General Angular notes

Friday, June 29, 2018 3:18 PM

Directives - Typically added by using an attribute selector

Examples:

Create new angular project:
Ng new <project name>

Add/Remove Bootstrap:
Npm install --save bootstrap@version
Npm uninstall bootstrap

If/Else directive
```
<p *ngIf="serverCreated; else noServer"></p>
 <ng-template #noServer>
  <p>No server was created.</p>
 </ng-template>
```

ngFor:
(print app-server component one time for each element in the components servers array)
```
<app-server *ngFor="let server of servers"></app-server>
```

ngStyle / ngClass Directive:
```
<p [ngStyle]="{backgroundColor: getColor()}" [ngClass]="{online: serverStatus === 'online' }">
```
(getColor() is a typescript method defined in the component where this html code is contained)

These can be expressions too:
```
    <li class="list-group-item"
    [ngClass]="{even: number % 2 == 0 }"
    [ngStyle]="{backgroundColor: number % 2 !== 0 ? 'yellow' : 'transparent'   }"
    *ngFor="let number of evenNumbers"
     >
      {{number}}
    </li>
```

Two-Way Databinding Directive:
```
<input type="text" class="form-control" [(ngModel)]="serverName" />
```

Property/Event Binding:
```
<button class="btn btn-primary" [disabled]="!allowNewServer"
(click)="onCreateServer()">Add Server</button>
```

One Way Binding:
```
{{'Server'}} with ID {{ serverId }} is {{getServerStatus()}}
```

Custom Property/Event Binding:
(called from app.component.html)
```
<app-server-element *ngFor="let serverElement of serverElements"
[element]="serverElement">
</app-server-element>
```

In server-element.component.ts:

```
@Input() element: { type: string; name: string; content: string };
```

In app.component.ts:
```
export class AppComponent {
serverElements = [
{ type: "server", name: "testserver", content: "just a test!" }
];
}
```

This effectively binds the server-element.component.ts *element* property to the app.component.ts *serverElement* property. See lecture 62.

Local Element References - establishes a way to alias an input and use that input locally only on the *.html. Provides an alternative to 2 way databinding.

```
<input type="text" class="form-control" [(ngModel)]="newServerName">
Would Become:
<input type="text" class="form-control" #serverNameInput>
```

By creating a reference through virtue of the #<refName>, this name can be used anywhere else in the angular markup on the html file, but not in the typescript file. Also note that this reference isn't holding a value, but rather an HTMLInputElement object that has *a value* property.

```
<button class="btn btn-primary"
(click)="onAddServer(serverNameInput,serverContentInput)">Add Server</button>
```

Then on the *.ts file:

```
onAddServer(nameInput: HTMLInputElement, contentInput: HTMLInputElement) {
this.serverCreated.emit({
serverName: nameInput.value,
serverContent: contentInput.value
});
}
```

Whereas with the 2 way databinding, this would have been:

```
onAddServer() {
this.serverCreated.emit({
serverName: this.newServerName,
serverContent: this.newServerContent
});
}
```

And required propreties *newServerName* and *newServerContent* be declared on the component.

<ng-content></ng-content>

By default anything you put b/t the start & closing tags of an angular component, is lost. To change this, use the ng-content directive. See lecture 71

Event Emitting

To allow a child component to notify it's parent of an event happening, declare an @Output property of new EventEmitter<type>(). In the child component's html, have some event trigger a method call to the component. When the parent component declares the child in the html, specify the event emitter property in parenthesis and set it equal to a method on the parent with "$event" passed as the parameter. $event is a special parameter that will hold the value and type of whatever was emitted from the child component.

Child Html

```
<button class="btn btn-danger" (click)="removeItem()" >DELETE</button>
```

Child Component

```
…
@Output() onDelete = new EventEmitter<ListItem>();
…
removeItem() {
this.onDelete.emit(this.currentItem);
}
```

Parent Html

```
<list-item [currentItem]="item" (onDelete)="removeItem($event)"></list-item>
```

Parent Component

```
removeItem(item: ListItem) {
…do stuff…
}
```

View Child & Local References

A local reference refers to an element in the html denoted by #<name>. Once an element is marked in this manner it may be used anywhere else in the html by simply referencing <name>.

A ViewChild allows the ts component to have property references to the local references. Of note however is that the type of such a reference will be ElementRef.

Html

```
<div class="row">
  <div class="col-sm-5 form-group">
    <label for="name">Name</label>
    <input type="text" id="name" #nameInput class="form-control">
  </div>
  <div class="col-sm-2 form-group">
    <label for="amount">Amount</label>
    <input type="number" id="amount" #amountInput class="form-control">
  </div>
</div>
<div class="row">
  <div class="col-xs-12">
    <button class="btn btn-success" type="submit" (click)="AddIngredient()">Add</button>
```

Component

```
@ViewChild('nameInput') nameInputRef: ElementRef;
@ViewChild('amountInput') amountInputRef: ElementRef;

AddIngredient() {
  this.onAddIngredient.emit(
    new Ingredient(
      this.nameInputRef.nativeElement.value,
      this.amountInputRef.nativeElement.value));
}
```

Lifecycle Hooks Lecture 72

ngOnChanges - Called after a bound input property changes
ngOnInit - Called once the component is initialized
ngDoCheck - Called during every change detection run
ngAfterContentInit - Called after content(ng-content) has been projected into view
ngAfterContentChecked - called every time the projected content has been checked
NgAfterViewInit - called after the components view has been initialized
ngAfterViewChecked - called every time the view (and child views) have been checked
ngOnDestroy - called once the copmonent is about to be destroyed

ViewEncapsulation:

By default in angular, components are encapsulated so styling defined in the css of a component does not apply globally. This can be toggled by amending the @Component declaration in the .ts class as done below. Note the ViewEncapsulation must be imported from angular core.

```
@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.
})

export class AppComponent {

}
```

```
Emulated  (enum member) ViewEncapsulation.Emulat…
Native
None
ShadowDom
```

StackBlitz.com

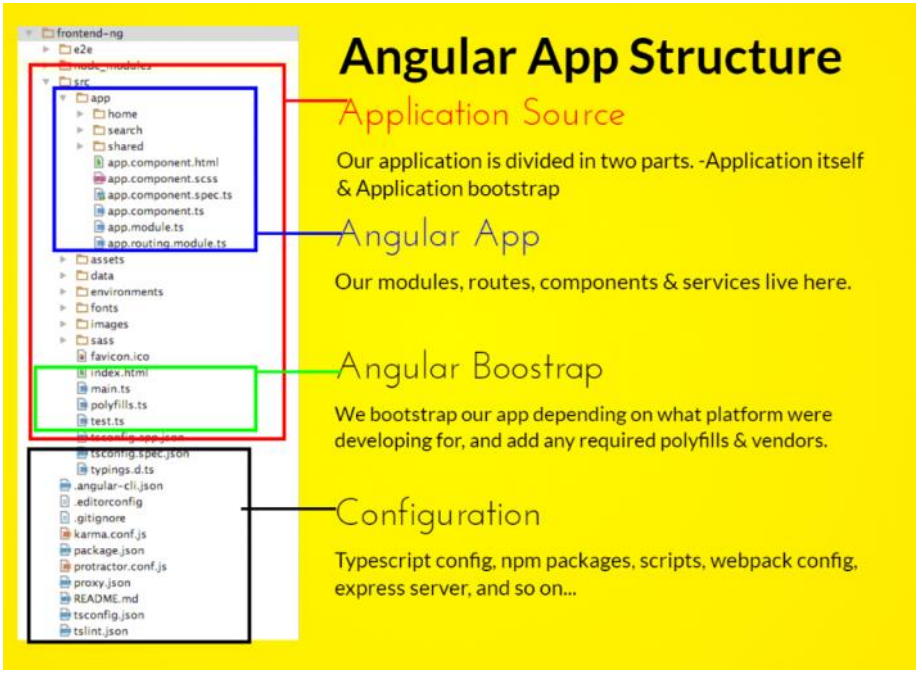
- Adding Bootstrap
  - Add a angular-cli.json file

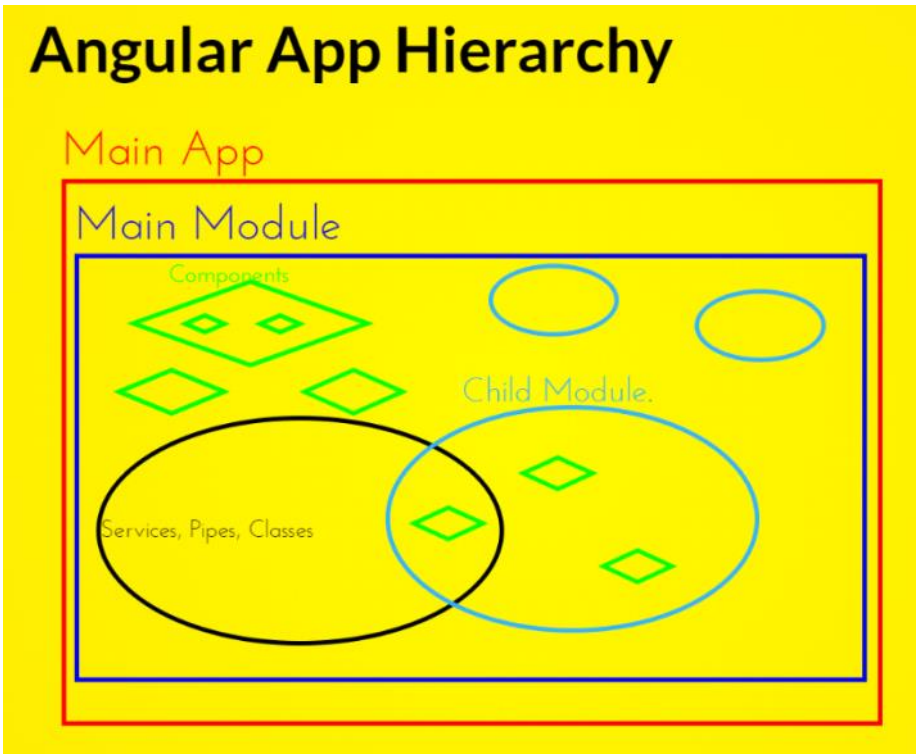
```
{
"apps": [{
"styles": ["styles.css"]
```

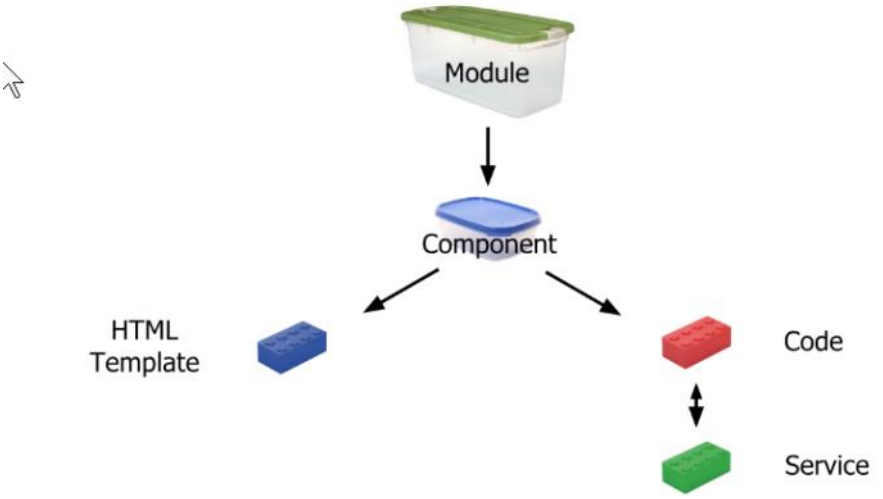
```
}]
}
```

- Add to style.css
  - `@import "bootstrap/dist/css/bootstrap.min.css";`

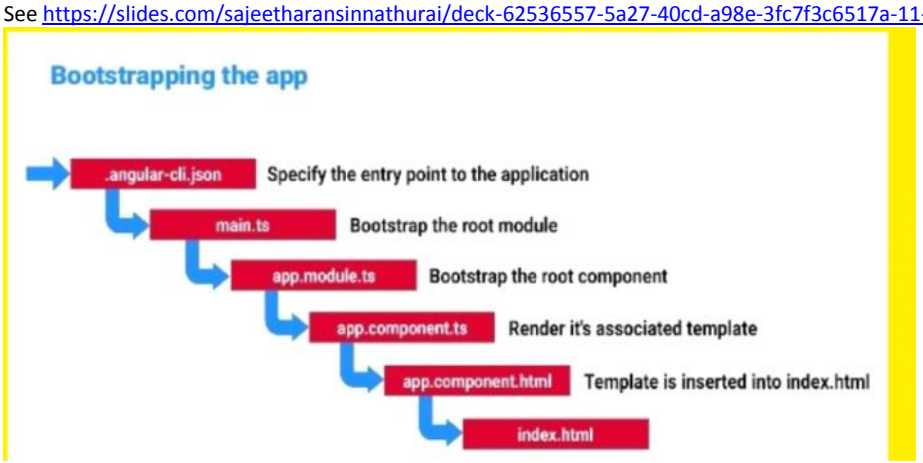
To pass data from outside into a component, import "Input" from angular core then create a component property prefixed with "@Input() ".







See https://slides.com/sajeetharansinnathurai/deck-62536557-5a27-40cd-a98e-3fc7f3c6517a-11-10#/8



@NgModule

Help organize an application into cohesive blocks of functionality.

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {FormsModule} from '@angular/forms';
import {HttpModule} from '@angular/http';

import {AppComponent} from './app.component';
import { HomeComponent } from './home/home.component';
import {AppRoutingModule} from "./app-routing.module";

@NgModule({
    declarations: [
        AppComponent,
        HomeComponent
    ],
    imports: [
        BrowserModule,
        FormsModule,
        HttpModule,
        AppRoutingModule
    ],
    providers: [/**DataService, UserService**/],
    bootstrap: [AppComponent]
})
export class AppModule {
}
```

This is Angular

The rest is a standard JS/Typescript class

@Component

Components are the most basic building block of an UI in an Angular application

```
import {Component, OnInit} from '@angular/core';
import {Router} from "@angular/router";
import {UserService} from "../shared/services/user.service";

@Component({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.scss']
})

export class LoginComponent implements OnInit {

    constructor(private us:UserService, private route: Router) {
    }

    ngOnInit() {
    }

    login() {
        this.us.login();
    }

    logout() {
        this.us.logout();
    }
}
```

Decorator

Annotate this is a Angula component

Angular Injects for us every dependency

Attribute Directives : ngStyle, ngClass, ...
Structural Directives : ngIf, ngSwitch, ngFor, ...

```
<div [ngStyle]="setStyles()">
  <p *ngFor="let player of players">...</p>
</div>
```

COPY

DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, PercentPipe ...

```
import { Pipe } from '@angular/core';
import { PipeTransform } from '@angular/core';
@Pipe({name: 'filterReviewByStatus'})
export class FilterReviewByStatusPipe implements PipeTransform {


}
```

```
<p>
    The chained mario's birthday is
    {{  birthday | date:'fullDate' | uppercase}}
</p>
```

Services

Components inject services

Service is a **class** that **encapsulates** some sort of **functionality** and **provides** it as a **service** for the rest of the application

```
export class UserService {
  private users: User[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger
  ) { ... }

  getAllUSers() {
    return this.users;
  }
}
```

Lfe cycle hooks

- Implemented by components and directives
- Methods which are called when specific events occur
- **ngOnChanges()** – called when data-bound input properties are set/reset
- **ngOnInit()** – called shortly after the component is created
- **ngOnDestroy()** – Called just before Angular destroys the directive component

Lecture 86 - 91 - creating own attribute directive

See https://stackblitz.com/edit/angular-section7-directives-deep-dive?file=src%2Fapp%2Fapp.component.html

This is done by creating a directive ts file. Import Directive. Provide a select for the @Directive({}) decorator of the export class. Via Angular's dependency injection, an ElementRef will be passed into the constructor which will allow you to meddle with the native element directly, though, this is a bad practice.

import { **Directive**, **ElementRef**, OnInit } from '@angular/core';

@**Directive**({
  selector: '[appBasicHighlight]'
})

```
export class BasicHighlightDirective implements OnInit {

  constructor(private elementRef: ElementRef) {

   }

  ngOnInit() {
    this.elementRef.nativeElement.style.backgroundColor = 'green';
  }
}
```

Instead use the Renderer2 class. (https://stackblitz.com/edit/angular-section7-directives-deep-dive)

```
import { Directive, Renderer2, OnInit, ElementRef, HostListener, HostBinding, Input } from
'@angular/core';

@Directive({
  selector: '[appBetterHighlight]'
})
export class BetterHighlightDirective implements OnInit {
  @Input() defaultColor: string = 'transparent';
  @Input('appBetterHighlight') highlightColor: string = 'blue';
  @HostBinding('style.backgroundColor') backgroundColor:string = this.defaultColor;

  @HostListener('mouseenter') mouseover(eventData: Event ) {
    this.backgroundColor = this.highlightColor;
    // this.renderer.setStyle(this.elRef.nativeElement,
    //  'background-color',
    //  'blue'    );

  }

  @HostListener('mouseleave') mouseleave(eventData: Event ) {
    this.backgroundColor = this.defaultColor;

    // this.renderer.setStyle(this.elRef.nativeElement,
    //  'background-color',
    //  'transparent'    );
  }

  constructor(private elRef: ElementRef, private renderer: Renderer2) { }

  ngOnInit() {
    this.backgroundColor = this.defaultColor;
    // this.renderer.setStyle(this.elRef.nativeElement,
    //  'background-color',
    //  'blue'    );
  }
```

With html:

```
<p [defaultColor]="'yellow'" [appBetterHighlight]="'red'">style me with a better
directive </p>
note the ommitted square brackets and single quotes. Caveat: ensure the directive
name is clear so that it's not confused with an actual html element name
<p [defaultColor]="'yellow'" appBetterHighlight="red">style me with a better
directive </p>
```

For custom event based HostListener example: https://stackblitz.com/edit/angular-customevent-hostlistener

Lecture 92 - Structural Directive behind the scenes

The "*" in *ngFor means the directive is a structural directive. This will embed the element generated in an
<ng-template> element.

Lecture 94

```
<div [ngSwitch]="value">
<p *ngSwitchCase="5">Value is 5</p>
<p *ngSwitchCase="10">Value is 10</p>
<p *ngSwitchCase="100">Value is 100</p>
<p *ngSwitchDefault>Value is Default</p>
</div>
```

Where "value" is a property on the component. Better alternative to custom structure directive (see appUnless on stackblitz) or multiple if cases.

Lecture 95 - This directive adds or removes the 'open' class on the element the directive is applied.

```
@Directive({
selector: '[appDropdown]'
})
export class DropdownDirective {
@HostBinding('class.open') isOpen = false;
@HostListener('click') toggleOpen() {
this.isOpen = !this.isOpen;
}
constructor() {
}
}
```

Lecture 96 - services and dependency injection

Services centralize code for use by one or more components

See on stackblitz: https://stackblitz.com/edit/angular-section9-services?file=src%2Fapp%2Fapp.component.html

Do not use services through instantiation as Angular has a better way. Ex:

```
const service = new LoggingService();
this.service.logStatusChange(accountStatus);
```

Instead use Angular's Dependency Injection by providing a "private" property of the service type in the constructor. Add the service to the array of Providers in the @Component declaration and import the service itself. (**lecture 99**)

```
import {LoggingService} from '../logging.service';

@Component({
 selector: 'app-account',
 templateUrl: './account.component.html',
 styleUrls: ['./account.component.css'],
 providers: [LoggingService]
})

constructor(private loggingService:LoggingService) {}
```

Lecture 101 Hierarchical injector

Lecture 103 Injecting services into services

Import the service you want to inject into the service where it is being injected. Add a private local variable for the service through the services constructor. Mark the service class with '@Injectable()'

```
// Add this to the service where you want something injected (the logging service
in this case)
@Injectable()
```

Lecture 104 Using services for cross component communication

Very helpful lecture on using services across components. Worth watching more. The idea is you add an eventemitter to the service. One component using the service could emit an event through it, while another subscribes to the services event emitter.

Lecture 106 services for project app
Shopping List & Recipe will get Services created

Lecture 117 Routing

Initializing routes in app.module.ts:

```
import {Routes, RouterModule} from '@angular/router';
const appRoutes: Routes = [
{ path: '', component: HomeComponent }, //localhost:4200/
{ path: 'users', component: UsersComponent }, //localhost:4200/users
{ path: 'servers', component: ServerComponent } //localhost:4200/servers
];
@NgModule({
declarations: [
AppComponent,
HomeComponent,
UsersComponent,
ServersComponent,
UserComponent,
EditServerComponent,
ServerComponent
],
imports: [
RouterModule.forRoot(appRoutes),
BrowserModule,
FormsModule,
HttpModule
],
```

Lecture 118

Creating links that use routes:

The router-outlet directive must be present on the page so Angular knows where to load the content.

```
<router-outlet>
</router-outlet>
```

Then
define links using routerLink.

```
<li role="presentation" class="active" ><a routerLink="/">Home</a></li>
<li role="presentation"><a routerLink="/servers">Servers</a></li>
<li role="presentation"><a [routerLink]="['/users']">Users</a></li>
```

Lecture 119

Note if you omit the '/' in the value of a routerLink, whatever value is listed will be appended to the current url.

So clicking `<a routerLink="servers">Servers</a>` from the home page will load localhost:4200/servers but if while on that page this same link was present, it would load localhost:4200/servers/servers.

Can also preface routerLink values with "../" which means go up 1 level

Lecture 120 - styling active link

Use 'routerLinkActive' to apply a css class to the active link. In the case of the root route, options must be applied so it doesn't look as though it's always selected.

```
<li role="presentation"
routerLinkActive="active"
[routerLinkActiveOptions]="{exact:true}"
>
<a routerLink="/">Home</a></li>
```

Other links:

```
<li role="presentation" routerLinkActive="active" ><a routerLink="servers">
Servers</a></li>
<li role="presentation" routerLinkActive="active" ><a
[routerLink]="['/users']">Users</a></li>
```

Lecture 121 - programmatic routes

```
<button (click)="onLoadServers()">Load Servers</button>
```

*.ts file :

```
constructor(private router:Router) { }

onLoadServers() {
//complex calculation , etc , etc
this.router.navigate(['/servers']);
}
```

Lecture 122 Relative Paths in Programmatic Navigation

Recall that relative paths are ones that omit a '/'.

```
import { Router, ActivatedRoute } from '@angular/router';
 constructor(private serversService: ServersService,
        private router:Router,
        private route:ActivatedRoute) { }
 onReload() {
   this.router.navigate(['servers'], {relativeTo:this.route} );
 }
```

Lecture 123 passing params to routes

```
{ path: 'users/:id', component: UsersComponent }, //localhost:4200/users
```

Lecture 124 fetching route parameters

```
import { Router, ActivatedRoute } from '@angular/router';
constructor(private route:ActivatedRoute) {
```

```
 }

ngOnInit() {
  this.user = {
    id: this.route.snapshot.params['id'],
    name:  this.route.snapshot.params['name']
  }
}
```