**Directive Vs Component** https://stackoverflow.com/questions/32680244/directive-vs-component-in-angular

We write a component when you want to create a reusable set of DOM elements of UI with custom behavior. Write a directive when you want to write reusable behavior to supplement existing DOM elements.

directives are of 3 types:

- **Attribute** Do not change the **structure** of DOM –[ngStyle] (the sq braces are for property binding), [ngClass] they are named so because they look like normal html attribute. they just change the look and feel for the element they are placed on.
- **Structural** –*ngIf *ngFor- changes DOM but they don't have their own view.
- 
```
<div *ngFor="let logItem of log">{{ logItem }}</div>
```
- **Component** - A component, not just add/modify behavior, actually creates its own view that is why it must have template and selector attached to it. Components usually have @input and @Output+EventEmitter<EmpDetail>

Important directives

*ngIf – structural. Add/remove element from DOM not just hide

```
<p *ngIf="serverCreated; else noServer">Server was created, server name is {{
  serverName }}</p>
<ng-template #noServer>
  <p>No server was created!</p>
</ng-template>
```
Marker directive. It is marked with #noServer

**Node.js** is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. JavaScript now has the capability to do things that other scripting languages like Python can do.

**Angular CLI** is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.

**WebPack**: bundles the complete app. Angular CLI uses it instead of systemjs
**primeng** : angular components
**Bootstrap:** for css
**Testing** karma (jasmin)

**angular.json:**
- main ts and html file.
- Specifies the location of src folder
- Compiler output folder
- Boot strap css and other external css
- Prod build settings like aot, source map, prod env file

**tsconfig.json:** type script configuration
**package.json:** project compile and dev dependencies

**Install ng/cli**: npm install -g @angular/cli
**New App**: ng new <app name>

**New Component**: ng generate component <component name> or ng g c < component name >
**Run app** cd <app name> & ng serve http://localhost:4200/
**New component**: `ng generate component <component name>`
ng build --prod --base-href .
**Polyfills** in angular are few lines of code which make your application compatible for different browsers. The code we write is mostly in ES6(New Features: Overview and Comparison) and is not compatible with IE or firefox and needs some environment setups before being able to be viewed or used in these browsers.

**Angular app Structure:** **https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655700?start=240#questions**
**https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655726#questions**
1. In angular.json we provide the reference of main.ts file

```
"index": "src/index.html",
"main": "src/main.ts",
```

2. In main.ts we boot strap our module "AppModule" (only one module here)

```
      platformBrowserDynamic().bootstrapModule(AppModule)

  .catch(err => console.error(err));
```

3. We can have multiple modules (which is prerequisite for lazy loading) or just one module "AppModule" looks like below

@NgModule({

```
declarations: [
    AppComponent – Our application components
  ],
  imports: [
    BrowserModule, - Angular modules
    AppRoutingModule,
    MyRouteModule
  ],
  providers: [], - Services
  bootstrap: [AppComponent "usually one bootstap"] – The List of components that
angular must be aware of before it reads inxed.html
```

})

4. The AppComponent has the selector let's say "selector: 'app-root'" that goes in index.html

**Other Popular decorators from '@angular/core'**

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

OnInit OnDestroy

`import { FormsModule } from '@angular/forms'- for two way data binding. It is required for [(ngModel)] to work`
**`npm install primeng -save`**: `--save add it to package.json`

**App Dependencies:** package.json
dependencies &
devDependencies – Development tools like cli, jasmin, tslint, typescript. These are not required by app to run but developers need them do develop efficiently
**package-lock.json** : When I publish my source code (usually without node modules) then this file ensures that all dependencies that gets installed on any compute are exactly the same as specified in lock file and not the latest one if available at that timehttps://www.youtube.com/watch?v=H3n75nHN5qY

**tsconfig.json**: TypeScript compiler configuration.
**tsconfig.app.json**: it extends tsconfig.json. It is the bootstrap file for application

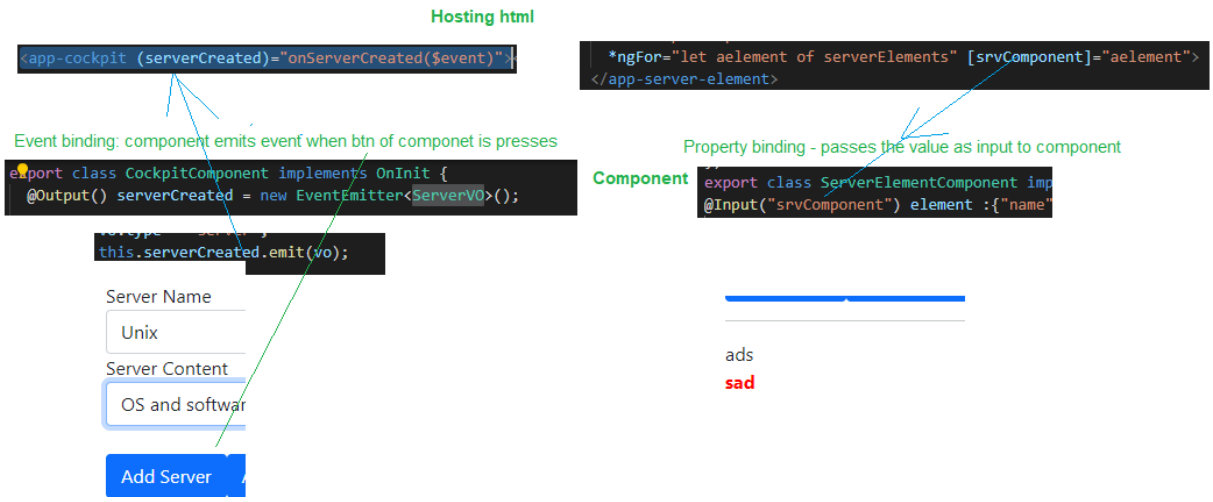
**TypeScript** : is typed version of java script. Like java we use class and interface. Type script is converted to javascript by angular cli
**strict mode**, you cannot, for example, use undeclared variables
**Module** is a package of components that we build or import from built in by angular.
***Components:*** allows to split the business logic/style in small reusable parts.

**Three ways to write selector in a component**

```
  selector: 'app-root',//To be used as element <app-root></app-root>
  //selector: '[app-root]' - attribute selector - <div app-root> <div>
  //selector: '.app-root' - to be used as class like <div class="app-root"></div>
```

**Data binding**:
**1. String interpolation** {{data}} Use this for output something in template
https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655804#overview
https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655814#overview
**2. Property binding** [property]= "data" – Array syntax Data from the host component to child component https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6655806#overview
**3. Event binding** (event)="expression" – function style data from child to host component
**4. Two way** [(ngModel)]="data"

**To pass a value to a component "@input property"**
**To pass value from component** "@Output event" using eventEmitter
Service: When component talk to each other




**#localReference in template (not available in ts) can be used as replacement for ngModel**

```
<!--<input type="text" class="form-control" [(ngModel)]="newServerName">-->
<input type="text" class="form-control" #newServerName>
```

```
(click)="onAddServer(newServerName.value)">
```

#localReference can also be fetched in TS via @ViewChild()

```
<!--<input type="text" class="form-control" [(ngModel)]="newServerContent">-->
<input type="text" class="form-control" #newServerContent>
```

In TS we use @ViewChild to get elementReference

```
@ViewChild('newServerContent', {static: true}) serverContentInput: ElementRef;
vo.content = this.serverContentInputA.nativeElement.value;
```

```
<ng-content></ng-content>: Inside the HTML of component of app-cockpit
<app-cockpit >passed to ng-content</app-cockpit>: From our hosting HTML
```

https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656100#overview

**angular Lifecyle hook**
**ngOnInit**
**ngOnDestroy** when *ngIf is called
**ngOnChanges** when there is a change in a property annotated with @Input()

**How to access dom elements:** use **Renderer2 & ElementRef –** Where angular is not running in browsers (server workers)
**https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656172#overview**

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

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

**@HostListener:** attach a listener to the events occurring in HTML where the component is hosted

```
@HostListener('mouseenter') mouseover(eventData: Event) {
```

**@HostBinding:** we can refer any property of the host element

```
@HostBinding('style.backgroundColor') backgroundColor: string = 'transparent';

@HostListener('mouseleave') mouseleave(eventData: Event) {
  // this.renderer.setStyle(this.elRef.nativeElement, 'ba
false);
  this.backgroundColor = 'transparent';
}
```

**The property setter** & custom structural directive https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656186#overview
The below code means whenever the aProperty (this is a property of component) is changed the function code gets executed.

```
@Input()  set aProperty($event){
console.log("prop change "+$event)
}
```

**structural directive**: In these directives we need reference of TemplateRef (what content) and ViewContainerRef (where)

---

**Services (helps avoiding duplicate code and data, & helps inter component communication) and DI**: different pieces communicate via services.
**To use a service in component:** Service is a simple VO class i.e. no decorator
Services are created in hierarchy. So if I have provider at appModule.ts then everything in app gets the same object.

1. add to the provider array in appMpodule or any of your componet `providers:[LoggingService]`

2. in constructor specify dependency of it `constructor(private logSrv: LoggingService){}`

3. Then use it `this.logSrv.logAccStatus("some text");`

**@Injectable():** You add to a Service class if that service is dependent on some other service

**Routing**: (outsource routes)https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656338#questions

```
'@angular/router';
```

Step 1. Add an array of Routes in appModule.ts or preferable in separate module

```
const appRoutes : Routes = [
  {  path:"Home/:id",component: HomeComponent}]
```

Step 2. In imports add

```
RouterModule.forRoot(appRoutes)
```

Step 3. In appcomponent.html
Simply add

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

Step 4 . Use routerLink in the html to route to the pages

```
<a routerLink="/servers" >Servers</a>
```

If you don't want to change the URL use skipLocationChange
https://www.youtube.com/watch?v=4aVXWmIRz7I&t=230s
Make router tab active CSS stuff https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656288?start=225#overview

Router as separate module: https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/14466488?start=255#questions
**Observables**: is like data store. It works on observable and observer pattern.

```
import {interval, Subscription} from 'rxjs';
private sub: Subscription
```

```
ngOnInit() {
    this.sub = interval(1000).subscribe(count => {
      console.log("Count: "+ count)
    });
  }
```

```
ngOnDestroy(){
    this.sub.unsubscribe();
  }
```

| Observables (rxjs feature) | Promises(native es6 feature) |
| --- | --- |
| Emit multiple values over a period of time. | Emit a single value at a time. |
| Are lazy: they're not executed until we subscribe to them using the subscribe() method. | Are not lazy: execute immediately aft creation. |

| | |
|---|---|
| Have subscriptions that are cancellable using the unsubscribe() method, which stops the listener from receiving further values. | Are not cancellable. |
| Provide the map for forEach, filter, reduce, retry, and retryWhen operators. | Don’t provide any operations. |
| Deliver errors to the subscribers. | Push errors to the child promises. |

Observer has : next, error and complete method. Promise has fulfil and error message

**Callbacks vs Promises vs RxJS vs async/await** : https://academind.com/tutorials/callbacks-vs-promises-vs-rxjs-vs-async-awaits/
https://youtu.be/jgWnccjXR4I

async await works with promises. The code looks synchronous(hence easy to understand) but it is asynchronous code indeed.

```
button.addEventListener("click", async () => {
  const isAuth = await checkAuth()
  let user = null;
  if (isAuth) {
    user = await fetchUser()
  }
  setText(user.name)
});
```

A promise demo (Native to ES6 – no imports required)

```
aPromiseFun = () => {
    return new Promise ( (resolve, reject) => {
      setTimeout(() => {
       resolve ({ name: 'Max' })
      }, 1000);
    });
};
```

```
//callPromomise and callPromomiseAwait does the same job
 callPromomise(){
   this.aPromiseFun().then( (data: dataVO) => {
   console.log("using regular call"+data.name)
   })

 }

 async callPromomiseAwait(){
  let data =  await this.aPromiseFun();
  console.log("using await "+data['name']);
 }
```

**Create an observable**

```
private CusObser : Observable<number>;
  private sub: Subscription
  ngOnInit() {
     this.CusObser = Observable.create( observer => {
      let count : number = 0;
      setInterval( () => {
        observer.next(count);
        count++;
        count++;
      },
        1000)
    }) ;

    this.sub =this.CusObser.subscribe(c => {
      console.log(c)
    })
  }
```

**Forms**: take values and submit values to server, form validations
**Template Driven Vs Reactive**
https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656462#questions

**Step 1.** Import "FormsModule" Add ngModel (As values will be passes as form object, so binding is not required) and "name" to the controls. This steps tell angular that what all the controls in my form

```
<input type="email" id="email" class="form-control" name="email"

 ngModel>
```

**Step 2.** Add ngSubmit to the form

```
<form ngSubmit >
```

**Step3** add a local reference "#f" and assign it the angular form and then pass f to the event listener of ngSubmit

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

**Step4** Now in the TS file refer it

```
onSubmit(form : NgForm){
 console.log(form.value["email"])
  }
```

**Reactive**

**Step1: import** "ReactiveFormsModule" and Create new formGroup

```
ngOnInit() {
    // Add specified validators
    this.addAssetForm = new FormGroup({
      'assetName': new FormControl(null, Validators.required),
      'assetCategory': new FormControl('Select a Category', Validators.required),
      'assetDescription': new FormControl(null, Validators.required),
      'dateOfPurchase': new FormControl(null, Validators.required),
      'assetCost': new FormControl(null,Validators.required)

    });
  }
```

**Step2**: Attach this form to the form in template

```
<form (ngSubmit)="addAsset()" [formGroup]="addAssetForm">
```

**Step 3:** Attach the formControl in html with TS

```
<input type="text" class="form-
control" id="assetName"  formControlName="assetName">
```

Step4:

**Alternative to get form** is @ViewChild. This approach is useful to get access to form even without submitting it

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

```
@ViewChild('f') signUpForm: NgForm;
  onSubmit(){
 console.log(this.signUpForm.value["email"])
  }
```

Form validation in CSS

```
input.ng-invalid.ng-touched {
  border: 1px solid red;
}
<input type="email" id="email" class="form-control" name="email"
            ngModel required email #userEmail="ngModel" >
            <span class="help-block" *ngIf="userEmail.invalid &&
userEmail.touched">Invalid email Address</span>
```

**Reactive Forms**: [https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656500#questions](https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656500#questions)

**Pipes**: to transform the output: {{data | uppercase | date:'ddMMyy':<anotheParameter> }}

```
<strong>{{ server.name | shorten:5 }}</strong>
```

```
@Pipe({
  name: 'shorten'
})
export class ShortenPipe implements PipeTransform {
  transform(value: any, limit: number) {
    if (value.length > limit) {
      return value.substr(0, limit) + ' ...';
    }
    return value;
  }
}
```

**Progress bar using pipe:** CusObser is a observable and angular will create a subscription to that we need to explicitly subscribe to that when we pipe with async.

```
Progress Status : {{CusObser | async}}
        <button type="button" (click)="getProgress()">Get progress</button>
```

```
private  CusObser : Observable<number>;

  getProgress(){
    this.CusObser = this.getProgressObser();
  }
  getProgressObser (){

    return Observable.create( observer => {
      let count : number = 0;
      setInterval( () => { //fake http call
        if (count >10){
```

```
          observer.complete();
        }
        observer.next(count++);

      },
        100)
    }) ;
  }
```

**Optimization:**
1. Split your application into feature module (function bases division)
https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/14466490?start=300#questions
2. Lazy loading

**2. ng build –prod** : AOT Ahead of time compilation that don't ships angular compiler with you application

**getter property:** We can use this function as a property in our template to get something

```
get ingredientsControls() {
  return (this.recipeForm.get('ingredients') as FormArray).controls;
}
```

```
*ngFor="let ingredientCtrl of ingredientsControls;
```

Deploy
1. ng build --prod --base-href .

Union type:

```
let course: string | number | boolean = 'React - The Complete Guide';

course = 12341;
```

Type alias

```
type Person = {
  name: string;
  age: number;
};

let person: Person;
```

**Rest parameter**

The rest parameter has to be the last argument, as its job is to collect all the remaining arguments into an array.

```
fun(...input){
    let sum = 0;
    for(let i of input){
        sum+=i;
    }
    return sum;
}
this.fun(1, 5,6)
```

**Spread Operator**: looks similar to rest parameter but works in opposite direction.
concat operation

let arr = [1,2,3];
let arr2 = [4,5];
arr = [...arr,...arr2];

```
function insertAtBeginning<T>(array: T[], value: T) {
  const newArray = [value, ...array];
  return newArray;
}

const demoArray = [1, 2, 3];

const updatedArray = insertAtBeginning(demoArray, -1); // [-1, 1, 2, 3]

updatedArray[0].split('');
```

**Interfaces Vs Type alias:**

1. No Merge in Type: Type is to declare a new type like number, string. So once declared we cannot change it at some other place it will give error like this

```
type A = {
  a: number
};

type A = {
  b: string
};
```

Where as in interface we can merge them like this

```
interface X {
  a: number
}
interface X {
  b: string
}
```

```
let i : X = {a:1, b:'interface '}
console.log(i)
```

2. We can have classes to extend interfaces but not type

```
type Human = {
  firstName: string;
  age: number;

  greet: () => void;
}

let max: Human;

max = {
  firstName: 'Max',
  age: 32,
  greet() {
    console.log('Hello!');
  },
};
```

```
interface Human {
  firstName: string;
  age: number;

  greet: () => void;
}

let max: Human;

max = {
  firstName: 'Max',
  age: 32,
  greet() {
    console.log('Hello!');
  },
};
```

**Route guard:**
**Enter** https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656342#questions
**Exit** https://www.udemy.com/course/the-complete-guide-to-angular-2/learn/lecture/6656346#questions
Running some functionality before entering / exiting a route

**Hash map in java script**
let map : { [key: string]: boolean} = {};
map["foo"] = true;