Angular Interview Questions

Prepared by Eng\ Ahmed A. Mansour

Checkout my playlists on Youtube:

https://www.youtube.com/ProgramWithAhmedMansour

You can find updates on this file by following this GitHub repository:

https://github.com/xx7Ahmed7xx/dotNetRevision

Introduction:

Angular is a **TypeScript**-based front-end web application framework. It targets both the browser and the server. Angular is one of the most popular frameworks for UI design today, with hundreds of thousands of developers worldwide contributing to its community project. Its compelling features and performance have made it an essential tool for any web developer to have in their toolbox. It is maintained by Google, and its primary function is to design **single-page applications**. Angular has significant advantages as a framework, while also providing a common architecture for developers to work with. It allows developers to create large, maintainable applications.

Angular is a robust front-end **JavaScript framework** that is widely used for app development. With the increased popularity, there is a high demand for Angular developers. Angular was introduced to create Single Page applications. This framework brings structure and consistency to web applications and provides excellent scalability and maintainability. It uses **HTML's** syntax to express your application's components clearly.

Angular is an open-source web application development framework created by **Google**. It is used to build front-end and single-page applications that run on JavaScript. It is a full-fledged framework.

1. **Why were client-side frameworks like Angular introduced?**

Back in the day, web developers used VanillaJS and jQuery to develop dynamic websites but, as the logic of one's website grew, the code became more and more tedious to maintain. For applications that use complex logic, developers had to put in extra effort to maintain the separation of concerns for the app. Also, jQuery did not provide facilities for data handling across views.

For tackling the above problems, client-side frameworks like Angular came into the picture, which made life easier for the developers by handling the separation of concerns and dividing code into smaller bits of information (In the case of Angular, called Components). Client-side frameworks allow one to develop advanced web applications like Single-Page-Application. Not that we cannot develop SPAs using VanillaJS, but by doing so, the development process becomes slower.

2. **How does an Angular application work?**

Every Angular app consists of a file named angular.json. This file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application. Inside the build section, the main property of the options object defines the entry point of the application which in this case is main.ts. The main.ts file creates a browser environment for the application to run, and, along with this, it also calls a function called bootstrapModule, which bootstraps the application.

3. **What are some of the advantages of Angular over other frameworks?**
    - Features that are provided out of the box - Angular provides a number of built-in features like routing, state management, rxjs library and http servicesstraight out of the box. This means that one does not need to look for the above-stated features separately. They are all provided with angular.
    - Declarative UI - Angular uses HTML to render the UI of an application. HTML is a declarative language and is much easier to use than JavaScript.
    - Long-term Google support - Google announced Long-term support for Angular. This means that Google plans to stick with Angular and further scale up its ecosystem.
4. **What are the advantages of Angular over React?**

Angular	React
Angular supports bidirectional data binding as well as mutable data.	React only supports unidirectional and immutable data binding.
The biggest benefit of Angular is that it enables dependency injection.	React allows us to either accomplish it ourselves or with the aid of a third-party library.
Angular can be used in both mobile and web development.	React can only be used in UI development only.
Angular features a wide wide range of tools, libraries, frameworks, plugins, and so on that make development faster and more fun.	In React we can use third-party libraries for any features.
Angular uses Typescript.	React uses Javascript.

5. **List out differences between AngularJS and Angular?**

Features	AngularJS	Angular
Architecture	AngularJS uses MVC or Model-View-Controller architecture, where the Model contains the business logic, the Controller processes information and the View shows the information present in the Model.	Angular replaces controllers with Components. Components are nothing but directives with a predefined template.

Features	AngularJS	Angular
Language	AngularJS uses JavaScript language, which is a dynamically typed language.	Angular uses TypeScript language, which is a statically typed language and is a superset of JavaScript. By using statically typed language, Angular provides better performance while developing larger applications.
Mobile Support	AngularJS does not provide mobile support.	Angular is supported by all popular mobile browsers.
Structure	While developing larger applications, the process of maintaining code becomes tedious in the case of AngularJS.	In the case of Angular, it is easier to maintain code for larger applications as it provides a better structure.
Expression Syntax	While developing an AngularJS application, a developer needs to remember the correct ng-directive for binding an event or a property.	Whereas in Angular, property binding is done using "[ ]" attribute and event binding is done using "( )" attribute.

6. How are Angular expressions different from JavaScript expressions?

The first and perhaps, the biggest difference is that Angular expressions allow us to write JavaScript in HTML which is not the case when it comes to JavaScript expressions.
Next, Angular expressions are evaluated against a local scope object whereas JavaScript expressions are against a global window object.
In Angular, undefined and null are handled implicitly, while in JavaScript it will show undefined.
The difference which makes Angular expressions quite beneficial is the use of pipes. Angular uses pipes(called filters in AngularJS), which can be used to format data before displaying. JS doesn’t have it.

7. What are Single Page Applications (SPA)?

Single page applications are web based applications that only need to be loaded once, with new functionality consisting of only minor changes to the user interface. It does not load new HTML pages to display the content of the new page, but rather generates it dynamically. This is made feasible by JavaScript's ability to alter DOM components on the current page. A Single Page Application method is speedier, resulting in a more consistent user experience.

8. What are templates in Angular?

A template is a kind of HTML that instructs Angular about how to display a component. An Angular HTML template, like conventional HTML, produces a view, or user interface, in the browser, but with far more capabilities. Angular API evaluates an HTML template of a component, creates HTML, and renders it.

There are two ways to create a template in an Angular component:

- Inline Template (defined in typescript file)

- Linked Template (html file linked inside the template attribute in the typescript file)

9. What are directives in Angular?

A directive is a class in Angular that is declared with a @Directive decorator.Every directive has its own behaviour and can be imported into various components of an application.

When to use a directive?

Consider an application, where multiple components need to have similar functionalities. The norm thing to do is by adding this functionality individually to every component but, this task is tedious to perform. In such a situation, one can create a directive having the required functionality and then, import the directive to components which require this functionality.

Types of directives:

- Component directives

These form the main class in directives. Instead of @Directive decorator we use @Component decorator to declare these directives. These directives have a view, a stylesheet and a selector property.

- Structural directives

These directives are generally used to manipulate DOM elements. Every structural directive has a ‘ * ’ sign before them. We can apply these directives to any DOM element.

- Attribute Directives

These directives are used to change the look and behaviour of a DOM element.

10. Explain Components, Modules and Services in Angular?

Components:

In Angular, components are the basic building blocks, which control a part of the UI for any application.

A component is defined using the @Component decorator. Every component consists of three parts, the template which loads the view for the component, a stylesheet which defines the look and feel for the component, and a class that contains the business logic for the component.

For creating a component, inside the command terminal, navigate to the directory of the application created, and run the following command: ‘ng generate component test’ Or ‘ng g c test’

Modules:

A module is a place where we can group components, directives, services, and pipes. Module decides whether the components, directives, etc. can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator.

By default, modules are of two types:

- Root Module

- Feature Module

Every application can have only one root module whereas, it can have one or more feature modules.
A root module imports BrowserModule, whereas a feature module imports CommonModule.
To create a feature module, run the following command: ‘ng g m test-module’

Services:

Services are objects which get instantiated only once during the lifetime of an application. The main objective of a service is to share data, functions with different components of an Angular application.

A service is defined using a @Injectable decorator. A function defined inside a service can be invoked from any component or directive.

To create a service, run the following command: ‘ng g s test-service’

11. What is a scope?

In Angular, a scope is an object that refers to the application model. It is a context in which expressions can be executed. These scopes are grouped hierarchically, comparable to the DOM structure of the application. A scope aids in the propagation of various events and the monitoring of expressions.

12. What is data binding in Angular?

Data binding is one of the most significant and effective elements for creating communication between the DOM and the component. It makes designing interactive apps easier by reducing the need to worry about data pushing and pulling between the component and the template.

There are Four types of Data binding in Angular:

- **Property Binding**: One method of data binding is called property binding. In property binding, we connect a DOM element's property to a field that is a declared property in our TypeScript component code. In reality, Angular transforms string interpolation into property binding internally.
- **Event Binding**: Using event binding, you may respond to DOM events like button clicks and mouse movements. When a DOM event (such as a click, change, or keyup) occurs, the component's designated method is called.
- **String Interpolation**: In order to export data from TypeScript code to an HTML template(view), String Interpolation is a one way data binding approach. The data from the component is shown to the view using the template expression enclosed in double curly braces. The value of a component property is added by using string interpolation.
- **Two-way data binding**: Data sharing between a component class and its template is referred to as two-way data binding. If you alter data in one area, it will immediately reflate at the other end. This happens instantly and automatically, ensuring that the HTML template and TypeScript code are always up to date. Property binding and event binding are coupled in two-way data binding.

13. **What are Decorators and their types in Angular?**

Decorators are a fundamental concept in TypeScript, and because Angular heavily relies on TypeScript, decorators have become an important element of Angular as well.
Decorators are methods or design patterns that are labeled with a prefixed @ symbol and preceded by a class, method, or property. They enable the modification of a service, directive, or filter before it is utilized. A decorator, in essence, provides configuration metadata that specifies how a component, class, or method should be processed, constructed, and used at runtime. Angular includes a number of decorators which attach various types of metadata to classes, allowing the system to understand what all these classes signify and how they should function.

Types of decorators:

- **Method Decorator**: Method decorators, as the name implies, are used to add functionality to the methods defined within our class.
- **Class Decorator**: Class Decorators are the highest-level decorators that determine the purpose of the classes. They indicate to Angular that a specific class is a component or module. And the decorator enables us to declare this effect without having to write any code within the class.
- **Parameter Decorator**: The arguments of your class constructors are decorated using parameter decorators.
- **Property Decorator**: These are the second most popular types of decorators. They enable us to enhance some of the properties in our classes.

14. **What are annotations in Angular ?**

These are language features that are hard-coded. Annotations are merely metadata that is set on a class to reflect the metadata library. When a user annotates a class, the compiler adds an annotations property to the class, saves an annotation array in it, and then attempts to instantiate an object with the same name as the annotation, providing the metadata into the constructor. Annotations in AngularJs are not predefined, therefore we can name them ourselves.

15. **What are pure and impure Pipes?**

Pure are pipelines that only employ pure functions. As a result, a pure pipe does not employ any internal state, and the output remains constant as long as the parameters provided remain constant. Angular calls the pipe only when the parameters being provided change. A single instance of the pure pipe is utilized in all components.
Angular calls an impure pipe for each change detection cycle, independent of the change in the input fields. For each of these pipes, several pipe instances are produced. These pipes' inputs can be altered.
By default, all pipelines are pure.

16. **What is Pipe transform Interface in Angular?**

An interface used by pipes to accomplish a transformation. Angular calls the transform function with the value of a binding as the first argument and any arguments as the second parameter in list form. This interface is used to implement custom pipes.

17. **Write a code where you have to share data from the Parent to Child Component?**

You have to share the data amongst the components in numerous situations. It may consist of unrelated, parent-child, or child-parent components.
The @Input decorator allows any data to be sent from parent to child.

```
// parent component

import { Component } from '@angular/core';
@Component({
  selector: 'app-parent',
  template: `
<app-child [childMessage]="parentMessage"></app-child>
`,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  parentMessage = "message from parent"
  constructor() { }
}
// child component
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-child',
  template: `Say {{ childMessage }}`,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() childMessage: string;
  constructor() { }
}
```

18. **What do you understand by scope in Angular?**

The scope in Angular binds the HTML, i.e., the view, and the JavaScript, i.e., the controller. It as expected is an object with the available methods and properties. The scope is available for both the view and the controller. When you make a controller in Angular, you pass the $scope object as an argument.

19. How to use ‘ngFor’ in a tag?
    - The ngFor directive is used to build lists and tables in the HTML templates. In simple terms, this directive is used to iterate over an array or an object and create a template for each element.
    - `<ul>`
    - `    <li *ngFor = "let items in itemlist"> {{ item }} </li>`
    - `  </ul>`
    - “Let item” creates a local variable that will be available in the template
    - “Of items” indicates that we are iterating over the items iterable.
    - The * before ngFor creates a parent template.

20. **What is HttpClient, and what are its benefits?**

HttpClient is an Angular module used for communicating with a back-end service via the HTTP protocol. Usually, in front-end applications, for sending requests, we use the fetch API. However, the fetch API uses promises. Promises are useful, but they do not offer the rich functionalities that observables offer. This is why we use HttpClient in Angular as it returns the data as an observable, which we can subscribe to, unsubscribe to, and perform several operations on using operators. Observables can be converted to promises, and an observable can be created from a promise as well.

21. **What is REST?**

REST in Angular stands for Representational State Transfer. It is an API that works on the request of HTTP. Here, the requested URL points to the data that has to be processed, after which an HTTP function is used to identify the respective operation that has to be performed on the data given. The APIs that follow this method are referred to as RESTful APIs.

Experienced Level:

22. **What are Template and Reactive forms?**

Template-driven approach	Reactive Form Approach
○ In this method, the conventional form tag is used to create forms. Angular automatically interprets and creates a form object representation for the tag. ○ Controls can be added to the form using the NGModel tag. Multiple controls can be grouped using the NGControlGroup module. ○ A form value can be generated using the "form.value" object. Form data is exported as JSON values when the submit method is called. ○ Basic HTML validations can be used to validate the form fields. In the case of custom validations, directives can be used. ○ Arguably, this method is the simplest way to create an Angular App.	○ This approach is the programming paradigm oriented around data flows and propagation of change. ○ With Reactive forms, the component directly manages the data flows between the form controls and the data models. ○ Reactive forms are code-driven, unlike the template-driven approach. ○ Reactive forms break from the traditional declarative approach. ○ Reactive forms eliminate the anti-pattern of updating the data model via two-way data binding. ○ Typically, Reactive form control creation is synchronous and can be unit tested with synchronous programming techniques.

23. **Angular by default, uses client-side rendering for its applications. Can we activate server-side rendering?**

Yes, angular provides a technology called Angular Universal, which can be used to render applications on the server-side. The advantages of using Angular Universal are:

First time users can instantly see a view of the application. This benefits in providing better user experience. Many search engines expect pages in plain HTML, thus, Universal can make sure that your content is available on every search engine, which leads to better SEO.
Any server-side rendered application loads faster since rendered pages are available to the browser sooner.

24. **What is Eager and Lazy loading?**
    - Loading: The eager loading technique is the default module-loading strategy. Eager loading feature modules are loaded before the program begins. This is primarily utilized for small-scale applications.
    - Lazy Loading: Lazy loading loads the feature modules dynamically as needed. This speeds up the application. It is utilized for larger projects where all of the modules are not required at the start.

25. **What is view encapsulation in Angular?**

View encapsulation specifies if the component's template and styles can impact the entire program or vice versa.
Angular offers three encapsulation methods:

- Native: The component does not inherit styles from the main HTML. Styles defined in this component's @Component decorator are only applicable to this component.
- Emulated (Default): The component inherits styles from the main HTML. Styles set in the @Component decorator are only applicable to this component.
- None: The component's styles are propagated back to the main HTML and therefore accessible to all components on the page. Be wary of programs that have None and Native components. Styles will be repeated in all components with Native encapsulation if they have No encapsulation.

26. **What are RxJs in Angular ?**

xJS is an acronym that stands for Reactive Extensions for JavaScript. It is used to enable the use of observables in our JavaScript project, allowing us to do reactive programming. RxJS is utilized in many popular frameworks, including Angular since it allows us to compose our asynchronous or callback-based code into a sequence of operations executed on a data stream that releases values from a publisher to a subscriber. Other programming languages, such as Java and Python, offer packages that allow them to develop reactive programs utilizing observables.
Most of the time, rxJs is used in HTTP calls with angular. Because http streams are asynchronous data, we can subscribe to them and apply filters to them.

27. How are observables different from promises?

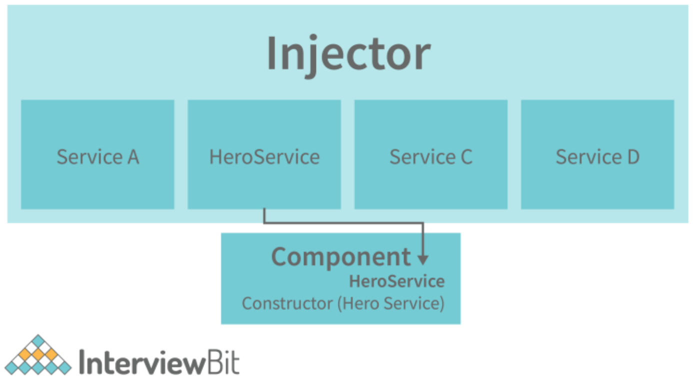
The first difference is that an Observable is lazy whereas a Promise is eager.

Promise	Observable
Emits a single value	Emits multiple values over a period of time
Not Lazy	Lazy. An observable is not called until we subscribe to the observable
Cannot be cancelled	Can be cancelled by using the unsubscribe() method
	Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc.

28. Explain the concept of Dependency Injection?

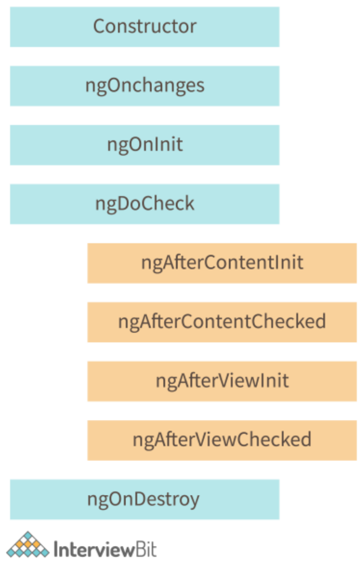
Dependency injection is an application design pattern which is implemented by Angular. It also forms one of the core concepts of Angular.

So what is dependency injection in simple terms?
Let’s break it down, dependencies in angular are nothing but services which have functionality. The functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies into our components and directives. So basically, we are just making dependencies which are injectable across all components of an application.



29. What are lifecycle hooks in Angular? Explain a few lifecycle hooks.

Every component in Angular has a lifecycle, and different phases it goes through from the time of creation to the time it's destroyed. Angular provides hooks to tap into these phases and trigger changes at specific phases in a lifecycle.



- ngOnChanges( ) This hook/method is called before ngOnInit and whenever one or more input properties of the component change.
- This method/hook receives a SimpleChanges object which contains the previous and current values of the property.
- ngOnInit( ) This hook gets called once, after the ngOnChanges hook.
- It initializes the component and sets the input properties of the component.
- ngDoCheck( ) It gets called after ngOnChanges and ngOnInit and is used to detect and act on changes that cannot be detected by Angular.
- We can implement our change detection algorithm in this hook. ngAfterContentInit( ) It gets called after the first ngDoCheck hook. This hook responds after the content gets projected inside the component.
- ngAfterContentChecked( ) It gets called after ngAfterContentInit and every subsequent ngDoCheck. It responds after the projected content is checked.
- ngAfterViewInit( ) It responds after a component's view, or a child component's view is initialized.
- ngAfterViewChecked( ) It gets called after ngAfterViewInit, and it responds after the component's view, or the child component's view is checked.
- ngOnDestroy( ) It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.

30. What are router links? And What exactly is the router state?

RouterLink is an anchor tag directive that gives the router authority over those elements. Because the navigation routes are set.

RouterState is a route tree. This tree's nodes are aware of the "consumed" URL segments, retrieved arguments, and processed data. You may use the Router service and the routerState property to get the current RouterState from anywhere in the application.

31. What does Angular Material means?

Angular Material is a user interface component package that enables professionals to create a uniform, appealing, and fully functioning websites, web pages, and web apps. It does this by adhering to contemporary web design concepts such as gentle degradation and browser probability.

32. What is transpiling in Angular ?

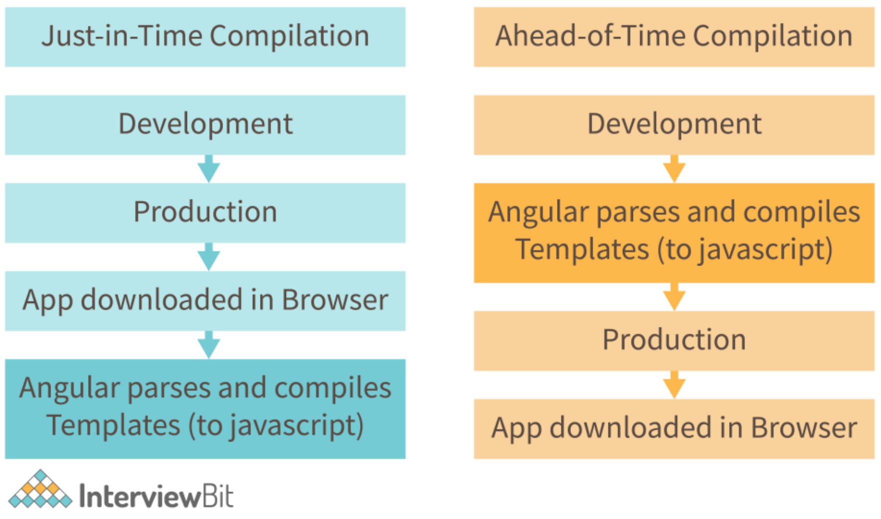
Transpiling is the process of transforming the source code of one programming language into the source code of another. Typically, in Angular, this implies translating TypeScript to JavaScript. TypeScript (or another language like as Dart) can be used to develop code for your Angular application, which is subsequently transpiled to JavaScript. This occurs naturally and internally.

33. What are HTTP interceptors?

Using the HttpClient, interceptors allow us to intercept incoming and outgoing HTTP requests. They are capable of handling both HttpRequest and HttpResponse. We can edit or update the value of the request by intercepting the HTTP request, and we can perform some specified actions on a specific status code or message by intercepting the answer.

34. What is AOT compilation? What are the advantages of AOT?

Every Angular application consists of components and templates that the browser cannot understand. Therefore, all the Angular applications need to be compiled first before running inside the browser. Angular provides two types of compilation:


Just-in-Time Compilation
Development
Production
App downloaded in Browser
Angular parses and compiles Templates (to javascript)
Ahead-of-Time Compilation
Development
Angular parses and compiles Templates (to javascript)
Production
App downloaded in Browser
InterviewBit

- JIT(Just-in-Time) compilation
- AOT(Ahead-of-Time) compilation

In JIT compilation, the application compiles inside the browser during runtime. Whereas in the AOT compilation, the application compiles during the build time.

The advantages of using AOT compilation are:

- Since the application compiles before running inside the browser, the browser loads the executable code and renders the application immediately, which leads to faster rendering.
- In AOT compilation, the compiler sends the external HTML and CSS files along with the application, eliminating separate AJAX requests for those source files, which leads to fewer ajax requests.
- Developers can detect and handle errors during the building phase, which helps in minimizing errors.
- The AOT compiler adds HTML and templates into the JS files before they run inside the browser. Due to this, there are no extra HTML files to be read, which provide better security to the application.

By default, angular builds and serves the application using JIT compiler:
ng build
ng serve

For using AOT compiler following changes should be made:
ng build –aot
ng serve --aot

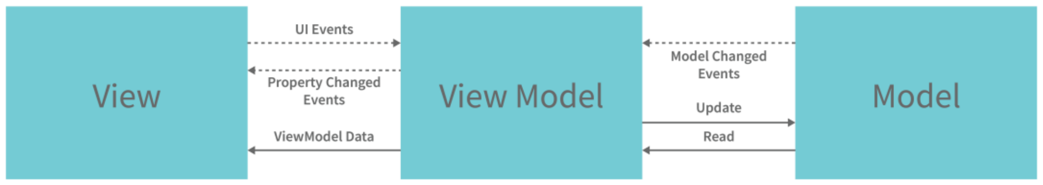
35. What is Change Detection, and how does the Change Detection Mechanism work?

The process of synchronizing a model with a view is known as Change Detection. Even when utilizing the ng Model to implement two-way binding, which is syntactic sugar on top of a unidirectional flow. Change detection is incredibly fast, but as an app's complexity and the number of components increase, change detection will have to do more and more work.
Change Detection Mechanism-moves only ahead and never backward, beginning with the root component and ending with the last component. This is what one-way data flow entails. The tree of components is the architecture of an Angular application. Each component is a child, but the child is not a parent. A $digest loop is no longer required with the one-way flow.

36. Explain MVVM architecture?

MVVM architecture consists of three parts:

- Model
- View
- ViewModel



InterviewBit

- Model contains the structure of an entity. In simple terms it contains data of an object.
- View is the visual layer of the application. It displays the data contained inside the Model. In angular terms, this will be the HTML template of a component.
- ViewModel is an abstract layer of the application. A viewmodel handles the logic of the application. It manages the data of a model and displays it in the view.

View and ViewModel are connected with data-binding (two-way data-binding in this case). Any change in the view, the viewmodel takes a note and changes the appropriate data inside the model.

37. How does one share data between components in Angular?

Parent to child using @Input decorator
Consider the following parent component:

```
@Component({
  selector: 'app-parent',
  template: `
    <app-child [data]=data></app-child>
  ` ,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  data:string = "Message from parent";
  constructor() { }
}
```

In the above parent component, we are passing “data” property to the following child component:

```
import { Component, Input} from '@angular/core';
  @Component({
    selector: 'app-child',
    template:`
      <p>{{data}}</p>
    `,
    styleUrls: ['./child.component.css']
  })
  export class ChildComponent {
    @Input() data:string
    constructor() { }
  }
```

In the child component, we are using @Input decorator to capture data coming from a parent component and using it inside the child component’s template.

Child to parent using @ViewChild decorator

Child component:

```
import {Component} from '@angular/core';
    @Component({
      selector: 'app-child',
      template:`
        <p>{{data}}</p>
      `,
      styleUrls: ['./child.component.css']
    })
    export class ChildComponent {
      data:string = "Message from child to parent";
      constructor() { }
    }
```

Parent Component

```
import { Component,ViewChild, AfterViewInit} from '@angular/core';
    import { ChildComponent } from './../child/child.component';
    @Component({
      selector: 'app-parent',
      template: `
        <p>{{dataFromChild}}</p>
      `,
      styleUrls: ['./parent.component.css']
    })
    export class ParentComponent implements AfterViewInit {
      dataFromChild: string;
      @ViewChild(ChildComponent,{static:false}) child;
      ngAfterViewInit(){
        this.dataFromChild = this.child.data;
      }
      constructor() { }
    }
```

In the above example, a property named “data” is passed from the child component to the parent component.
@ViewChild decorator is used to reference the child component as “child” property.
Using the ngAfterViewInit hook, we assign the child’s data property to the messageFromChild property and use it in the parent component’s template.

Child to parent using @Output and EventEmitter

In this method, we bind a DOM element inside the child component, to an event (click event for example) and using this event we emit data that will captured by the parent component.

Child Component:

```
import {Component, Output, EventEmitter} from '@angular/core';
   @Component({
     selector: 'app-child',
     template:`
       <button (click)="emitData()">Click to emit data</button>
     `,
     styleUrls: ['./child.component.css']
   })
   export class ChildComponent {
     data:string = "Message from child to parent";
     @Output() dataEvent = new EventEmitter<string>();
     constructor() { }
     emitData(){
       this.dataEvent.emit(this.data);
     }
   }
```

As you can see in the child component, we have used @Output property to bind an EventEmitter. This event emitter emits data when the button in the template is clicked.
In the parent component's template we can capture the emitted data like this:

```
 <app-child (dataEvent)="receiveData($event)"></app-child>
```

Then inside the receiveData function we can handle the emitted data:

```
receiveData($event){
     this.dataFromChild = $event;
   }
```

38. How do you deal with errors in observables?

When dealing with errors in observables in Angular, catchError operator can be used to handle and recover from errors. This operator allows you to provide a fallback value or execute alternative logic when an error occurs. You can chain the catchError operator after the observable that might produce an error and define a callback function to handle the error. Within the callback function, you can perform error handling tasks such as logging the error, displaying an error message to the user, or initiating a retry mechanism.

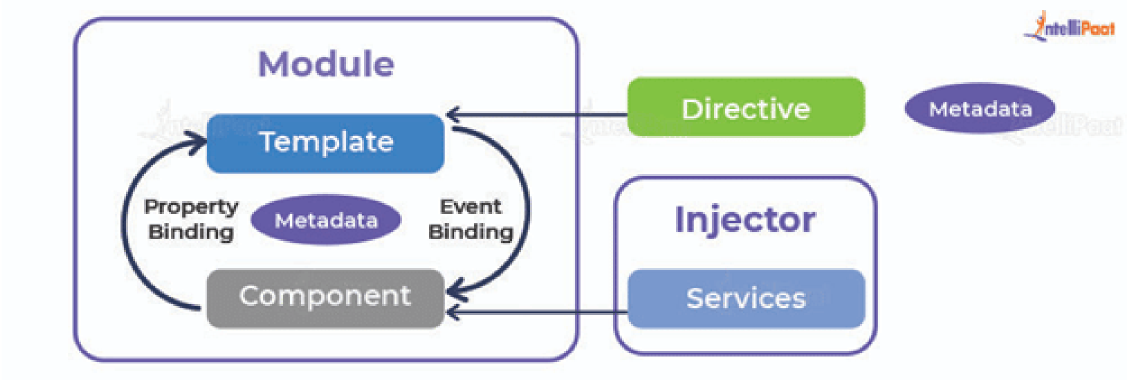
39. What happens when we use the script tag within a template?

Using the script tag within an Angular template is not a recommended practice. Angular templates are intended for defining the structure and layout of the user interface, and including scripts directly within the template goes against the separation of concerns principle. When a script tag is used within a template, the browser treats it as part of the HTML content and attempts to execute it. However, Angular's template compiler does not process or execute scripts within templates. Instead, scripts should be placed in separate JavaScript files and included using the appropriate Angular mechanisms, such as component logic or Angular modules.

40. Write a pictorial diagram of Angular architecture?

The architecture of Angular comprises the following elements. The pictorial representation of the same is given below:

- Components and Templates
- Ng Modules
- Metadata
- Data Binding
- Directives
- Services
- Dependency Injection



41. You need to create an UI that will adapt based on the role of the user. What will be your approach?

To achieve the role-based UI, we need to implement Angular Guards and directives for dynamic content display. This ensures a proper UI experience, which also makes sure of security and can be reused.

```typescript
@Injectable({ providedIn: 'root' })
export class AuthService {
  // Mock role check
  hasRole(role: string): boolean {
    return localStorage.getItem('userRole') === role;
  }
}

@Directive({
  selector: '[appRoleBasedAccess]'
})
export class RoleBasedAccessDirective {
  constructor(private el: ElementRef, private authService: AuthService) {}

  @Input() set appRoleBasedAccess(role: string) {
    if (!this.authService.hasRole(role)) {
      this.el.nativeElement.style.display = 'none';
    }
  }
}
```

42. **A form with numerous inputs is experiencing sluggish responses, especially during data entry. How can you enhance the performance of the form so that the functionality or the user experience does not get affected?**

To enhance the form and improve its performance we can implement "ChangeDetectionStrategy.OnPush" which is a strategy that the default change detector uses to detect changes. It reduces the load of processing inputs. This improves the form responsiveness by changing the rendering cycle to whenever a change is done. It minimizes the DOM updates.

```
@Component({
  selector: 'app-user-form',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class UserFormComponent {
  userForm = new FormGroup({
    name: new FormControl('')
  });

  constructor() {
    this.userForm.get('name').valueChanges.pipe(
      debounceTime(300)
    ).subscribe(value => {
      // Handle the changes made
    });
  }
}
```

43. **The Project involves fetching data from multiple APIs on a single page. Create a strategy to fetch and join this data in an organized manner. Also, ensure that there is minimal impact on user experience and performance of the code.**

We can use "forkJoin" or "combineLatest" operators from the RxJS. These operators helps when we have multiple detectables that rely on each other for some calculations or determination.

```
constructor(private http: HttpClient) {}
fetchData() {
  forkJoin({
    data1: this.http.get('/api/data1'),
    data2: this.http.get('/api/data2')
  }).subscribe(({ data1, data2 }) => console.log(data1, data2));
}
```

44. Create a custom pipe to convert strings to title case.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'titleCase'
})
export class TitleCasePipe implements PipeTransform {
  transform(value: string): string {
    if (!value) return value;
    return value.replace(/\w\S*/g, (txt) => {
      return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
    });
  }
}
```

45. Build a reactive form with validation.

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
@Component({
  selector: 'app-user-form',
  templateUrl: './user-form.component.html',
  styleUrls: ['./user-form.component.css']
})
export class UserFormComponent implements OnInit {
  userForm: FormGroup;
  ngOnInit() {
    this.userForm = new FormGroup({
      'name': new FormControl('', Validators.required),
      'email': new FormControl('', [Validators.required, Validators.email]),
      'password': new FormControl('', [Validators.required, Validators.minLength(6)])
    });
  }
  onSubmit() {
    console.log(this.userForm.value);
  }
}
```

46. Implement a service with HttpClient.

```
import  { Injectable } from '@angular/core';
import  { HttpClient } from '@angular/common/http';
@Injectable({
 providedIn: 'root'
})
export  class  UserDataService {
 constructor(private  http: HttpClient) {}
 fetchUserData() {
   this.http.get('https://jsonplaceholder.typicode.com/users')
     .subscribe(data => console.log(data));
 }
}
```

47. Use ngFor directive to display a list.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-item-list',
  template: `
    <ul>
      <li *ngFor="let item of items">{{ item }}</li>
    </ul>
  `
})
export class ItemListComponent {
  items = ['Item 1', 'Item 2', 'Item 3'];
}
```

48. Create a directive to change the background color of an element.

```
import { Directive, ElementRef, Renderer2, OnInit } from '@angular/core';
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective implements OnInit {
  constructor(private el: ElementRef, private renderer: Renderer2) {}
  ngOnInit() {
    this.renderer.setStyle(this.el.nativeElement, 'backgroundColor', 'yellow');
  }
}
```

References:

https://www.interviewbit.com/angular-interview-questions/#why-were-client-side-frameworks-like-angular-introduced

https://www.simplilearn.com/tutorials/angular-tutorial/angular-interview-questions

https://intellipaat.com/blog/interview-question/angular-interview-questions/