Angular Advantages:

* Powerful CLI
* Typescript support
* DI
* Change detection
* All in solution – routing, forms etc

Angular disadvantages

* More learning curve
* Knowledge of RxJS
* Bundle size

Module is just a package. It is used to bundle different components into packages. It has @NgModule decorator like component has @Component.

@Component takes in selectors, templates, styles and @NgModule takes in declarations, imports, providers, bootstrap.

**@NgModule –** declarations would include the components, imports will have modules,Bootstrap contains the component which the app should be aware of when it starts, in other words which is part of index.html

**@Component**

Main.ts contains the first module (mostly AppModule) to be bootstrapped/started

***platformBrowserDynamic***().bootstrapModule(AppModule)

Data Binding is how view and component communicate with each other

A screenshot of a computer

Description automatically generated with medium confidence

[] means property binding. {{}} means string interpolation. Example [disabled] = “allowNewServer”. When you want to output something in the template use String interpolation and when you want to change some html property or directive or component use property binding. **String interpolation is when data flows from component to view.**

(click)=”onCreateServer()” inside () denotes event binding in angular and is equivalent to onClick. (input) event on input text box is triggered when something is input

ngModel which helps in two way binding is part of FormsModule. [(ngModel)]=”serverName”. You can send data from component from view but when some event happens you can send data from view to component

Directives help to attach behavior to HTML DOM. Three types of directives – structural, attribute and component directive

Attribute directive changes the appearance of the DOM. For example [hidden]. It doesn’t change the structure as it hides and shows but not removes.

Component directive has its own UI and own Code like <app-tasks>

ngModel is a directive

Graphical user interface, table

Description automatically generated

\*ngIf - \* denotes it is structural directive which changes the structure of the DOM. \*ngIf=”serverCreated; else noServer” and noServer is defined as <ng-template #noServer>

\*ngFor=”let server of servers; let i = index”

@Input when we want to pass value into the child component. @Output when we want to pass outside from the child component to parent component

You can define references by # for example <input #serverNameInput> and then you can use it anywhere in the template (.html file) without #. This returns element. You can use this ref inside .ts file by using @ViewChild. This returns ElementRef.

Similar to @ViewChild you can use @ContentChild to access ng-content element from html in .ts file

Ng-content is like props.children in React.

ngOnChanges is called in the starting and also whenever properties with @Input changes

ngDoCheck is called during every change detection run, like an event is fired, some property is changed, some observable received value

ngOnChanges() (OnChanges): to detect changes for variables passed by value

ngDoCheck() (DoCheck) : to detect changes for variable passed by reference such as arrays, which are not detected by ngOnChanges() as the old value and the new value have the same reference

Lifecycle Methods – ngOnChanges, ngOnInit, ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit,ngafterviewchecked,ngdestory

Diagram

Description automatically generated

ngAfterViewInit gives access to elements in template, like elements tagged with @ViewChild. These elements are not present in ngOnInit

Angular Observables and Subjects Difference

Observables are unicast by design, which means each subscriber receives different values. They are lazy that means they start emitting value only when someone subscribes to it. It creates different instance for different subscribers.

These are also called cold observables

Subjects can help us overcome this issue. Subjects can multicast. Multicasting basically means that one Observable execution is shared among multiple subscribers.

When calling subscribe on a Subject it does not invoke a new execution that delivers data. It simply registers the given Observer in a list of Observers.

Subjects will make sure each subscription gets the exact same value as the Observable execution is shared among the subscribers.

subject.next is use to push data

rxjs offers different types of Subjects, namely: BehaviorSubject, ReplaySubject and AsyncSubject.

Promise and Observable difference

Graphical user interface, text, application

Description automatically generated

Error handling in observables is in error callback method

Graphical user interface, text

Description automatically generated

Currying

Function doesn’t take all arguments upfront. Give first argument which returns new function, then give another argument and it returns new function, the last function will then return value

Let dragon =

name =>

size =>

element=>name + ‘is a ‘ + size + ‘dragon ‘+element+’!’

console.log(dragon(‘inja’)(‘20’)(‘lightining’))

You can do currying by closures

let multiply = x=>

y=>console.log(y\*x);

let multiplyByTwo = multiply(2);

let multiplyByThree = multipleByTwo(3);

Nx – rwl extentions

Ng –buildable – means you can publish to npm registry

Npm init nx-workspace

MVVM – M – Model, V- View (HTML), VM – Angular(View Model, helps in binding view with model). MVW is Model, View, Whatever

Diagram

Description automatically generated

MVVM has two way binding between View Model and Model

Ivy Engine – in Angular 9

Typescript extends Javascript by adding types to the language. It is superset of javascript and makes it strongly typed. It also gives Object Oriented programming environment like classes. This helps in improving quality and productivity

Angular CLI helps to create new angular project with boiler-plate code

Decorator defines what kind of Angular class is it. If you decorate class with @Component then it means Angular Component. @NgModule means Angular Module

Angular Architecture

Diagram

Description automatically generated

1. Templates
2. Components
3. Modules
4. Data binding
5. Directives
6. Services
7. Dependency Injection

Routing helps to define the navigation of the application

<router-outlet> where routes html code is inserted

An application can have multiple router outlets. Unnamed router outlet is primary outlet and all other outlets should be named outlets

<router-outlet></router-outlet>

<router-outlet name="sidebar"></router-outlet>

-----------

Const Routes =[

{ path: "products", component: ProductListComponent },

{

path: "",

component: SidebarComponent,

outlet: "sidebar"

},

{

path: "products",

component: ProductListSidebarComponent,

outlet: "sidebar"

}

]

<a [routerLink]="[{ outlets: { primary: ['products'],sidebar: ['products'] } }]">

Products List

</a>

In this case both primary and auxiliary(secondary) routes are products (defined by outlets in html)

You need to specify all the outlets where you want the navigation to take place including the primary outlet.

routerLink navigate from html to that route

router.navigate navigate from component to that route

routerLinkActive will add this class when this route is active

A picture containing text

Description automatically generated

If no route matches then go to wildcard route



RouterState is a tree of activated routes

{

path: 'items/:date',

component: ItemsComponent,

resolve: { items: APIResolver }

}

Resolve in Routes is for RouteResolver. This allows to fetch data before navigating to that route. You can fetch the route params using ActivatedRouteSnapshot. Contains the information about a route associated with a component loaded in an outlet at a particular moment in time.

Example

import { Resolve } from '@angular/router';

import { ActivatedRouteSnapshot } from '@angular/router';

@Injectable()

export class APIResolver implements Resolve<any> {

constructor(private apiService: APIService) {}

resolve(route: ActivatedRouteSnapshot) {

return this.apiService.getItems(route.params.date);

}

}

ActivatedRoute contains information about route associated with a component loaded in an router outlet. You can access the route path and params with route.params and route.url where route:ActivatedRoute

Text

Description automatically generated

Lazy loading loads the component when it is needed. To lazy load create different modules in your app and for the module that you want to lazy load add loadChildren

Graphical user interface, text

Description automatically generated

Services to share common functionality across modules like logging, validation, http

Dependency injection (DI)

Text

Description automatically generated

Benefit of DI – It helps in decoupling. If you change at one place it will be reflected everywhere. Object creation and usage of object should be kept separate

If you create new Instance using new then it is tightly coupled and you have to

change everywhere if the constructor changes. It makes the code efficient and usable, increases modularity and easy to test

Difference ng-serve and ng-build

Text

Description automatically generated

Ng build –prod

Text

Description automatically generated

AuthGuard – rule to check before navigating to a route like security check and can be defined using canActivate

Text

Description automatically generated

Pipe takes in data as input and transforms it to a desired output.

{{birthday | date: `dd/MM/yyyy` }

@Pipe({

name: 'myCustomPipe',

pure: false/true <----- here (default is `true`)

})

Pure pipe – date pipe is pure pipe as not matter how many times you give input, output will be same. can be shared across many usages without affecting the output result. If the input parameters don’t change the output won’t change. This reasoning allows Angular to optimize the pipe and call transform method only when input parameters change. Hence it is more performant. Follows the concept of memoization

Async pipes are impure pipes that is output depends on state of the component at that time. cannot be shared because the internal state can be affected from outside

 The performance hit comes from the fact that Angular creates multiple instances of an impure pipe and also calls it’s transform method on every digest cycle.

Mutable is a type of variable that can be changed. Examples arrays and objects

Immutable are the objects whose state cannot be changed once the object is created. Example primitive types. For a pipe to be pure the input to the pipe cannot be mutable.

A picture containing chart

Description automatically generated

Chain pipes

{{birthday | date: `dd/MM/yyyy` | uppercase}}

Custom Pipes

Timeline

Description automatically generated with low confidence

Change Detection:

Ng-Zone is a parallel thread that runs parallel to main thread.

Whenever there is a change, ng-zone executes change detection.

In case of mouse move or scrolling you may not always want to run automatic

Change detection so you can switch off change detection by default

In constructor

ngZone.runOutsideAngular() and then to run something inside call ngZone.run

You can disable the zone entirely by adding following

platformBrowserDynamic().bootstrapModule(AppModule, {

ngZone: ‘noop’

}) and then to enable it add this in constructor

Constructor(applicationRef: ApplicationRef) {

applicationRef.tick();

}

Component Level:

CD Strategies:

1. Default - always on
2. onPush

onPush will make the change detection run in the component only if there is change in @Input that is the value is changed from outside the component or when an event is triggered in the component

But Angular checks for a===b that is it compares the value using strict comparison so if you pass the same instance of an object it will not run the CD. We should use Immutable.js for this purpose. It helps in getting a new reference on change. Immutable returns new list and doesn’t change the real list

import { List } from immutable

And then change salesList: EmployeeData[]

To salesList: List<EmployeeData>

In angular.json you can add, whenever you create new component using angular cli it will have onPush switched on

"@nrwl/angular:component": {  
 "style": "scss",  
 "changeDetection": "OnPush"  
}

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

When we get data from api onPush will not run as it is run only if @Input is changes so we will call markForCheck

Text, letter

Description automatically generated

Async Pipe –

Graphical user interface, text

Description automatically generated

It also takes care of unsubscribing from the observable

Angular core code for Async Pipe

Graphical user interface, text, application, email

Description automatically generated

Improve Performance by

1. Change Detection
2. Async Pipes
3. Encapsulation – For example if you have a list(names and salary) with add component. Every time you input something in add component onPush detects that there is an event change and runs the CD cycle and it runs on the list also and calculates salary all employees again. So encapsulate and create two components one for add and other for list so that change in one doesn’t trigger change in other as they are siblings

EmployeeListComponent

ListComponent

AddComponent

1. Memoization – If we have a function which returns same value multiple times in a list for example we are calculating fib (27) to calculate salary of multiple employees then if we use memo it calculates fib(27) but rest of the times it picks from cache

Import memo from ‘memo-decorator’ and then apply memo decorator to function.

@memo()

Function transform(n){fib(n)}

1. ngFor trackBy – It will ensure that only list element that has changed wll get rerendered
2. Code Splitting – Lazy loading routes and lazy loading components
3. Preload Strategy in routing file. OnDemandPreloadService is custom service you can define
4. @NgModule({
5. imports: [
6. RouterModule.forRoot(routes, {
7. preloadingStrategy: OnDemandPreloadService
8. })
9. ],
10. exports: [RouterModule]
11. })
12. export class AppRoutingModule {}

If a component decides that it doesn’t want to be part of CD then it can detach itself

changeDetector.detach()

Diagram

Description automatically generated

Whenever CD run it will not check ListComponent as it has detached itself

forRoot - module must be registered with the root NgModule of an application while invoking the forRoot() method.

It returns module and providers

public static forRoot(): ModuleWithProviders {

return {ngModule: ModalModule, providers: [ComponentLoaderFactory, PositioningService]};

}

forChild also does same thing but doesn’t create new Router service. "There is already a Router instance available in the app so please just register all of these routes with that instance."

[Angular demo runner (ng-run.com)](https://ng-run.com/edit/Yd7SpjEtZR2qD5Cs4qsy?open=app%2Fmodule2%2Fmodule2.module.ts)

Static method forRoot returns modules and providers(services) whereas Static method forChild just returns modules. Services are global and should be instantiated just once so forRoot will be used at that time and forChild in subsequent invokations

If the RouterModule didn’t have forRoot() then each feature module would instantiate a new Router service instance, which would break the application as there can only be one Router. By using the forRoot() method, the root application module imports RouterModule.forRoot(...) and gets a Router , and all feature modules import RouterModule.forChild(...) which does not instantiate another Router service .

Beyond the usage of control what we instantiate when importing our modules, forRoot and forChild offer us something much more useful. It allows us to inject configurations to our modules.

Greeting.module

[static](https://angular.io/api/upgrade/static) forRoot(config: UserServiceConfig): [ModuleWithProviders](https://angular.io/api/core/ModuleWithProviders)<GreetingModule> { return { ngModule: GreetingModule, providers: [ {provide: UserServiceConfig, useValue: config } ] }; }

app.module

import { GreetingModule } from './greeting/greeting.module'; @[NgModule](https://angular.io/api/core/NgModule)({ imports: [ GreetingModule.forRoot({userName: 'Miss Marple'}), ], })

forChild can also return providers

when we want to deliver a provider that is visible only to the “children” modules of our module

@HostListener('document: click', ['$event'])

@HostListener To handle click outside component. For example click outside context menu

Providers – have two things key and value.

Key any string with which components are injected and value can be useClass,useValue,useFactory, useExisting

    {

      provide: CSRF\_TOKEN,

      useFactory: getCSRFTokenFromCookie

    },

 { provide: SessionStorageService, useValue: TestHelper.getMockSessionStorageService() },

useExisting will use any existing provider and not instantiate new instance for the provider. For example

providers: [{provide: LoggerServiceService, useExisting: ExperimentalLoggerServiceService}]

We do not want to change the LoggerServiceService

View child versus Content child

App.component

<parent><parent>

Parent.component

<a><child></child><a>

Here child is view child of parent component

App.component

<parent><child></child></parent>

Parent component

<a><ng-content></ngcontent><a>

Here child is content child

viewProviders work only on viewChildren and not contentChildren

[EP 10.6 - Angular / Dependency Injection & Providers /Providers and viewProviders - YouTube](https://www.youtube.com/watch?v=XpfxmHM6E4E)

JWT – JSON Web Token – This is stored on client and has three parts – headers, data(name, iat issued at and expired at) and secret. Servers just need to know secret

Google monolith repo has 1 billion files and total size 86 terabytes

Nx Architecture

A picture containing company name

Description automatically generated

@nrwl/tao something that adds to Angular CLI

Workspace mantains the relationship

Reactive versus Template driven forms

Template-driven forms make use of the "FormsModule", while reactive forms are based on "ReactiveFormsModule".

Template-driven forms are asynchronous in nature, whereas Reactive forms are mostly synchronous.

In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code. Reactive forms are model driven forms. Let us get started by generating a component and then we'll update our form code.

Formbuilder - This FormBuilder allows us to build our form-model with less code.

Formcontrol – attached to each control (input,checkbox)

Formgroup - group of form controls

FormArray − Used to aggregate the values of form control into an array

ControlValueAccessor − Acts as an interface between Forms API to HTML DOM elements.

Without formbuilder

    createFormGroup() {

        return new FormGroup({

          personalData: new FormGroup({

            email: new FormControl(),

            mobile: new FormControl(),

            country: new FormControl(),

          }),

          requestType: new FormControl(),

          text: new FormControl(),

        })

      }

With Formbuilder

    createFormGroupWithBuilder(formBuilder: FormBuilder) {

        return formBuilder.group({

          personalData: formBuilder.group({

            email: 'defaul@email.com',

            mobile: ['', [Validators.required,

Validators.pattern("^[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$")] ],

            country: ''

          }),

          requestType: '',

          text: ''

        });

      }

FormArray

Pipe function – It takes the source observable performs functions on it and outputs the output observable

Eventbus emit and observable next –

Eventbus emits the data and subscriber subscribes to event bus but event bus is not the real source observer. Observable.next() emits the data and is also source of the data

ViewChild and ViewChildren

RxJS Operators

1. switchMap – If new result comes it \*switches\* to new result as soon as it comes in and cancels the previous one

concatMap – doesn’t cancel previous request but \*adds/Concatenates\* new one to the previous one. It will wait for first one to complete before starting with the new one

mergeMap- Just merges with previous one doesn’t cancel or wait for first one to complete

mergeAll -

1. from – creates observable from array which you can then subscribe to

from([1,2,3]).subscribe(data => console.log(data)) output 1 and 2 and 3,

but “of” will output array

1. of is same as from but in from you can give only one parameter and in of you can give multiple parameters

of([4,5,6],"hello").subscribe(data =>  
 ***console***.log(data)  
)

This will output [4,5,6] and hello

But below throws compile time error

from([4,5,6],"hello").subscribe(data =>  
 ***console***.log(data)  
)

1. fromEvent(document,'click').subscribe(()=>console.log("Clicked"))
2. every – will return true if all values are returned true but while iterating as soon as it find one value false it will return false and stop the iteration
3. .pipe, .tap, .filter, .map

Difference between directive and pipe

To bring it to the point in the most simple terms, i would say a pipe is to manipulate data, while a directive is more for DOM manipulation.

A pipe gets data as an input, transforms it and outputs this data in another way.

A directive gets a DOM element it's "attached" to and enhances it with some kind of features.

For example if you want to change date format then use the date but if you want to add color to text then use directive

Differential loading lets you serve up different bundles to different browsers and make your application even faster! It creates bundles based on browsers given in .browserslistrc. For UIF it is. This means last 1 version of chrome, same with firefox, last 2 version of Edge, Safari, ioS

last 1 Chrome version  
last 1 Firefox version  
last 2 Edge major versions  
last 2 Safari major versions  
last 2 iOS major versions  
Firefox ESR  
not IE 11 # Angular supports IE 11 only as an opt-in. To opt-in, remove the 'not' prefix on this line.

Ivy benefits

1. Smaller bundle size – my moving both template and ts code into js. Earlier it was 2 files .js(for ts) and ngfactory.js(for template). Treeshaking also helps as it would remove all the dead code
2. Faster Compilation
3. Easy to debug

Design Principles

DRY – Don’t repeat yourself

KIS – Keep it simple

S.O.L.I.D

Design Patterns which implement design principles

1. Strategy
2. Factory
3. DI
4. Observables

OAUTH2

[Demystifying OAuth 2.0 - A Tutorial & Primer :: Devansvd — Personal website](https://devansvd.com/oauth/)

Passport is authentication middleware for Node.js

Infinite scrolling

[Infinite scroll in Angular an RxJS (strongbrew.io)](https://blog.strongbrew.io/infinite-scroll-with-rxjs-and-angular2/)

<https://stackblitz.com/edit/angular-zr9jqh?file=app/infinite-scroll-list.component.ts>

Avoid memory leaks –

always unsubscribe subscriptions when not needed.

Async pipe takes care of unsubscription by default

Use takeUntil(compdestroyed) like in cases of polling

Resolution Modifiers

@Self - look at element injector for current component and directive

@Skipself – opposite of self – looks at parents etc but not self

@Optional – is if it doesn’t find provider then it doesn’t throw error but just returns null

@HostListener() function decorator allows you to handle events of the host element in the directive class.

the @HostBinding() function decorator allows you to set the properties of the host element from the directive class.

@HostBinding('style.border') border: string = '5px solid red';  
  
@HostListener('mouseleave') onMouseLeave() {  
 this.border = '5px solid blue';  
}

For Loops

For…in to iterate through keys

For…of to iterate through values

For loop if you want to break in between

Foreach if you are sure you want to iterate through entire array. It is more readable than for

Some when you may want to break in between but it doesn’t return item like for loop but Boolean to say if value is found

Every if you want to check every item follows that rule that is every item is even or not. It returns Boolean

4 types of decorators

1. Component decorators like @Component and @NgModule
2. Property decorators @Input and @Output
3. Method decorators @HostListeners
4. Parameter decorators @Optional, @Inject in constructor

ngx translate

localization

Graphical user interface, text, application, email

Description automatically generated

Array push, pop, increase length of array

Interceptor: Is a intermediate step where it takes the input and adds more to it before forwarding. For example HttpInterceptor would take http request, attach headers to it and then forward the request

Local storage (Persistent) –

It is HTML5 Web storage

* will stay even if user closes and opens the browser.
* It is accessible to all the pages of the domain.
* Limit: 5 MB. Error after that QUOTA\_EXCEEDED\_ERR. You cannot extend localstorage
* It can be different for different browsers ranging from 3-10 MB
* It would be better to use indexedDB like Redis for caching if more storage is needed. Methods:

Session storage –

It is HTML5 Web storage

* is for the session and gets cleared out on browser close.
* Local only to one URL and one browser session.
* When storage limits are reached, data stored in session storage is evicted by means of an LRU (Least Recently Used) policy.
* Size is different for different browsers

CDN - content delivery network - caches content (such as images, videos, or webpages) in proxy servers that are located closer to end users than [origin servers](https://www.cloudflare.com/learning/cdn/glossary/origin-server/).

A [cache hit](https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/) is when a client device makes a request to the cache for content, and the cache has that content saved. A cache miss occurs when the cache does not have the requested content. A CDN is made up of a network of servers (“points of presence,” or POPs) in locations all over the world.

[What is caching? | How is a website cached? | Cloudflare](https://www.cloudflare.com/learning/cdn/what-is-caching/)

Web workers is a thread that runs parallel to JS. Sometimes heavy computing functions can cause your UI to be fully blocked and unresponsive. You can move that code to web worker so that your rest of the UI is not blocked

ng generate web-worker app

worker.postMessage

worker.onmessage

Service Workers

ng add @angular/pwa. This will add the module, update package.json, update index.html

Angular's service worker is designed to optimize the end user experience of using an application over a slow or unreliable network connection, while also minimizing the risks of serving outdated content. Service workers can listen to all outgoing requests for javascript, api requests etc. It is more like a proxy. It is parallel thread to JS

[Angular Service Worker Tutorial - YouTube](https://www.youtube.com/watch?v=5YtNQJQu31Y). When we are offline the content is served from service worker cache. In config.json define dataGroups if you want to fetch data from cache

dataGroups:[{

“name”:”posts” // any random name

“urls”: [[https://swapi.dev/api/people?page=${page}](https://swapi.dev/api/people?page=$%7bpage%7d)],

“cacheConfig”: {“strategy”:”performance”} //strategy can be performance or freshness. Freshness is always try to reach api first and then try to get from cache

}]

Async await – If you make async call then make that method as async and whichever function needs to wait for response add await for that. Which means anything that is in async block after await will have to wait for that function to complete

ngOnInit(): void {  
 this.callMe();  
 ***console***.log("I am four");  
}

callMe = async () => {  
 ***console***.log("I am one");  
 let dosValue = await this.dos();  
 ***console***.log(dosValue);  
 ***console***.log("I am three");  
}

dos = () => {  
 return new ***Promise***((resolve, reject) => {  
 setTimeout(()=>{  
 resolve("I am two");  
 },3000)  
 })  
}

Ouput:

I am one

I am four

I am two //waits for 3000 ms

I am three

Await makes it sequential. Async await is much cleaner so if you have in depth promise chaining then use async await. Both of them solve the same purpose.

forkJoin - takes a number of input observables and waits for all passed observables to complete. Once they are complete, it will then emit a group of the last values from corresponding observables.

HTML 2 – forms were introduced

HTML 3 – embedded CSS

HTML 4 – external CSS

HTML5 – lot of new tags and structures like footer, header, input password email, sections, audio

SCSS

* Have inheritance
* structure
* Mixins make code reusable, define a mixin and then #include in other property

@mixin edit {border-bottom: 1px; color:”red”}

.editbtn {border-top: 1px;#include edit}

* Mathematical Operations

CSS universal selector versus body selector

\*{}

Body{}

In body selector only fonts and line-heights etc are inherited by rest of the DOM, margins paddings are not in universal selector everything is inherited by child elements

cloneDeep

ViewEncapsulation – for every component Angular applies unique attribute in DOM that is why when you set some CSS for one component it is not applied to other components

Graphical user interface, text, application

Description automatically generated

As Angular added \_ngcontent-rsm-c42 to h3, so any other h3 will not get red color. Angular creates shadow DOM behind the scenes

@Component {

Selector

.

.

Encapsulation: ViewEncapsulation:None //to override default behavior

}

ViewEncapsulation.Emulated is where the component is encapsulated with style class and all the element under that will have that style class. This is by default Emulated but you can change it to ShadowDOM and None. As all browsers don’t support so Angular has Emulated property to add unique attributes to have same behavior like ShadowDOM. Emulated and ShadowDOM are same. ShadowDOM is similar to iframe.

None removes all these unique attributes

Shadow DOM – is to encapsulate DOM tree and styling information

Cypress –

[Testing Angular with Cypress.io | Joe Eames | AngularConnect 2018 - YouTube](https://www.youtube.com/watch?v=eZyD-8qglWY)

Selenium is very hard to set. It can work with any framework. More like React. Cypress has Mocha as testing framework and Chai as assertion library. More like Angular. Cypress helps in Integration and E2E testing. Cypress has in browser and that’s why it has full control on DOM. E2E testing is done with live DB. Cypress Interceptor can intercept the request.

Cy.visit to visit a URL

Graphical user interface

Description automatically generated

Integration testing doesn’t include testing with live db but mock data and doesn’t need trip to App server

Graphical user interface, website

Description automatically generated

Advantages

1. Nice UI
2. Bundled technologies like jquery, Promise, underscore, moment.js etc
3. Simple syntax
4. XHR interceptor
5. Fixture folder has mock data jsons from which cypress picks up mockdata

it.only to only run that test

Webpack

cacheGroups under optimization is where we define rules for how Webpack should group chunks into output files.This is for third party.

For our app code

|  |
| --- |
| entry: { |
|  | main: path.resolve(\_\_dirname, 'src/index.js'), |
|  | ProductList: path.resolve(\_\_dirname, 'src/ProductList/ProductList.js'), |
|  | ProductPage: path.resolve(\_\_dirname, 'src/ProductPage/ProductPage.js'), |

Bundles created will be

Graphical user interface, application

Description automatically generated

Webpack is responsible for loading lzay loaded files dunamically

Const and readonly

const Arr = [1,2,3];

Arr[0] = 10; //OK

Arr.push(12); // OK

Arr.pop(); //Ok

//But

Arr = [4,5,6] // ERROR

But in case of ReadonlyArray you can not change the array as shown above.

arr1 : ReadonlyArray<number> = [10,11,12];

arr1.pop(); //ERROR

arr1.push(15); //ERROR

arr1[0] = 1; //ERROR

Module Loading: Services are available globally but components are loaded only in their module. It doesn’t load in the hierarchy, you need to reload it. BrowserModule includes CommonModule so you do not need to load CommonModule again in AppModule, but you need to include CommonModule again in every child module. Import [BrowserModule](https://angular.io/guide/ngmodule-faq#q-browser-vs-common-module) only in the root AppModule.

[Understanding Angular modules (NgModule) and their scopes | by Cyrille Tuzi | Medium](https://medium.com/@cyrilletuzi/understanding-angular-modules-ngmodule-and-their-scopes-81e4ed6f7407#:~:text=Angular%20itself%20is%20subdivided%20in,you%20import%20these%20other%20modules.)

[CommonModule](https://angular.io/api/common/CommonModule) from @angular/common gives access to the Angular directives such as [NgIf](https://angular.io/api/common/NgIf) and NgFor.

Declarable classes are the public classes.

Modules which has only services need to be added just once in the app as services are global example [HttpClientModule](https://angular.io/api/common/http/HttpClientModule)

Modules to import only once

* HttpClientModule
* BrowserAnimationsModule or NoopAnimationsModule
* any other module providing you services only.

Examples of some OOTB pipes – DatePipe, JSONPipe, CurrencyPipe, PercentagePipe, LowerCasePipe, UpperCasePipes

.d.ts –

**providedIn: 'root' – creates single instance at root of the application. Its better that providing in app.module as it allows tree shaking and will be used only when any component that needs it is loaded.** This method is preferred because it enables tree-shaking of the service if nothing injects it

**providedIn: 'platform' – creates singe instance for multiple applications**

**providedIn:’any’-create new instance wherever used**

data-attributes are defined in template and used to access those elements in component

ngtemplate is used to define dynamic templates. ngtemplate doesn't render anything unless called from ngoutletemplate

$implicit in ngTemplateOutletContext helps to set default value. Below we have not set value for name so it will take from $implicit

[ngTemplateOutlet: The secret to customisation - Angular inDepth](https://indepth.dev/posts/1405/ngtemplateoutlet)

<ng-template let-name let-message="message" #template3>

<p>Dear {{name}} , {{message}} </p>

</ng-template>

<ng-container [ngTemplateOutlet]="templates"

[ngTemplateOutletContext] ="{$implicit:'Guest',message:'Welcome to our site'}">

</ng-container

ViewChild("foo',static:true) means it will be available in ngOninit and false means it is dynamic and will be available only in ngafterviewinit after the change detection has run and ngif and ngfor are resolved

This applies to ViewChild and ContentChild.

ViewChildren and ContentChildren always have static false, that is they are always available in ngafterviewinit and ngaftercontent init

Forwardref means that this class is referred before it is defined

<https://stackblitz.com/edit/ng-template-outlet-example>

Renderer 2

Renderer2 is a service which provides methods like createElement, createText, appendChild and addClass that help you query and manipulate the DOM elements in your Angular application.

Angular compatibility compiler (ngcc). CLI commands run ngcc as needed when performing an Angular build.

JIT(just in time) and AOT(ahead of time) are compilers

View Engine was earlier rendering engine and Ivy is replacement rendering engine of that. Ivy uses AOT underneath