Deployment →

→ When we build our application using angular-cli with the ng build --prod angular will produce highly optimized (minification, uglification, bundling, dead-code elimination, AOT) bundles and then we can ~~the~~ simply deploy these files to a non-development machine.

⇒ ~~AOT~~ Angular-compilation →

An angular app consist largely of components and their HTML templates. Before the browser can render the application, the components & templates must be converted to executable JS by an angular compiler.

⇒ Angular offers two ways to compile your app →

① Just-In-Time (JIT), which compiles your app in the browser at runtime.

② Ahead-of-Time (AOT), which compiles your app at build time.

AOT converts our angular HTML & Typescript code into efficient JS code during the build phase. ~~before the~~

→ JIT compilation is the default when we do →

ng-build → no AOT, still (vendor.js → 2.24MB)
ng-serve

No optimization → mini, ugli, deadcode elimination

ng-serve bundled the app files but did not optimize our app

Only bundling

→ both command clears the output folder before they build the project. dist (default)

→ For AOT compilation, append the --aot

```
ng build --aot
ng serve --aot
```

→ compiles the appⁿ into an output directory.

✱ The --prod meta-flag compiles ~~the~~ with AOT by default.

⇒ How AOT works :- Two Phases

- Analysis – in which it simply records a represantation of the sorce.
- Code-generation –

Analysis →

tool for generating and developing angular appⁿ
→ Angular-Cli → A command-line interface for Angular

→ Angular-Cli does many thing → simplify below things for developer.

(1) → build, serve

(2) Unit & end-to-end tests → ng test
⇓
Add our test files the CLI uses Karma & protractor for running our test and it ~~works~~, pretty easy to customize too
⇓
ng-Test command builds the app in watch mode, and launches the Karma test runner

Test runner

Jasmine
⇓
Test framework in JS

(3) Lint :- (ng lint --fix) → fix easy issues
If we want to make sure our Code follows tslint standard, just run ng-lint and Angular cli will tell us where we failed in the rules.
Some std ⇒ "your forgot a space/ semicolon here".

(4) Generate Code :- Save precious minutes of development by generating our component, Pipe, service, directive, routes directly from the terminal

ng-optimize →

Webpack

→ Webpack is a static module bundler for modern javascript application.

→ when webpack processes our app, it internally builds a dependency graph which maps every module our project needs and generates one or more bundles.

⇒ Core Concepts

- Entry → default value ⇒ ./src/index.js
- output
- Loaders
- Plugins

but we can specify a different (multiple entry points also)

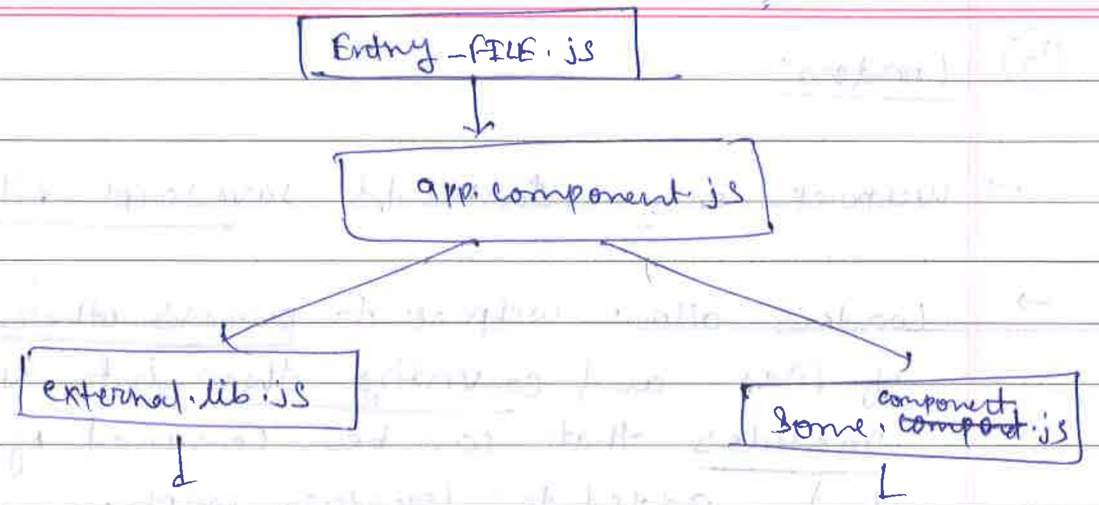
① entry → entry point indicates which module webpack should use to begin bundling out its ~~do~~ internal dependency graph.

official

• webpack will figure out which other modules and lib that entry point depends on (directly ~~or~~ Indirectly)

✱ The first javascript file to load to "kick-off" your app in the browser.

~~Webpack~~



```
module.exports = {
    entry: "./lib/index.js"
        :
  }
```

② Output :- This property tells angular where to ~~emit~~ keep the bundles it creates and how to name these files,

⇒ it's defaults → ./dist/main.js
⇓
for main output file

⇒ ./dist → folder for any other generated files

```
module.exports = {
    entry : './lib/index.js',
   output : {        './dist'
        path: ~~path.resolve~~  ~~__dirname + '/dist'~~
        filename: '/main.js',
        :
   },
   :
```

③ Loaders:-

→ webpack only understands javascript files.

→ loaders allow webpack to process other types of files and converting them into valid modules that can be consumed by our app and added to dependency graph.

→ At high level, loaders have two properties in webpack configuration:-

① test → this property identifies which file or files should be trasformed.

② use → this property indicates which loader should be used to do the trasforming.

```
module.export = {
    entry:    ,
    output: {    },
    loaders: [
      { test: /\.ts$/, loader: 'ts' },
      { test: /\.js$/, loader: 'babel' }
      {test: /\.css$/, loader: 'css' }
    ],
}
```

.js

app.component.ts

converts es6 d

.css — using css loader — .js

→ like above we can add loader for scss to convert it to css than js.

```
loaders: [
    {
      test: regex,
      loader: String,
      loaders: Array<string>,
      include: Array<regex>,
      exclude: Array<regex>
    }
]
```

test → A regular expression that instructs the compiler which files to run against ~~[illegible]~~ the loader.

loader → A string of the loader name you want to run.

loaders → An array of string representing the modules you want to run.
→ provides multiple loaders separated by '!'.
ex. 'style!css!less'

include:- Array of regex, instruct the compiles which folders/files to include
→ will only search paths provided with the include. → include: /some_dir-name/

exclude:- An array of regex, that instructs the compiler which folders/files to ignore. like we can ignore node modules, spec files.
→ exclude: [/\.spec[illegible]

```
ng serve --environment = test
```

Environments.

→ we use environments to setup differnt-differnt environments.
like production, devdopennent, QA.

→ In environments folder we can define
↓
environment.ts → for devlopemnt
environment.prod.ts → for prodution
environment.qa.ts → for testing.

→ Inside these files we can define multiple propedies like

production: False | True
↓ for non prodiction environmt | ↓ for production environmt

⇒ ~~we can define navigation bar color~~

⇒ we can define addition properties like →

For example you may want to change the color of the navigation bar depending on the environment
⇓
This way tester's know that they are looking at the actual testing websites not the production website, so they don't accidentally modify some data in production

```
'ng build --env = prod'
```

→ or we may want to change the name of the application in navigation bar, you may want to add the word testing

→ or perhaps we want to use a different API end points in testing environment

→ we can add all these properties in this environment object.

Ex. environments
- environment.ts
- environment.prod.ts

```
export const environment = {
        production: False,
        navBarBackground color : 'blue'
};
```

```
export Const environment = {
        production: true;
        navBarBackground color: 'Green'
};
```

→ when we run our app with angular cli and apply the environment flag, angular-cli pick one one of those environment file and put it in our bundle
⇓
So we don't have to write any code to work with a specific environment object.

Ex. ng build --prod or ng build

→ Angular-cli -picks correct envirment file while compiling either Prod or qa
we have to import normal environment file.
↑ environment file while compiling angular cli picks correct file

navbar.component.ts

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

export class NavbarComponent {

    backgroundColor = environment.navBarBackgroundColor

}
```

Bind this color in UI

→ If we are ~~g~~ creating some ~~to~~ custom environments like staging & qa then we have to create a environment file with environment object inside it like other production & development environments.

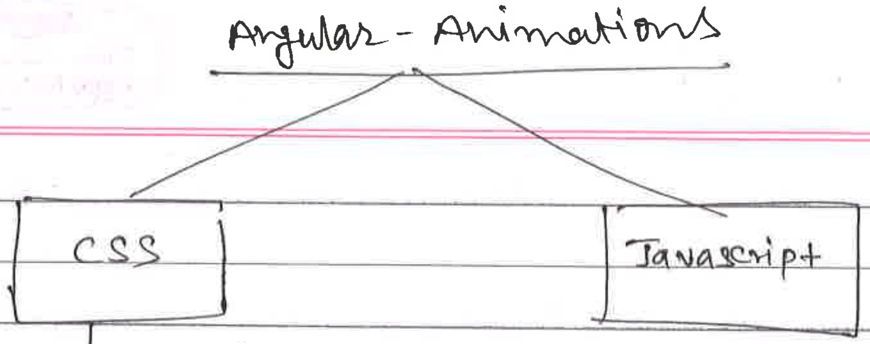
→ After this we have to add this file in angular-cli.json inside

```
"enviroments": {

    "dev": "environments/environment.ts",
    "prod": "  ———————————  prod.ts",
    "staging": "  ———————————  stagg.ts",
    "qa" : "  ———————————  qa.ts"
}
```

* For these environment files or object we don't have hot module replacement feature of webpack

→ our changes will not be visible immediatley.



css properties — transition
with this we — animation
can animate dom element

⇓

```
.streech {
  animation-name: streeh;
  animation-duration: 1.5s;
  ....
}
```

⇒ css based library → animate.css

↓

no need to do manually

animation. provides pre-defined animation classed.

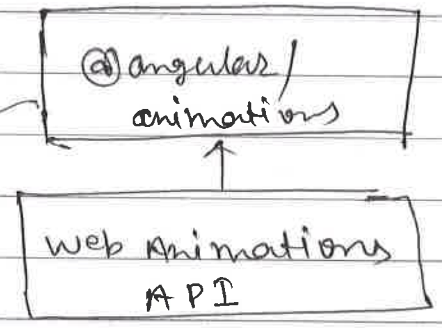
→ Install this lib by npm and use.

→ impot it in styles.css same we did for bootstrap.

problem:- · limited control
· suitable for simple, one-shot ani--

→ Javascript animation:— · jQuery
· GSAP
· zepto
Recommanded → · Web Animation API

→ Angular also has animation support through @angular/animations module. It is built upon web Animation API.



benefit:- our code is going to be easier to test and easier to post to a differ platform

→ So when we code against these abstractions we can take our code and run it inside an ios or an android environment and use animation natively in that environment.

→ So we are not tightly coupled to the implementation of web animation in browsers.

→ @angular/animations has some helpful fn
- trigger()
- trasition()
- State ()
- animate()

material design which is a visual language developed by google in 2014

Angular material :-

This is the same language we see in gplus, Android and many other apps.

→ A library of high-quality UI components built with Angular and Typescript.

→ Modern UI Components that work across the web, mobile, desktop.

- Internationalize → so users with differ lang. can use them
- clean & simple API
- well tested
- customizable
- fast → very fast & minimal performance overhead.
- well documented.

→ so these material controls (like checkbox etc) have the same API as the native checkboxes we have used so far but they look very pretty and they also have some nice animation.

Angular material	Bootstrap
• still new (and imature)	more mature (for complex UI needs)
• same quality std. (look n feel)	
• common API	
• Easy to use	• A lot of dependencies
	• we have to use 3rd-party components.

Redux - code

app.module.ts

```
import { IAppState , rootReducer, INITIAL_STATE }
        from '.|reduxfiles| store' ;


export class AppModule {
    Constructor ( ngRedux : NgRedux <IAppState>) {
        ngRedux . configureStore (rootReducer,
                                  INITIAL _ STATE );
    }
}
```

Store . js

```
export  interface  IAppState  = {
     counter : number
}

~~export function rootReducer ( state~~
export  const  INITIAL _ STATE : IAPPSTATE = {
     counter : 0
}

export function rootReducer ( state : IAppState , action)
                                              {Inmstnn}
   switch ( action . type ) {
      Case   INCREMENT :
          return  { 'counter' : state . counter +1 };

     Case  DECREMENT :
         return  tassign ( state , { counter : state .
                                   Counter ~~+~~ - 1 })
   }
   return state;
}
```

Style
Date:.....................
Page No:.................

actionis →

```
export cont INCREMENT = "INCREMENT";
———— DE———— = "DE————";
```

Dispaching → data

```
this.ngRedux.dispatch ({type: INCREMENT})
———— ( ———— DECREMENT})
```

Reading the data from store →

3ways

→ Can assign an alias

```
① @Select ( ) counter

② @Select ([ 'messaging', 'newmessage' ]) newmessage;

③ @Select ((s: IAppState) => s.messagin.newMessage})
        newmessage1;
```

Reading a slice of the object

Object in store :-

```
export interface IAppState {
    counter: number;
    messaging: {
        newMessage: number;
    }
}
```

cross - component communication

First.component.ts:

```
// method: 1 - using   EventEmitter's
    ShareData () {
        this.eventEmitterService.statusupdated
              .emit ( this.StrValue );
    }

// method 2: using  BehaviourSubject

    this.behaviourSubService. changeMessage (this.message)

// method 3: using  Subject Observable

    this.subjectser.changeMessage (this.message);
```

→ eventEmitter.service.ts:-

```
export  class  EventEmitterService {
          Statusupdate = new EventEmittel <String>();
}
```

→ behaviourSubject.service.ts →

```
export class   BehaviourSubjectService {
     private messageSource = new BehavioursSubject("Def msg")
currentmessage   = this.messageSource.asObservable();

 changemessage (message: String) {
          this.messageSource.next(message);
 }
```

→

```
private  messageSource  = new  Subject <any> ();
currentMessage   = this.messageSource.asObservable
                                                 ();
ChangeMessage (message: string) {
```

second.component.ts →

```
// method:1 → subscribe  eventEmitter

  this.eventEmitterService.statusUpdated.subscribe(
       (data: string) =>   this.strValue =data;
  );


// method:2 → subscribe. BehaviourSubject

  this.behaviourSubjectService.currentMessege. subscribe(
       message => {
              this.message = message;
  })


// method:3 →

  this.subjectService.currentMessage. subscribe (
         message => {
               this.message = message;
  });
```