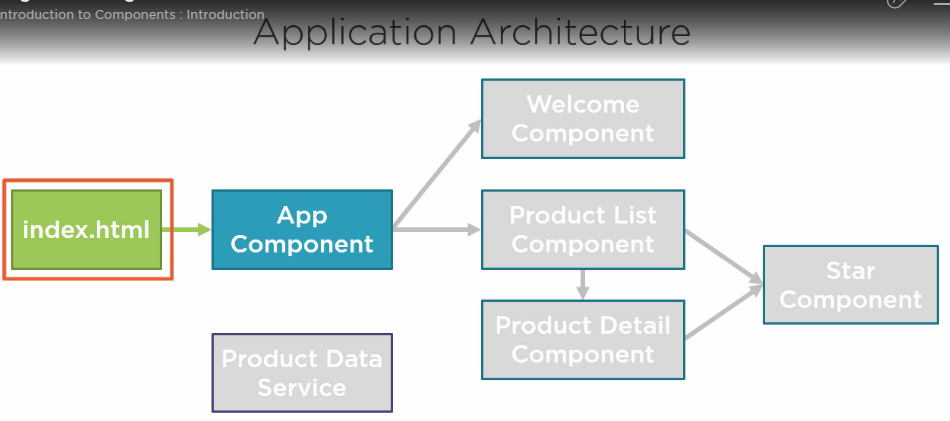
Angular application architecture



Q) WHAT ARE THE FEATURES OF ANGULAR 4.3?

Introducing **HttpClient**, a smaller, easier to use, and more powerful library for making HTTP Requests.

New **router life cycle**events for Guards and Resolvers. Four new events: **GuardsCheckStart**, **GuardsCheckEnd**, **ResolveStart**, **ResolveEnd** join the existing set of life cycle event such as NavigationStart.

Conditionally **disable**

**Q) WHAT ARE THE DIFFERENCES BETWEEN ANGULAR AND ANGULAR JS?**

Angular does not have a concept of “scope” or controllers, instead it uses a hierarchy of components as its main architectural concept

Angular has a different expression syntax, focusing on “[ ]” for property binding, and “( )” for event binding

Mobile development – desktop development is much easier when mobile performance issues are handled first

Modularity – much core functionality has moved to modules, producing a lighter, faster core

Modern browsers only – reducing the need for browser compatibility workarounds

Angular recommends the use of Microsoft’s TypeScriptlanguage, which introduces the following features:

Class-based Object Oriented Programming

Static Typing

Generics

TypeScriptis a superset of ECMAScript 6 (ES6), and is backwards compatible with ECMAScript 5 (i.e.: JavaScript). Angular also includes the benefits of ES6:

Lambdas

Iterators

For/Of loops

Python-style generators

Reflection

Improved dependency injection– bindings make it possible for dependencies to be named

Dynamic loading

Asynchronous template compilation

Simpler Routing

Replacing controllers and $scope with components and directives **– a component is a directive with a template**

Reactive programmingsupport using RxJS

Q) WHAT’S NEW IN ANGULAR 4? AND WHAT ARE THE IMPROVEMENTS IN ANGULAR 4?

Smaller & Faster Apps

View Engine Size Reduce

Animation Package : **Animations package :** @angular/platform-browser/animations

**NgIf and ngFor Improvement**

Template

**NgIf with Else**

Now in Angular 4, possible to use an else syntax as,

<div \*ngIf="user.length > 0; else empty"><h2>Users</h2></div>

<ng-template #empty><h2>No users.</h2></ng-template>

**Use of AS keyword**

A new addition to the template syntax is the “as keyword” is use to simplify to the “let” syntax.

<div \*ngFor="let user of users | slice:0:2 as total; index as = i">

{{i+1}}/{{total.length}}: {{user.name}}

</div>

Pipes

Angular 4 introduced a new “titlecase” pipe “|” and use to changes the first letter of each word into the uppercase.

<h2>{{ 'anil singh' | titlecase }}</h2>

<!-- OUPPUT - It will display 'Anil Singh' -->

HTTP Request Simplified

Adding search parameters to an “HTTP request” has been simplified as,

//Angular 4 -

http.get(`${baseUrl}/api/users`, { params: { sort: 'ascending' } });

//Angular 2-

const params = new URLSearchParams();

params.append('sort', 'ascending');

http.get(`${baseUrl}/api/users`, { search: params });

Apps Testing Simplified

Introduce Meta Tags

A new service has been introduced to easily get or update “Meta Tags” i.e.

@Component({

selector: 'users-app',

template: `<h1>Users</h1>`

})

export class UsersAppComponent {

constructor(meta: Meta) {

meta.addTag({ name: 'Blogger', content: 'Anil Singh' });

}

}

Added some Forms Validators Attributes

One new validator joins the existing “required”, “minLength”, “maxLength” and “pattern”. An email helps you validate that the input is a valid email.

Added Compare Select Options

A new “compareWith” directive has been added and it used to help you compare options from a select.

<select [compareWith]="byUId" [(ngModel)]="selectedUsers">

<option \*ngFor="let user of users" [ngValue]="user.UId">{{user.name}}</option>

</select>

Enhancement in Router

A new interface “paramMap” and “queryParamMap” has been added and it introduced to represent the parameters of a URL.

const uid = this.route.snapshot.paramMap.get('UId');

this.userService.get(uid).subscribe(user => this.name = name);

Added Optional Parameter

Improvement Internationalization

Q) WHAT IS THE USE OF INTERCEPTORS IN ANGULAR 4?

The Interceptors is a common used to set default headers for all responses.

Q) HOW TO SET HTTP REQUEST HEADER IN ANGULAR 4 AND ANGULAR 2?

The HTTP Interceptors are used to Set Http Headers Request in Angular 4 using the import from “@angular/common/http”. The HTTP Interceptors are available in Angular 4.x versions.

The HTTP Interceptors are not supported in Angular 2. We are creating the HttpClient Injectable class to achieve this. You can see the below examples for set http headers request with and without HTTP interceptors.

[] -> property binding

() -> event binding

{} -> data binding

Q) WHAT CLASSES SHOULD I ADD TO MODULE'S DECLARATIONS?

components, directives and pipes classes in the @NgModule.

Q) WHAT CLASSES SHOULD I Do NOT ADD TO MODULE'S DECLARATIONS?

Module, Service, objects, strings, numbers, functions, entity models, configurations, business logic, and helper classes

Q) WHAT HAPPEN WHEN I IMPORT THE SAME MODULE TWICE IN ANGULAR 4?

No problem! We can import the same module twice but Angular does not like modules with circular references.

Q) HOW TO GET AND LOG AN ERROR IN ANGULAR 4?

Two types of errors –

If the backend returns an unsuccessful response like – 404, 500 and so on

If something goes wrong in the client side like -network error etc.

In the both cases – We are using HttpErrorResponse and return the useful information on what went wrong in this call!

Q) WHAT IS JSON WEB TOKEN?

JSON Web Token (JWT) is an open standard which used for securely transmitting information between parties as a JSON object.

The JWTs can be signed with –

HMAC algorithm

RSA algorithm

Q) WHEN SHOULD YOU USE JSON WEB TOKENS

Authentication

Information Exchange

Q) WHAT IS THE JSON WEB TOKEN STRUCTURE

The JSON Web Tokens consist of three parts separated by dots (.), which are:

Header

Payload

Signature

the following Steps are used to building authentication and authorization for RESTful APIs and applications. It might help you –

The users send their credentials to the server which is verified by the database credentials. If everything is verified successfully, the JWT is sent back to them.

The JWT is saved in the user’s browser in local storage or in a cookie and so on.

The presence of a JWT saved in the browser is used as an indicator that a user is currently logged in.

The expiry time of JWT is continually checked to maintain an authenticated state in the Angular applications.

The client side routes are protected and access by authenticated users only.

When user sends the XHR requests for APIs, the JWT gets sent an Authorization header using your cookies or Bearer.

When XHR requests coming on the server, before send back the responses it’s validated first with configured app’s secret keys. If everything is looking good then returns successfully responses other send the back to the bad request.

Q) What Is Angular CLI?

The Angular CLI is a tool to initialize, develop, scaffold and maintain Angular applications.

**Installation –** npm install -g @angular/cli

**Deleting angular CLI -** npm uninstall -g @angular/cli

**Installing latest angular CLI -** npm install -g @angular/cli@latest

1. ng new – create a new application

2. ng serve -– to start the application similarly we have many commands

3. ng generate – ng generate c will generate a new component

4. ng lint

5. ng test

6. ng e2e

7. ng build – will build the project

8. ng get

9. ng set

10. ng doc

11. ng eject

12. ng xi18n

13. and so on

Q) What Is Bootstrapping in Angular 4

**The bootstrapping process creates the components listed in the bootstrap array and inserts each one into the browser (DOM).**

Most of the applications have only one component tree and they bootstrap a single root component and you can call the one root component anything you want but most developers call it AppComponent.

The Angular Module (NgModules) helps us to organize an application into connected blocks of functionality.

The NgModule properties for the minimum “AppModule” generated by the CLI which are follow as -

* Declarations — Use to declare the application components.
* Imports —Every application must import BrowserModule to run the app in a browser.
* Providers — There are none to start.
* Bootstrap — This is a root AppComponent that Angular creates and inserts into the index.html host web page.

By default Bootstrap file is created in the folder “src/main.ts” and “main.ts” file is very stable. Once you have set it up, you may never change it again and its looks like –

Q) Best Way to Set Http Request Header in Angular 4

Steps 1 - Writing an interceptor for adds a header for every request!

**import** {HttpEvent, HttpInterceptor, HttpHandler, HttpRequest} from '@angular/common/http';

**export** **class** AddHttpHeaderInterceptor **implements** HttpInterceptor {

intercept(req: **HttpRequest**<any>, next: **HttpHandler**): Observable<HttpEvent<any>> {

// Get the auth header from the service.

**const** authHeader = 'token\_value';

**const** clonedReq = req.clone({headers: **req.headers.set**('Authorization', authHeader)});

**return** next.handle(clonedReq);

}

}

Steps 2 - Providing your interceptor!

**import** { HTTP\_INTERCEPTORS } from '@angular/common/http';

@NgModule({

providers: [{

provide: **HTTP\_INTERCEPTORS**,

useClass: **AddHttpHeaderInterceptor**,

multi: **true**,

}],

})

**export** **class** AppModule {}

Q) What Is the forRoot Method in Angular 4?

For routing the urls to respective components

**export** **const** sharedConfig: **NgModule** = {

bootstrap: [ AppComponent ],

declarations: [

AppComponent,

NavMenuComponent,

HomeComponent,

UserComponent,

BarCodePipe

],

imports: [

RouterModule.forRoot([

{ path: '', redirectTo: 'home', pathMatch: 'full'},

{ path: 'home', component: **HomeComponent** },

{ path: 'user', component: **UserComponent** },

{ path: '\*\*', redirectTo: 'home' }

])

],

providers: [UserService]

Q) If Else and Then Conditions

1) <element \*ngIf="[condition expression]; else [else template]"> </element>

2)

<div \*ngIf="users | async; else loadingGrid; let user">

<p>{{user.Id}}</p>

<p>{{user.name}}</p>

<p>{{user.Age}}</p>

</div>

<ng-template #loadingGrid>loading...</ng-template>

3)

<div \*ngIf="isValid;then then\_content else other\_content">If IsValid then display other</div>

<ng-template #then\_content>content here...</ng-template>

<ng-template #other\_content>other content here...</ng-template>

Routing Example:

In html

<ul **class**='nav navbar-nav'>

<li [routerLinkActive]="['link-active']">

<a [routerLink]="['/login']">

<span **class**='glyphicon glyphicon-Login'></span> Login

</a>

</li>

<li [routerLinkActive]="['link-active']">

<a [routerLink]="['/registration']">

<span **class**='glyphicon glyphicon-Register'></span> Register

</a>

</li>

<li [routerLinkActive]="['link-active']">

<a [routerLink]="['/Users']">

<span **class**='glyphicon glyphicon-Users'></span> Users

</a>

</li>

<li [routerLinkActive]="['link-active']">

<a [routerLink]="['/Dashboard']">

<span **class**='glyphicon glyphicon-Dashboard'></span> Dashboard

</a>

</li>

</ul>

In .ts

**const** appRoutes: **Routes** = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'registration', component: **RegistrationComponent**},

{ path: 'login', component: **LoginComponent**},

{ path: ''home/:id', component: HomeComponent, data: { title: 'Home' }},

{ path: 'users', component: UserListComponent, data: { title: 'User List' }},

{ path: 'dashboard', component: DashboardComponent, data: { title: 'Dashboard' }}

{ path: '\*\*', component: **PageNotFoundComponent** }

];

@NgModule({

imports: [

UniversalModule,

RouterModule.forRoot(**appRoutes**),

FormsModule,

ReactiveFormsModule

],

})

**@angular/forms** is to import form modules

**Router State -**

After the end of each successful navigation life-cycle, the router builds a tree of Activated-Route objects that make up the current state of the router. You can access the current Router-State from anywhere in the application using the Router service and the router-State property.

Each Activated-Route in the Router-State provides methods to traverse up and down the route tree to get information from parent, child and sibling routes.

**Router Events -**

1.     **NavigationStart**- This event is triggered when navigation starts.

2.     **RoutesRecognized** - This event is triggered when Router parses the URL and routes are recognized.

3.     **RouteConfigLoadStart**- This event is triggered before the Router lazy loads a route configuration.

4.     **RouteConfigLoadEnd** - This event is triggered after a route has been lazy loaded.

5.     **NavigationEnd** - This event is triggered when navigation ends successfully!

6.     **NavigationCancel** - This event is triggered when navigation is cancelled.

7.     **NavigationError** - This event is triggered when navigation fails due to error.

Angular 2 component/directive has lifecycle events, managed by @angular/core. It creates the component, renders it, creates and renders its children, processes changes when its data-bound properties change, and then destroys it before removing its template from the DOM. Angular provides a set of lifecycle hooks(special events) which can be tapped into this lifecycle and perform operations when required. The constructor executes prior to all lifecycle events. Each interface has a single hook method prefixed with ng. For example, *ngOnint*interface has *Oninit*method that must be implemented in the component.

Some of the events are applicable for both component/directives while few are specific to components.

* **ngOnChanges**: Responds when angular sets its data-bound property which receives the current and previous object values.
* **ngOnInit**: Initializes the component/directive after first ngOnChange triggers. This is most frequently used method to retrieve the data for the template from a back-end service.
* **ngDoCheck**: Detect and act upon changes occuring outside Angular context. It is called when every change detection run.
* **ngOnDestroy**: Cleanup just before Angular destroys the directive/component. Unsubscribe observables and detach event handlers to avoid memory leaks.

**Component-specific hooks:**

* **ngAfterContentInit**: Component content has been initialized
* **ngAfterContentChecked**: After Angular checks the bindings of the external content that it projected     into its view.
* **ngAfterViewInit**: After Angular creates the component’s view.
* **ngAfterViewChecked**: After Angular checks the bindings of the component’s view.

Q) Lazy loading and How to enable lazy loading in angular 2

Lazy lading enables us to load only the module user is interacting and keep the rest to be loaded at runtime on demand.

Every Angular application must have one main module say AppModule. The code should be splitted into various child modules (NgModule) based on the application business case.