download and Install node.js from https://nodejs.org/en/
- HTTP Restful api server for workshop
- Visual studio code IDE
  - https://code.visualstudio.com/download

Angular-cli

- Setup of angular-cli
  - Install angular-cli globally npm install -g @angular/cli
- Create basic project structure with angular-cli
  - Create project: ng new my-app
- How angular bootstrap process works



- Introduction of all angular-cli commands
  - Create new project
    - ng new <project name> i.e. ng new my-app
    - ng new my-app --prefix=my --style=sass i.e. prefix is used to create custom application prefix. Default prefix is app. Style is used to set the default style of project. By default style is css. We can set it to sass, less etc.
  - Create new component
    - ng g c <component name>
      - –flat: creates component in same directory
      - - - it : creates inline template
      - - - is : creates inline style
      - - - spec false : will not generate spec file
  - Create new service
    - ng g s <service-name>
  - Create new class
    - ng g cl <class-name>
  - Create new module
    - ng g m <module-name>
- How to add third party libraries i.e. Bootstrap
  - npm install bootstrap –save
  - Add following content into .angular-cli.json file

```
"styles": [
  "styles.css",
  "../node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```

Exercise:

- Create <my-app> project by using angular-cli.
  `ng new my-app`
- Create components
  `ng new header --spec false`
- Add bootstap library
- Launch application by using
  `ng serve --open`

- Introduction of components
  Components are the most basic building block of an UI in an Angular application. An Angular application is a tree of Angular components.

  - Create component with angular-cli: `ng new data-binding --spec false`
  - Component selector
  - Component templates
  - Component Styles
  - Access component into another component

  - Get data into component through Input
  - Subscribing to components events through Output
  - Component Life Cycle

Data Binding

- Interpolation
  - Create a component ng g c data-binding
  - {{title}}
- Property binding
  - [disabled] = “true”
- Event binding
  - (click)=”clickMe()”
- ngModel
  - <input [(ngModel)]=”name”>{{name}}

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

Directive

- Structure directive (*ngIf, *ngFor, *ngSwitch)
  - Create a new component ng g c structure-directive
  - ngFor supported parameters : index, even, odd, first, last
  - ngFor: trackBy is used for performance
- Attribute Directive (ngStyle, ngClass)

Exercise

- Create a component.
- Display your name by using data binding
- Display list of heroes with static array. Hero = [‘Misko’, ‘Brad’, ‘Ram’, ‘Sham’]
- Add new hero into this list by using two way data binding
- Use ngClass to apply css on alternate rows of heroes.

Forms

- Template driven approach

Include Formsmodule into app.module.ts file

Create a component i.e. ng g c template-driven-form

Create a model class Hero

Initialize Hero model into component

Implement two way data binding to display data of model into html

```
<input type="text" class="form-control" id="name" required
[(ngModel)]="model.name" name="name">
```

Defining a name attribute is a requirement when using [(ngModel)] in combination with a form. Name property required by angular form to register the control with the form.

Add similar ngModel and name attribute with other form elements.

Update form tab

```
<form #heroForm="ngForm">
```

Submit the form with ngSubmit

```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
```

Disable the button if form is invalid

```
<button type="submit" class="btn btn-success"
[disabled]="!heroForm.form.valid">Submit</button>
```

-

Reactive approach

- Form Control
- Form Group
- Form Builder
- Validator

Add `ReactiveFormsModule` into app.module.ts file

```
import { ReactiveFormsModule } from '@angular/forms';


imports: [
. . .,
ReactiveFormsModule
],
```

Create a reactive-form-component.ts ng g c reactive-form

Update component.ts file

```
heroForm = new FormGroup ({
```

```
name: new FormControl()

});

onSubmit(){
console.log('heroForm -- ', this.heroForm.value);
}
```

formControl accepts 3 optional arguments: the initial data value, an array of validators, and an array of async validators.

Update template file

```
<form [formGroup]="heroForm" novalidate>
    <div class="form-group">
        <label class="center-block">Name:
            <input class="form-control" formControlName="name">
          </label>
    </div>
    <button (click)="onSubmit()">Submit</button>
</form>
```

```
export class HeroDetailComponent3 {
  heroForm: FormGroup; // <--- heroForm is of type FormGroup

  constructor(private fb: FormBuilder) { // <--- inject FormBuilder
    this.createForm();
  }

  createForm() {
    this.heroForm = this.fb.group({
      name: '', // <--- the FormControl called "name"
    });
  }
}
```

-

Validation

State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

- Add one variable to see classNames

```
<input type="text" class="form-control" id="name" required
[(ngModel)]="model.name" name="name" #spy> <br>TODO: remove this:
{{spy.className}}
```

- Add following css into style.css

```
.ng-valid[required], .ng-valid.required {
border-left: 5px solid #42A948; /* green */ }
.ng-invalid:not(form) { border-left: 5px solid #a94442; /* red */ }
```

- Show and hide validation error message

```
<label for="name">Name</label>
<input type="text" class="form-control" id="name" required
[(ngModel)]="model.name" name="name" #name="ngModel">
<div [hidden]="name.valid || name.pristine" class="alert alert-danger">
Name is required </div>
```

- To display error we need a template reference variable to access the input box's angular control form within the template. `#name="ngModel"`

Services

- Introduction
  - Services are singleton instance at module level
  - ng g c <service-name> i.e. ng g c hero
  - update module.ts – provider section
- Service Dependency
  - Initialize service into module provider section
- Dependency Injection
  - Use service into component constructor.
  - constructor(private service : HeroService){}
- Injectable Decorators
  - This decorator is required if service has dependency on any other element. But good practice always use @Injectable() metatag
- Provider Definition
- Singleton vs multiple service instance

- Http
  - Introduction
    - It is used to communicate rest services.
    - It exists into @angular/http package.
  - Dependency Injection of Http service
  - Http CRUD operation: Create, Read, Update and Delete

```
getHeroes(){
  return this.http.get(this.heroUrl)
                .map(res => res.json().data)
}
```

  - Subscribe that method

```
service.getHeroes()
.subscribe(
  (response) => {
    // Success response
  },
  (err) => {
    // error response
  },
  () => {
    // Finally block
  }
)
}
```

- Pipes
  - Pipes Introduction
  - Built in pipes (Currency, Date, Number, Percentage, JSON)
  - Creating custom pipes

- Passing argument to pipes
- Registering pipes
- Chaining Pipes
- Http web service call with Pipes
- Async pipes
- Handling Promises in pipes

Angular pipes, a way to write display-value transformations that you can declare in your HTML.

```
Pipe -> birthday.component

birthday = new Date(1988, 3, 15); // April 15, 1988


<p>The hero's birthday is {{ birthday }}</p>
<p>The hero's birthday is {{ birthday | date }}</p>
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }}</p>
<p>The chained hero's birthday is {{birthday | date:'fullDate' |
uppercase}}</p>
```

The hero's birthday is Fri Apr 15 1988 00:00:00 GMT+0530 (India Standard Time)

The hero's birthday is Apr 15, 1988

The hero's birthday is 04/15/88

The chained hero's birthday is FRIDAY, APRIL 15, 1988

Create a custom sample pipe

```
ng g p <pipename>
```

```
ng g p exponential-strength
```

```
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }

<p>Super power boost: {{2 | exponentialStrength: 10}}</p>
```

```
Output
```

Super power boost: 1024

- Rxjs
    - Introduction to Reactive extension Library (RxJs)
    - What are Observables
    - Subscription
    - Events: map
    - Streaming in Observable
    - Cancellable Observable
- Routing
    - Introduction to Single Page Application SPA
    - Route Introduction
    - Configuring Routes
    - Linking Routes
    - Redirection
    - Router Outlet
    - Nested Routes
    - Lazy loading
    - Passing values parameters between routes
    - Can Activate Guard
    - Can deactivate Guard

Use RouterModule in app.module.ts file and create routes constant.

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


const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'aboutUs', component: AboutUsComponent },
  { path: 'lazyModule', component : "./lazy/lazy.module#LazyModule"}
];
```

Imports : [

```
...,

RouterModule.forRoot(appRoutes)
]
```

Update main html file.

```
<div class="container">
  <nav class="navbar navbar-default">
```

```
    <ul class="nav navbar-nav">
       <li>
   <a routerLink="" routerLinkActive="active">Home</a>
</li>
      <li>
         <a routerLink="/aboutUs" routerLinkActive="active">About us</a>
      </li>
            <li>
                <a routerLink="/componentInteraction"
routerLinkActive="active">Component Interaction</a>
            </li>
   </ul>

    </nav>

    <router-outlet></router-outlet>

</div>
```

Barrel

Export all components in index.ts file

Unit Testing

- Introduction to Test Driven Development (TDD approach)
- Introduction to Jasmine and Karma

Technology	Purpose
Jasmine	The Jasmine test framework provides everything needed to write basic tests. It ships with an HTML test runner that executes tests in the browser.
Angular testing utilities	Angular testing utilities create a test environment for the Angular application code under test. Use them to condition and control parts of the application as they interact *within* the Angular environment.

Karma	The karma test runner is ideal for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes. This guide describes how to set up and run tests with karma.
Protractor	Use protractor to write and run *end-to-end* (e2e) tests. End-to-end tests explore the application *as users experience it*. In e2e testing, one process runs the real application and a second process runs protractor tests that simulate user behavior and assert that the application respond in the browser as expected.

- Jasmine Test suites, Specs and Expectations
  - karma.conf.js -> configuration file
  - describe -> test suite
  - it -> test case
  - beforeEach
  - Run test cases
    - ng test
    - ng test - - code-coverage
- Test Debugging
- Testing Component
  - TestBed : It creates and angular testing module
  - The `configureTestingModule` method takes an `@NgModule`-like metadata object. The metadata object can have most of the properties of a normal NgModule.
  - `TestBed.createComponent` creates an instance of
  - `TestBed.createComponent` creates an instance of AppComponent
  - `createComponent` method closes the current `TestBed` instance to further configuration.
  - The fixture provides access to the component instance itself and to the `DebugElement`
  - Use the fixture's `DebugElement` to `query` for the `<h1>` element by CSS selector

```
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));


  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
```

    - Test component with dependency
        - Create a folder component-with-dependency
        - Create a component ng g c welcome -is -it --flat
    - Testing Service
    - Testing with HTTP
    - Mocking HTTP Service
    - Testing Pipes
- Advanced Topics
    - Build process
    - Ahead of Time compilation
    - Just in time compilation
    - Authentication and Authorization
        - JWT token based authentication
        - Storing tokens into html storage
        - Authorization with angular (Can activate route/ Can deactivate route)