**Angular Review List**

**Analysis and Design activities**

1. Decide on UI structure, if the new feature creates a module.
2. Decide on components to be built by category of presentational or container components.
3. Decide on redux state activities to be performed
4. Decide on redux actions, reducers, effects.

**Coding rules**

1. Any additional npm package, utility or pattern to be implemented must be discussed in the team for a review.
2. UI structure adheres to accepted Angular project guidelines.
3. UI State management pattern should be implemented as per redux state management guidelines.

**Principles**

* LIFT – File and folder structure principle
  + Locate code quickly
  + Identify code at a glance
  + Flattest structure possible
  + Try to be DRY (don’t repeat yourself principle)
* SOLID
  + Single responsibility principle
  + Open/closed principle
  + Liskov substitution principle
  + Interface segregation principle
  + Dependency inversion principle
* 5 seconds rule
  + If you can’t understand code in 5 seconds, it probably needs refactoring.
  + 5 seconds is a figure of speech but it means if you can’t figure it out quickly its a candidate.
* Organization of Code for readability
  + Code
    - private variables
    - public properties
    - public methods
    - private methods
  + Grouped and sorted
  + Consistent naming and spelling matters
* One item per file – Component, Directive, Services, Pipes (Doesn’t apply to domain/model objects)

**Components**

* Before starting to code, identify smart vs. dummy components or container vs presentational components.
* The component should deal with ONLY display logic.
* Avoid logic in templates, it should talk to a model variable that is updated by logic in typescript file.
* Prefixing component selectors
  + <app name>-<component selector> for application-wide components available in core and shared modules.
  + <app name>-<feature name>-<component selector> for feature module components.
* Always use @Input() or @Output(), don’t declare them in @Component tag as it’s not recommended.
* Maintain Immutability when passing data to child components. This will ensure that with reference bugs that are hard to find in JavaScript will not happen.
* Safe Navigation Operator example \*ngIf = “**products.length**” will fail if products is null/undefined, use “**products?.length**“

**Class – Component/Service/Directives/Pipes/Interceptors etc**

* The class name and file name should be the same. eg product-list.component will have class as ProductListComponent.
* Use Angular guidelines for suffixes such as Component, Service, etc.
* The class name is PascalCase.
* Class variables/attributes/properties are camelCase.
* Class methods are camelCase.
* Observable type variable names should suffix $ in its name.
* Constant names are UPPER\_CASE with an underscore between each word.
* Class member sequence (Note variables/methods are public by default in TypeScript)
  + private variables
  + public properties
  + public methods
  + private methods
* Declare actual type used to keep application TypeSafe. DON’T use any until the real type is unknown.
* Always mark services as injectable
* Services are singleton by default
* Services, if declared as a provider in a lazily loaded feature module
  + Needs caution as you may end up having multiple instances.
  + Multiple instances may be needed at times but that should be the vision rather than an oversight.
* Consider caching your request results
* Service should contain all business and data manipulation Logic
* No long methods/functions
  + Methods should be readable
  + It should follow the single responsibility principle
  + The cyclomatic complexity of the method should be low.
  + Apply 5 seconds rule to refactor
* Follow DRY
  + If there is a code available, similar or few changed lines of code that duplicates same code needs refactoring
  + There should be only one copy of code if several implementations are needed it should be changed to apply DRY.
* Replace magic strings with constants (code reuse)
* Variable/method names should well define what it is doing or used for.
* Consider caching your request results.
* Use barrels to reduce the number of imports.
* Use Aliases for imports as we do with @shared, @core to have smaller imports

**Tests**

* Unit Tests are written to cover all public methods
* Variables/Methods should be written with a correct scope such as public/private etc. The scope should not be elevated for unit testing purposes.
* There should be negative and positive unit tests to cover both scenario’s
* There should be one unit test per logical path, we should have multiple tests to cover the overall public method.
* All dependencies of a Service or Component that is being tested should be mocked.
* Any utility type dependency in service or component under unit test should NOT be mocked.
* There can be more than one assertions in the unit test, There MUST be one assertion else it is not a test.
* 90% of code and test coverage is needed but the quality of the unit test is what matters.
* Change in code SHOULD break unit tests.
* A small set of data can be embedded in the unit test file. JSON injection is required, if we need a large set of data for unit tests.
* Setup and BeforeEach block should setup initial data for each test. An override can be done for the respective test.
* A unit tests should be small and readable – apply long method/function rules.

RxJS

* RxJS is a reactive programming library. (Reactive Extension for Java Script)
* It is usually used as a tool to accomplish the asynchronous task in Java scripts.
* It’s alternate to promises.
* In RXJS , Observable is a function used to create observer and attaches the source from where the values are expected from .
* Obervables are Lazy and can be cancelled.
* Observables are used to run asynchronously and we get the return value multiple times.

1. What is RxJS?
2. What is Stream?
3. What is Observable?
4. What is the difference between an observable and a promise?
5. What is the difference between Cold and Hot Observables?
6. What are RxJS Operators?
7. What is Observers and Subscriptions?
8. What is Subject?
9. What are different types of Subject?
10. What are different between of Subject, BehaviorSubject and ReplaySubject?
11. What is Reactive programming and how does it relate to Angular?
12. What is RxJS Map and What is Higher-Order Observable Mapping?
13. When we use the switchMap, mergeMap and concatMap?
14. What is RxJS concatMap?
15. What is RxJS mergeMap?
16. What is RxJS switchMap?
17. What is NgRx?
18. When we use zip and combineLatest and withLatestFrom?
19. What is Angular?
20. How to build full stack web application?
21. What is latest version of Angular?
22. What is difference between Angular and AngularJS?
23. What is Angular Material?
24. What is AOT (Ahead-Of-Time) Compilation?
25. What's New with Angular 8 Features? Angular 8 interview Questions
26. What's new in Angular 6?
27. What is ViewEncapsulation and how many ways are there do to do it in Angular?
28. Subjects are multicast
    * They are defined with multi datatypes
    * They broadcast notifications to one or more subscribers
    * They can support WebSockets for communication with the server.
    * They support bi-directional communication.

https://www.tektutorialshub.com/angular/subjects-in-angular/#:~:text=Subjects%20are%20Multicast,-Another%20important%20distinction&text=More%20than%20one%20subscriber%20can,separate%20instance%20of%20the%20observable.

1. In angular web application , need to send data from observer to its subscriber

<https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/>

* + Emit()
  + Next() – send data from observer to subscriber
  + Send()
  + Publish()

EventEmitter

EventEmitter is responsible for raising the event. The @output property normally is of type EventEmitter.

The child component will use the emit() method to emit an event along with the data.

//Define output property

@Output() customerChange:EventEmitter<Customer> =new EventEmitter<Customer>();

//Raise the event using the emit method.

update() {

this.customerChange.emit(this.customer);

}

1. Using Rxjs to retrieve data from HTTP
   * When the code calls the makeRequest() method of the Observable
   * When the object containing the Observable is created
   * When the Observable is subscribed to
   * When the Observable is created
2. What is correct about *ReplaySubject* and *BehaviourSubject* in RXJS?
   * Both will return the initial value or the current value on subscription.
   * ReplaySubject will not take any initial Value.
   * When initializing ,BehavourSubject is not required initial value.
   * None
3. Correct Usage of FormArray

While(formArray.length) formArray.removeAt(0);

formArray.clear();

formArray.invalidate();

formArray.removeAll();

formArray.invalid();

Methods in formArray

at()

push()

insert()

removeAt()

setControl()

setValue()

patchValue()

reset()

getRawValue()

clear()

\_syncPendingControls()

\_forEachChild()

\_updateValue()

\_anyControls()

\_allControlsDisabled()

1. To make use of the httpClient ,we need to include HttpClientModule in the service.
2. How to extract route parameters inside the component
   * Router
   * RouterLink
   * RouterLinkActive
   * ActivatedRoute
3. Which are true about ngOninit()
   * ngOninit() is a lifecycle hook that is called by Angular
   * The ngOninit is called after the constructore is executed
   * The ngOninit can be call many time in Angular life cycle
   * None.
4. Promises and Obervable
   * Promise and Observable work with multiple values over time .
   * Promise use Ractive Extensions
   * **Promise cancellable**
   * Observable is array whose items are asynchromnously over time.
5. Styling the child component from parent component .
   * Ng-deep
   * /deep/
   * Encapsulation mode
   * All the above
6. How to use SVGtemplate - In svg.component.ts

import { Component } from '@angular/core';

@Component({

selector: 'app-svg',

templateUrl: './svg.component.svg',

styleUrls: ['./svg.component.css']

})

export class SvgComponent {

fillColor = 'rgb(255, 0, 0)';

changeColor() {

const r = Math.floor(Math.random() \* 256);

const g = Math.floor(Math.random() \* 256);

const b = Math.floor(Math.random() \* 256);

this.fillColor = `rgb(${r}, ${g}, ${b})`;

}

}

1. Which are the decorators defined in root module
   * @RootModule
   * @Module
   * @NgRootModule
   * @NgModule

The **@NgModule** decorator identifies AppModule as an Angular module class (also called an NgModule class). @NgModule takes a metadata object that tells Angular how to compile and launch the application. imports — the BrowserModule that this and every application needs to run in a browser.

1. Incorrect statement about pipe .
   * Pipes are impure by default
   * There are 2 types of pipes -pure and impure
   * Angular executes a pure pipe only when it detects a pure change to the input value.
   * None
2. Correct statement about service class load the new component at the runtime .
   * Component Factory - Base class for a factory that can create a component dynamically. Instantiate a factory for a given type of component with resolveComponentFactory(). Use the resulting ComponentFactory.create() method to create a component of that type.
   * Dynamic ComponentFactory
   * DynamicFactoryLoader - Component templates are not always fixed. An application might need to load new components at runtime.
   * ComponenetFactoryResolver
3. Injecting a service in constructor without access modifiers
   * Work normally
   * Compilation error
   * Compilation success and Page not displayed
4. Subject / Behaviour Subject
   * Once you unsubscribe , BehaviourSubject can be reused and Subject cannot.
   * If you subscribe to a Subject or Behaviour Subject , you will be able to get the current value or initial value.
   * You have to define a default value whenever you declare behaviour subject or subject based on the data type
   * In subject , each next subscriber receives only the upcoming values.In behaviour subject , each next subscriber receives one previous value and upcoming values

* Subject :

In case of Subject, Observers who are subscribed at a later date will not obtain the data values emitted prior to their subscription.

* ReplaySubject :

In ReplaySubject, Observers who are subscribed at a later point will receive data values issued prior to their subscription. As it operates by using a buffer that holds the values emitted and re-emits them once new Observers are subscribed.

* BehaviorSubject :

BehaviorSubject functions similar to ReplaySubject but only re-issues the last emitted value. So you're interested in the last / current value of the observer, if BehaviorSubject is useful.

1. Services created in Angular
   * Service
   * Factory
   * Provider
   * All the Above
2. No Data
3. Import {component} from ‘angular/core’}

Import {of} from rxjs

@Component

Selector: app-root

Template : ‘<h1> Count : { (count $ | async</h1>}’

Export class AppComponent{

Count$=of(Nan)

1. Kinds of directives in Angular
   * Components Directive
   * Structural Directive
   * Attribute directive
2. Set background color of element to RED
3. Get data from parent component to child component
   * Using @input
   * Using @viewChild
   * Using service
   * All the above
   * None
4. If data model is updated outside the zone , if we need to update the view. How to do ?

If you'd like to manually fire an Angular change detection loop in a function, you can inject a reference to ChangeDetectorRef into your components, and call the detectChanges() method to fire this.

Will look something like this:

// import ChangeDetectorRef

import { ChangeDetectorRef } from '@angular/core'

// Your component's constructor

constructor(changeDetector : ChangeDetectorRef)

{ ... }

// Your setKey method

setKey(result) {

this.key = result;

this.changeDetector.detectChanges();

}

1. Use ng-for and ng-if in the same div

Use ngFor and ngIf together using ng-container

ng-container is a logical grouping element that will not be added to the DOM. that means no styles or layout applied to it.

We will refactor the above code using ng-container as shown below.

<ng-container \*ngIf="shouldShow">

<div \*ngFor="let order of orders">

<li></li>

</div>

</ng-container>

1. Life cycle of angular

<https://www.c-sharpcorner.com/article/life-cycle-of-angular-components/>

1. How to define “page not found” route.

import { NgModule } from '@angular/core';

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

import { PagenotfoundComponent } from

'./pagenotfound/pagenotfound.component';

import { PostCreateComponent } from

'./posts/post-create/post-create.component';

import { PostListComponent } from

'./posts/post-list/post-list.component';

const routes: Routes = [

{ path: '', component: PostListComponent },

{ path: 'create', component: PostCreateComponent },

{ path: 'edit/:postId',

component: PostCreateComponent },

//Wild Card Route for 404 request

{ path: '\*\*', pathMatch: 'full',

component: PagenotfoundComponent },

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

providers: []

})

export class AppRoutingModule { }

1. RxJs , display Nan
2. BOM Angular
   * BOM has W3C recommended standard specification
   * Bom can access and manipulate the browser window
   * BPM works a level above web page and includes browser attributes
   * None

|  |  |
| --- | --- |
| Document Object Model | Browser Object Model |
| 1. Represent content of Webpage 2. Manipulates HTML document 3. Objects are arranged in tree structure. 4. W3C Recommended standard specifications 5. Each browser has its own implementation | 1. In addition to webpage it includes browser attributes 2. Manipulate browser window   All global JavaScript objects, variables & functions become members of the window object implicitly |

1. <https://angular.io/guide/lifecycle-hooks>
   * @Viewchild is after AfterViewInit
2. Grouping of elements when using sructureal directives
   * Ng-group
   * Ng-container
   * Ng-template
   * Ng item
3. Change Detector and Abstract Change Detector?
   * <https://angular.io/api/core/ChangeDetectorRef>
4. Which form-control-class is set to true when value is modified.
   * Ng-touched
   * Ng-pristine
   * Ng-dirty
   * Ng-pending
5. @viewChild with life cycle hooks
6. LazyLoadedModule , ModuleForRoot ,ModuleWithProviders,Observanle<NgModule> , None
7. Differenct ways to build forms - template driven and reactive formas

Interface -driven

Model-driven

Template driven

1. No of change dedectors

<https://angular.io/api/core/ChangeDetectorRef>

1. Which directive is used to display views for given router
   * RouterOutlet
   * RouterLink
   * RouterLinkActive
   * RouterState
2. Inner animation element within an animationsequence.

<https://angular.io/api/animations/animateChild>

1. Once compoenet created , which hook is called
   * ngAfterViewInit
   * ngAfterContentInit
   * ngAfterviewChecked
   * ngOnInit
2. Pipes executed in order which we places
3. Best Efficient compiler

The Angular [ahead-of-time (AOT) compiler](https://angular.io/guide/glossary#aot) converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code. Compiling your application during the build process provides a faster rendering in the browser.