**What is Binding and different types of Bindings in Angular 4?**

Define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data.

1. **Interpolation**

Interpolation allows you to define properties in a component class, and communicate these properties to and from the template.

Name: {{ user.name }}

Email: {{user.email }}

1. **Property Binding**

Property binding is the same thing as interpolation, except you can set the properties and attributes of various HTML elements.

<input type=”text” [value]=”user.name”>

1. **Event Binding**

Event binding allows you to define events that occurs in the template, and communicate to the component class.

<input type=”button” (click) =”cook ()”>

1. **Two-Way Data binding**

Two way data binding allow to have the data flow both way.

<input type=”email” [ngModel]=”user.email”>

**What is difference between Component and Directives in Angular 4?**

The Component in Angular 4 defined as the basic classes which interact with the web page components such as Html file. A Components can be defined using @Component which is called as Decorator. The selector, style and template can be defined inside the component.

The Directive in Angular 4 can be defined as the extended HTML attributes which can be defined as custom attributes. The Directive in Angular 4 can be predefined or can be custom defined to manipulate the functionality of DOM element.

**Components break up the application into smaller parts; whereas, Directive add behavior to an existing DOM element.**

**What are the Directive in Angular 4 and types?**

1. **Structural Directive** – The Structural Directive defined as which changes the DOM structure by adding or removing the DOM elements.
2. **Attribute Directive** – The Attributes Directives are defined as a modification in the behavior of a component, or an element or any other directive.

**What are Services in Angular 4?**

The process of presenting the data from Model to View or vice versa is established by using Services. Services are best of applying the communication across different classes which do not know each other.

**What is AOT?**

The Angular Ahead-of-Time compiler converts your Angular HTML and Typescript code in 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.

**JIT-which compiles your app in the browser at runtime.**

**AOT-which compile your app at build time.**

**Observables**

Observable provide the support for passing the messages between publisher and subscribers in your application. Observables are declarative that is, you define the function for publishing values, but it is not executed until the consumer subscribes to it.

1. **Observables are lazy, which means nothing happens until a subscription is made. Whereas promise are eager, which means as soon as a promise created, the execution takes place.**
2. **Observable is a stream in which passing of zero or more events is possible and the callback is called for each event. Whereas, promise handles a single events.**

**Observer**

An Observer is a consumer of values delivered by an Observable. Observers are simply a set of callbacks, one for each type of notification delivered by the Observable.

1. Next - send any values such as Number, Arrays or Objects to its subscribers.
2. Error – send a JavaScript error or exception.
3. Complete – does not send any value.

**Subscription**

A subscription is an object that represents a disposable resource, usually the execution of an Observable.

**Subject**

An Rxjs Subject is a special type of Observable that allows values to be multicasted to many Observers. Subject are multicast.

**There are 3 types:**

1. **Behavior Subject** – the last event that occurred before the subscription.
2. **Replay Subject** – the observer receives all past events.
3. **Async Subject** – wait for complete to emit the last event and the complete event.

**Multicasted Observables**

A Multicasted Observable passes notifications through a Subject which may have many Subscribers, whereas a plain unicast observable only sends notifications to a Single Observer.

**Asynch Pipe**

When Observable and Promise returns something, we use a temporary property to hold the content. Later we bind the same content to the template. With the usage of AsyncPipe, the promise or observable can be directly used in a template and a temporary property is not required.

The Asynch Pipe subscribes to an Observable or Promise and return the latest value it has emitted. When the new value is emitted, the asynch Pipe marks the component to be checked for changes. When the component gets destroyed, the asynch pipe unsubscribes automatically to avoid potential memory leaks.

**Pipes**

Pipes are used to transform data, when we only need that data transformed in a template.

Built in Pipes

1. Currency Pipe – {{ 1234.56 | currency:’CAD’ }}
2. Date Pipe
3. Decimal Pipe
4. Json Pipe
5. Lowercase & Uppercase Pipe
6. Percent Pipe
7. Slice Pipe – returns a slice of an array.
8. AsynchPipe

**Transform Function**

The actual logic for the pipe is put in a function called transform.

**Creating a Pipe is very simple. We just decorate a class with the @Pipe decorator, provide a name and a transform function and we are done.**

**ng build**

Compiler an angular app into an output directory named dist at the given output path.

**ng profile**

**ng lint**

Runs linting tools on Angular app code in a given project folder.

**ng serve**

Builds and serves your app, rebuilding on file changes.

**ElementRef**

The ElementRef gives the directive direct access to the DOM element upon which it’s attached.

**ElementRef and Renderer classes are needed to get the reference of the host element and of the renderer.**

**HostListener**

This is a function decorator that accepts an event name as an argument. When that event gets fired on the host element it calls the associated function.

**HostBinding**

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

**Host**

A constructor parameter decorator that tells the DI framework to retrieve a dependency from any injector until reaching the host element of the current component.

ng generate module app-routing –flat – module=app

--flat puts the file in src/app instead of its own folder.

--module=app tells the CLI to register it in the imports array of the AppModule.

**REST**

Representational State Transfer is an architectural style for designing distributed systems. Such as being stateless, having client/server relationship and a Uniform Interface.

**Principle of REST**

1. **Resources**

Exposes easily understood directory structure URIs.

1. **Representational**

Transfer JSON or XML to represent data objects and attributes.

1. **Messages**

Use HTTP methods explicitly. (Get, Post, Put, Delete)

1. **Stateless**

Interactions store no client context on the server between requests.

**HTTP Methods**

1. **Get** – retrieve information.
2. **Post** – create a new entity, but it can also be used to update an entity.
3. **Put** – create a new entity or update an existing one.
4. **Patch** – update only the specified fields of an entity at a URI.
5. **Delete** – request that a resourced be removed.

**HTTP Status Codes**

1. **1XX**- Informational
2. **2XX**- Success
3. **3XX**- Redirection
4. **4XX**- Client error
5. **5XX**- Server error

**What new in Angular 5?**

1. Typescript version 2.4
2. Rxjs version 5.5
3. ActivationStart and ActivationEnd are in introduced in Router.

**What is Transpiling in Angular?**

The process of converting the typescript into javascript.

**Modules**

Angular app are modular and Angular has its own modularity system called NgModules. They can contains components, service providers and other code files whose scope is defined by the containing NgModule.

Every Angular app has at least one NgModule class, the root module, which is conventionally named AppModule and reside in a file named app.module.ts. An NgModule is defined by a class decorated with @NgModule(). Whose contains below properties.

1. declarations: the components, directives and pips.
2. exports: should be visible and usable in the other NgModule.
3. imports: imported other classes are needed by component templates.
4. providers: creator of services. Contributes to the global collection of services. They become accessible in all parts of the app.
5. bootstrap: set bootstrap property.

**View Encapsulation**

If we change a css class the effect is seen throughout an application, something special is happing here and it’s called View Encapsulation.

**View Encapsulation decides whether the styles defined in a component can affect the entire application or not.**

**View Encapsulation Property:**

1. **ViewEncapsulation**.**Native** – which uses the shadow DOM to scope styles only to that specific component.
2. **ViewEncapsulation**.**Emulated** – styles from other HTML do not spread to the component.
3. **ViewEncapsulation**.**None –** styles defined in a component are visible to all component.

**Life-Cycle hooks**

1. **onChanges**

Which fires when it detects changes to data bound input property. This method contains **SimpleChanges** object, which contain the current and previous property values.

**Angular uses dirty checking (===) for detecting changes in the input properties.** Whenever the parent changes the value of the input property, the angular raises the OnChanges hook event in the child component.

1. **onInit**

Initialize the Directive or Component after Angular first displays the data-bound properties and sets the Directive or Component input properties. This event is mainly used for the initialize the data in a component.

1. **doCheck**

This event is triggered every time the input properties of a component are checked.

1. **afterContentInit** – called after content projection initialization.
2. **afterContentChecked** – called after content projection is checked.
3. **afterViewInit** – called after a components view or child view initialized.
4. **afterViewChecked** – called after a components view or child view is checked.
5. **onDestroy** – called when the components is destroyed.

**Content** **Projection**

Content Projection allow you to insert a shadow DOM in your component. If you want to have custom text or content between the angular tags. <ng-content></ng-content> used for Content Projection.

**Dependency Injection**

When a component is dependent on another component the Dependency is injected/provided during runtime.

**Router Outlet**

It represents or renders the components on a template at a particular location.

**Package.json**

Mentioned typescript package and typescript version.

**Redux**

It is a library which helps us maintain the state of the application.

**Ng-class loading of css class, whereas in ng-style in set the css style.**

**Why typescript?**

It is a superset of JavaScript.

1. JavaScript not pure OOP language
2. JavaScript code difficult in maintainability and reusability.

**What new in Angular 4?**

1. Reduce the size of the **AOT (Ahead-Of-Time)** compiler generated code.
2. ‘**animations’** have been pulled out of ‘@angular/core’.
3. New ‘**titlecase** Pipe’ changes the first letter of each word into uppercase.
4. **‘ngComponentOutlet’** is possible to build dynamic components in a declarative way.

**Interceptors**

Interceptors are used to intercept mutable outgoing requests or incoming requests. It can be really useful for feature like caching and logging.

1. Setting the Origin for each outgoing request.
2. Adding authentication token to every requests.

**What is Node.Js?**

It is basically a runtime environment for executing JavaScript code outside the browser.

**Metadata**

Metadata tells Angular how to process a class.

**What is Module?**

Module allows to put logical boundaries in our application. That groups related Components, Directives, Pipes and Services together.

**Lazy Loading**

In lazy loading, our application does not everything at once. It loads only those things what the user expects to see when the app first load. It helps us decrease the startup time.

**Event Emitter**

Its only purpose is to emit events in components.

**Constructor vs. OnInit**

The class is initiated and ensures proper initialization of fields in the class and its subclass.

When we have task’s related to angular bindings we should use ngOnInit.

**Codelyzer**

Codelyzer is the open source tool that check if the coding conventions and guidelines are followed or not in our application.

**Security threats in Angular**

1. Cross site Scripting (XSS)
2. Avoid direct object reference.
3. Avoid external URLs, if not trusted.
4. Avoid direct use if DOM API.
5. Cross-Site request forgery (CSRF or XSRF)

**Create a Cutom Pipe**

1. Decorate the class with @Pipe decorator.
2. Implements PipeTranform interface in class.
3. Override the transform() method.
4. Configure the class with @NgModule.

**Forms**

Applications use forms to enable users to log in, to update a profile, to enter sensitive information and to perform many other data-entry tasks.

There are two types of forms to handling user inputs.

1. **Reactive Forms (Model Driven Forms)**

More scalable, reusable and testable. Data model structured. Predictability synchronous. Immutable.

A reactive forms is just an HTML form that’s been wired up with Rxjs to manage its state as a real-time stream.

There are three classes.

1. **FormControl** – an individual form input, checkbox, select, textarea.
2. **FormGroup** – a group of form fields that can be manipulated and validated together.
3. **FormBuilder** – a service that helps you build formsgroups easily.

Import ‘**ReactiveFormsModule’** from the **‘@angular/forms**’ package and add it to your NgModule’s imports array.

Import { ReactiveFormsModule } from ‘@angular/forms’

Import the ‘**FormControl’** class into your component and create a new instance.

Import { FormControl } from ‘@angular/forms’

Use the Constructor of FormControl to set its initial value.

name = new FormControl(‘’);

Using the FormControl binding control in the template.

<input type=”text” [formControl] = “name”>

**setValue()**

A FormControl instance provides a setValue() method that updates the value of the form control.

name.setValue(‘xxx’);

**Grouping the FormControls using FormGroup**

A FromGroup instance tracks the form state of a group of form control.

Import FormGroup from ‘@angular/form’

Create a property in the component class named ‘profileForm’ and set the property to a new form group instance.

profileForm = new FormGroup({

firstName:new FormControl(‘’),

lastName:new FormControl(‘’)

});

Binding FormGroup to template using form element.

<form [formGroup] = ’profileForm’>

<input type=’text’ formControlName=’firstName’>

</form>

FormGroup contains a group of controls, the ‘profileForm’ FormGroup is bound to the form element with the FormGroup directive, creating communication layer between model and the form containing the inputs.

**Nested form group**

Braks large forms groups into smaller, more manageable once.

profileForm = new FormGroup({

firstName:new FormControl(‘’),

lastName:new FormControl(‘’),

address:new FormGroup({

street:new FormControl(‘’)

});

});

Nested form in the template using formGroupName directive.

<div formGroupName=”address”>

</div>

**patchValue()**

You may want to update parts of the model.

**setValue**() method helps catch nesting errors in complex form, while **patchValue**() fails silently on those errors.

**FormBuilder**

The **Formbuilder** service provides convenient methods for generating controls.

Import the ‘**FormBuilder’** class from the ‘@angular/form’ package.

Inject the FormBuilder service by adding to the component constructor.

Constructor(private fb:FormBuilder){}

The formBuilder has three methods.

1. **Control**()
2. **Group**()
3. **Array**()

Use the group() method to create FormGroup controls.

profileForm = this.fb.group({

firstName:[‘’]

})

**FormArray**()

FormArray is an alternative to FormGroup for managing any number of unnamed controls. Dynamically insert and remove controls from the array instance.

**Validator**

Form validation is used to validate user to ensure its complete and correct.

Import the Validator class from the ‘@angular/form’ package.

Making the field required/minLength/maxLength.

firstName:[‘’,validators.required]

1. **Template-Driven Forms**

Basic form requirement and logic. Data model unstructured. Predictability Asynchronous. Mutable.