By László Kővári

Angular Interview questions / answers

(The following questions are based on my daily work, what we used on daily basis)

1. ## What is Angular CLI?

   Command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.
   Command Line Interface Provides more command line commands, for example ng new, ng g (version 6 and above a (a means application) and l), ng serve, ng build, ng test etc.
   Compiles TypeScript to JavaScript Uses Nodejs in the background for Angular Compilation

2. ## What are the following commands for?

   create new Angular *project* and create new *component* *ng new <project-name> ng g c <component-name>*

3. ## What are the most used naming conventions for Angular components? Provide an example for “test equipment” component.

   test-equipment.component.ts -> ng g c test-equipment

4. ## What is NPM?

   Node Package Manager, helps in installing packages for an Angular application.

5. ## Do you know other package manager? (which similar the NPM)

   The Yarn which is a fast, reliable, and secure dependency management tool. The benefit is of the Yarn is guaranties, whatever installation did on a machine, that is the exact same create dependencies that is going to be installed on another machine.

6. ## How to build an Angular project?

   ng build

7. ## How to run an Angular project

   ng serve

8. ## What does the *npm install* command do when you run it in your project folder?

   Install the runtime and development dependencies based on the package.json file.

9. ## What structural directives do you know?

```
<div *ngIf=”condition; else elseBlock ”>
  then block
</div>
<ng-template #elseBlock>
  elseBlock
</ng-template>
```

*<div **ngFor**=”let something of someCollection; let ix = index”>*

[ngSwitch]

```
<container-element [ngSwitch]="switch_expression">
  <some-element *ngSwitchCase="match_expression_1">

  </some-element>
  <some-element *ngSwitchCase="match_expression_2">

  </some-element>
<container-element>
```

10. What is the difference between the following rows?

*<div [hidden]="<logical expression>"> *
*<div *ngIf="<logical expression>"> *

The difference is, if the [hidden] expression is true then it will hide the content and will not remove it from the DOM, if the *ngIf expression is false then it will hide the content and it will be removed from the DOM.

11. What are the built-in structural directives?

ngIf conditionally creates or disposes of subviews from the template.

```
<div *ngIf="isLoggedIn; else loggedOut">

</div>

<ng-template #loggedOut>

</ng-template>
```

ngFor repeat a node for each item in a list.

```
<div *ngFor="let item of items; let i=index trackBy: trackByItems"">
        {{i + 1}} - {{item.name}}
</div>
trackByItems(index: number, item: Item): number { return item.id; }
```

ngSwitch a set of directives that switch among alternative views.

```
<div [ngSwitch]="currentItem.feature">
        <app-stout-item *ngSwitchCase="'stout'" [item]="currentItem">
        </app-stout-item>
        <app-device-item *ngSwitchCase="'slim'" [item]="currentItem">
        </app-device-item>
        <app-lost-item *ngSwitchCase="'vintage'" [item]="currentItem">
        </app-lost-item>
        <app-best-item *ngSwitchCase="'bright'" [item]="currentItem">
        </app-best-item>
```

```
        <!-- . . . -->
        <app-unknown-item *ngSwitchDefault [item]="currentItem">
        </app-unknown-item>
</div>
```

<ng-content> so you don't have to introduce extra levels of HTML.

12. What built-in attribute directives do you know in Angular?

```
<div [ngStyle]="{'property': 'value'}">

<div [ngClass]="{'active': isActive, 'disabled': isDisabled}">
<input [(ngModel)]="currentItem.name" id="example-ngModel">
```

13. What is the pipe?

Decorator that marks a class as pipe and supplies configuration metadata.

name

The pipe name to use in template bindings. Typically uses lowerCamelCase because the name cannot contain hyphens.

pure?

When true, the pipe is pure, meaning that the transform() method is invoked only when its input arguments change. Pipes are pure by default.

14. What built-in pipes do you know?

currency

{{ price | currency:'USD':false:3.2-2 }}

date -> ‘medium’, ‘short’, ‘fullDate’, ‘longDate’, ‘mediumDate’, ‘shortDate’, ‘mediumTime’ and ‘shortTime’

{{ someDate | date:'fullDate' }}

number

{{ decimalValue | number:'4.3-5' }}

percent

```
<p>A: {{a | percent}}</p>
```

lovercase

```
<pre>'{{value | lowercase}}'</pre>
```

uppercase

```
{{ value_expression | uppercase }}
```

15. What ways do you know to create data entry forms in Angular?

- the template-driven approach which uses HTML with two-way data binding
- the reactive approach manages the form and it's data in the component code

16. What are the most commonly used built-in validators in Angular?

required
minLength
maxLength
min
max
email
pattern

15. What are the most commonly used Class Decorators in Angular?

@Component
@Directive
@Pipe
@Injectable

16. What are the most commonly used Class field Decorators in Angular?

@Input
@Output
@HostBinding
@HostListener
@ContentChild
@ContentChildren
@ViewChild
@ViewChildren

17. What is the purpose of @NgModule decorator in Angular?

Defines a module that contains components, directives, pipes, and providers.

18. What is a most commonly used way to pass data into component?

@Input() variable: type; // input property

19. When you write a setter or getter for an @Input()?

-setter: when we should **intercept input property changes** and do something when it's occurred
-getter: when we should intercept access to an input property, for example when **we want to put the input data in the HTML**.

20. What is the purpose of the @Output decorator in the component?

When we would like to emit an event using and EventEmitter we use @Output

21. How to define reference in an Angular template?

Use the *#<referenceName>*

22. Which way can you use @ViewChild()?

- *@ViewChild('referenceName') variable: ElementRef;*
- *@ViewChild(ComponentName) variable: ComponenClassName;*

23. What is the ControlValueAccessor and what is the purpose of it?

Acts as a bridge between the Angular forms API and a native element in the DOM

24. What are the NgControlStatuses, and what are the meanings of each?

-*ng-valid*: the form control or form is valid
-*ng-invalid*: the form control or form is invalid
-*ng-pending*: when an asynchrony validation is in process, during it the status is pending
-*ng-pristine*: the form control or form has not yet been modified, clear
-*ng-dirty*: the form control or form is modified
-*ng-touched*: the form control or form is touched, the user clicked into the form control
-*ng-untouched*: no user interaction on the FormControl

25. What does *novalidate* means in the template when defined on a <form>?

<form (ngSubmit)="onSubmit(formData)" #formData="ngForm" ***novalidate****>*
novalidate attribute is used to disable the browser's native form validation

26. What is a Service in Angular and what is a typical usage of it?

A service is used when a common functionality needs to be provided to various modules. For example, we could have a database functionality that could be reused among various modules.
-to load and to store persistent information over your application
-to communicating between components

27. What is FormBuilder and when can we use it?

The FormBuilder is a class which is provided by Angular and when we want to create a Reactive form we inject it into our component class and build our Reactive form with it.

28. How to bind fields in the template to a FormControl element when we create a Reactive form?

-set the formControlName attribute of the element in the HTML template:
*<input formControlName="****firstName****" >*

*<input formControlName="****lastName****" >*

-set the form control name in the component code:

```
constructor(@Inject(FormBuilder) fb: FormBuilder) {    this.form
= fb.group(
  {
    personName: fb.group({         firstName:
['John', Validators.minLength(3)],         lastName:
['Doe', Validators.minLength(3)]

  })
 });
```

29. What is the RouterOutlet?

The RouterOutlet is a directive from the router library and it acts as a placeholder that marks the spot in the template where the router should display the components for that outlet. Router outlet is used like a component

30. What is the call order of Route guards?

canLoad, *canDeactivate*, *canActivate*, *resolve*, *canActivateChild*

Right call order is. 1. canDeactivate, 2. canLoad, 3. canActivateChild, 4. canActivate, 5. resolve

31. What is a typical usage of a canActivate guard?

The typical usage of *canActivate* is to prevent navigation to a route when necessary. For example, to check user rights.

32. What is the typical usage of resolve guard?

The typical usage of the resolve, to prefetch data before activating a route. With other words the router waits for the data to be resolved before the route is finally activated.

33. What is the typical usage of canDeactivate?

Prevent navigating away from a route. For example, save unsaved data before navigate away.

34. How can we access the component in the canDeactivate guard?

It has a component parameter that represents the component where we navigated away

35. What is the purpose of LazyLoading?

Lazy loading is one of the most useful concepts of Angular Routing. It helps us to download the web pages in chunks instead of downloading everything in a big bundle. It is used for lazy loading by asynchronously loading the feature module for routing whenever required using the property loadChildren. Let's load both Customer and Order feature modules lazily. We use lazy loading to speed up our application startup. We can use preloading to quickly launch our lazy loaded routes.

36. What preloading strategies do you know, and which is the default?

PreloadAllModules : Provides a preloading strategy that preloads all modules as quickly as possible.

NoPreloading : Provides a preloading strategy that does not preload any modules, enabled by default.

37. What is the difference between a regular function and an arrow function?

The keyword “this” in the simple function refers to those declarations which are inside the function scope. If we need to refer to the outside declarations we should use .bind(this)
The keyword “*this”* in an arrow function refers to all inside and outside declarations, we do not need to use *.bind()*

38. What is the purpose of the ngModelGroup directive in Angular?

Group logically the input fields into a sub-group which are logically cohesive (belongs to the same group) e.g., items of an address (street, city, country, zip, etc.) and validates it separately. One of the other usages of the *ngModelGroup* is when we are creating a nested component that will collect the input fields into a sub-group.

39. What is the difference between the following view providers?

viewProviders: [{ provide: ControlContainer, useExisting: NgModelGroup }]
viewProviders: [{ provide: ControlContainer, useExisting: NgForm }]
The difference is the component will be tied in below the parent *ngModelGroup* (1st row) or *ngForm* (2nd row)

40. Where can we define the canDeactivate guard if we need to use the component parameter value when the module loads lazily?

The *canDeactivate* guard should be defined in the routing module of the lazy loaded module, because if we define it in the parent routing module, the component will remain always null, because the module loading lazily.

41. How can you implement the route resolve content of two different service results in the following route resolve function?

```
resolve(route: ActivatedRouteSnapshot, _state: RouterStateSnapshot): Observable<any> |
Promise<any> { const someId =
    route.parent.parent.params['someId'];
const dataObservable1 = this.dataService1(someId);
const dataObservable2 = this.dataService2(someId);
```

For example: return Observable.combineLatest(dataObservable1, dataObservable2);

42. What are the Angular component Lifecycle Hooks? Please describe the call order of them.

Constructor, ngOnChange, ngOnInit, ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, ngAfterViewChecked, ngOnDestroy

43. What does the ngDoBootstrap() method do? Is it a real life-cycle hook?

Hook for manual bootstrapping of the application instead of using bootstrap array in @NgModule annotation.

```
class AppModule implements DoBootstrap {
ngDoBootstrap(appRef: ApplicationRef) {
   appRef.bootstrap(AppComponent); // Or some other component
  }
}
```

44. What is the AngularJS ng-hide directive equivalent solution in Angular (v2+), and why important it, which cases can we use?

- hidden property, we can use it with property binding.
- in cases we do not need to remove elements from the DOM, just visually hide them.
- for example, when we have live data in an input form e.g. On a tab which is currently hidden, but the validation should roll up its validation results to the form

45. What is the Angular content projection? Describe one advantage of using it?

One of the Angular features that help us the most in building reusable components is Content Projection and the <ng-content> element. Content projection is a pattern in which you insert, or project, the content you want to use inside another component.
Any component can be reusable. But what makes content projection so powerful is the ability to change the content inside of a component based on the needs of the application.

46. What template type checking do you know, and where can you set it?

- *fullTemplateTypeCheck* (deprecated in Angular v13 instead of it can use strictTemplates)

47. What are the three fundamental building blocks of Angular forms, (inherited from the AbstractControl), and which cases used that?

- *FormControl – data field*
- *FormGroup – data input form*

- *FormArray – array of data input form*

48. What is the network aware preloading strategy?

Preloading is an important feature of Angular Router's lazy loading. This is available since 2.1.0. By default, when the application uses lazy loading with loadChildren, chunked lazy modules will be loaded on-demand. It can reduce initial bundle size but users have to wait for loading of chunks on transition.
Preloading changes that. By preloading, the application will start loading chunked modules before needed. It can improve user experience with smooth transition.

```
@NgModule({
  imports: [
    RouterModule.forRoot(routes, {
      preloadingStrategy: NetworkAwarePreloadStrategy
    })
  ],
  exports: [RouterModule]
})
export class AppRoutingModule {}

export declare var navigator;

@Injectable({ providedIn: 'root' })
export class NetworkAwarePreloadStrategy implements PreloadingStrategy {
  preload(route: Route, load: () => Observable<any>): Observable<any> {
    return this.hasGoodConnection() ? load() : EMPTY;
  }


  hasGoodConnection(): boolean {
    const conn = navigator.connection;
    if (conn) {
      if (conn.saveData) {
        return false; // save data mode is enabled, so dont preload
      }
      const avoidTheseConnections = ['slow-2g', '2g' /* , '3g', '4g' */];
      const effectiveType = conn.effectiveType || '';
      console.log(effectiveType);
      if (avoidTheseConnections.includes(effectiveType)) {
        return false;
      }
    }
    return true;
  }
}
```

49. What is the provider?

A provider is an instruction to the Dependency Injection system on how to obtain a value for a dependency. Most of the time, these dependencies are services that you create and provide.

50. How to inject dynamic script in Angular?

```
import { Component, OnInit } from '@angular/core';
import { DomSanitizer } from '@angular/platform-browser';
@Component({
  selector: 'my-app',
  template: `
     <div [innerHtml]="htmlSnippet"></div>
  `,
})
export class App {
     constructor(protected sanitizer: DomSanitizer) {}
htmlSnippet: string =  this.sanitizer.bypassSecurityTrustScript("<script>safeCode()</script>");
}
```

51. How to inject programmatically a service?

Include the injector into constructor of the component

```
private injector: Injector
```

Get the service from the injector.

```
var zipCodeService = this.injector.get(ZipCodeService);
```

52. What is the purpose of wildcard route?

If the URL doesn't match any predefined routes then it causes the router to throw an error and crash the app. In this case, you can use wildcard route. A wildcard route has a path consisting of two asterisks to match every URL.

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

53. Which are the router events?

```
NavigationStart,
RouteConfigLoadStart,
RouteConfigLoadEnd,
RoutesRecognized,
GuardsCheckStart,
ChildActivationStart,
ActivationStart,
GuardsCheckEnd,
ResolveStart,
ResolveEnd,
ActivationEnd
ChildActivationEnd
NavigationEnd,
NavigationCancel,
NavigationError
Scroll
```

54. What is the purpose of async pipe?

The AsyncPipe subscribes to an observable or promise and returns the latest value it has emitted. When a new value is emitted, the pipe marks the component to be checked for changes.

```
@Component({
 selector: 'async-observable-pipe',
 template: `<div><code>observable|async</code>:
     Time: {{ time | async }}</div>`
})
export class AsyncObservablePipeComponent {
 time = new Observable(observer =>
  setInterval(() => observer.next(new Date().toString()), 2000)
 );
}
```

55. What is the data binding?

Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data. There are four forms of data binding (divided as 3 categories) which differ in the way the data is flowing.

Interpolation: {{ value }}
Property binding: [property]="value"
Event binding: (event)="methode"
Two-way data binding: [(ngModel)]="value"

56. What are the lifecycle hooks in Angular?

ngOnChanges()

Respond when Angular sets or resets data-bound input properties. The method receives a SimpleChanges object of current and previous property values.
Note that this happens very frequently, so any operation you perform here impacts performance significantly. See details in Using change detection hooks in this document.
Called before ngOnInit() (if the component has bound inputs) and whenever one or more data-bound input properties change.
Note that if your component has no inputs or you use it without providing any inputs, the framework will not call ngOnChanges().

ngOnInit()

Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties. See details in Initializing a component or directive in this document.
Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not (which is the case when there are no template-bound inputs).

ngDoCheck()

Detect and act upon changes that Angular can't or won't detect on its own. See details and example in Defining custom change detection in this document.
Called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.

ngAfterContentInit()
Respond after Angular projects external content into the component's view, or into the view that a directive is in.
See details and example in Responding to changes in content in this document.
Called once after the first ngDoCheck().

ngAfterContentChecked()
Respond after Angular checks the content projected into the directive or component.
See details and example in Responding to projected content changes in this document.
Called after ngAfterContentInit() and every subsequent ngDoCheck().

ngAfterViewInit()
Respond after Angular initializes the component's views and child views, or the view that contains the directive.
See details and example in Responding to view changes in this document.
Called once after the first ngAfterContentChecked().

ngAfterViewChecked()
Respond after Angular checks the component's views and child views, or the view that contains the directive.
Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

ngOnDestroy()
Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks. See details in Cleaning up on instance destruction in this document.
Called immediately before Angular destroys the directive or component.

57.

Angular Version Specific News/"questions"

1. What is the new in Angular v5?

1.Http deprecated, use instead of it, HttpClient, for example
import { HttpClientModule } from '@angular/common/http';

2.Support for Multiple Export Alias

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
styleUrls: ['./app.component.css'],
  exportAs:'dashboard, logBoard'
})
```

3.Internationalized Number, Date, and Currency Pipes

Angular 5 ships with new number, date, and currency pipes that increase standardization across browsers and eliminate the need for i18n polyfills.

4. Improved Decorator Support in Angular 5

provider: [{provide: 'token', useFactory: () => null}]

Before Angular 5
provider: [{provide: 'token', useValue: calculated()}]

5. Build Optimization

6. Angular Universal Transfer API

The Angular Universal team has added Domino to the platform-server. This simply means more DOM manipulations can happen out of the box within server-side contexts. Furthermore, two modules, *ServerTransferStateModule* and *BrowserTransferModule* have been added to Angular Universal.
7.Faster Compiler in Angular 5

8. Forms Validation in Angular 5

Now have the ability to decide when the validity and value of a field or form are updated via on blur or on submit

<input name="nickName" ngModel [ngModelOptions]="{updateOn: 'blur'}"> <form [ngFormOptions]="{updateOn: 'submit'}">

```
ngOnInit() {
  this.newUserForm = this.fb.group({
    userName: ['Bob', { updateOn: 'blur', validators: [Validators.required] }]
  });
}
```

9. Animations in Angular 5

In Angular 5, we have two new transition aliases, :increment and :decrement.

10. New Router Lifecycle Events in Angular 5

Some new lifecycle events have been added to the router. The events are GuardsCheckStart, ChildActivationStart, ActivationStart, GuardsCheckEnd, ResolveStart, ResolveEnd, ActivationEnd, and ChildActivationEnd. With these events, developers can track the cycle of the router from the start of running guards through to completion of activation.

11. Better Support for Service Workers in Angular 5

In Angular 5, we have better support for service workers via the @angular/service-worker package. The service worker package is a conceptual derivative of the @angular/serviceworker package that was maintained at github.com/angular/mobile-toolkit, but has been rewritten to support use-cases across a much wider variety of applications

12. Deprecations and Other Updates

NgFor has been removed as it was deprecated since v4. Use NgForOf instead. This does not impact the use of *ngFor in your templates.
The compiler option enableLegacyTemplate is now disabled by default as the <template> element was deprecated since v4. Use <ng-template> instead. The option enableLegacyTemplate and the <template> element will both be removed in Angular v6.

The method ngGetContentSelectors() has been removed as it was deprecated since v4. Use ComponentFactory.ngContentSelectors instead.

ReflectiveInjector is now deprecated. Use Injector.create as a replacement.

NgProbeToken has been removed from @angular/platform-browser as it was deprecated since v4. Import it from @angular/core instead.

2. What is the new in Angular v6?

1.Improved Service Worker Support

```
self.addEventListener('install', event => { self.skipWaiting(); });
self.addEventListener('activate', event => {
event.waitUntil(self.clients.claim());   self.registration.unregister().then(
     () => { console.log('NGSW Safety Worker - unregistered old service worker'); });
});
```

2. Goodbye Template Element Before:
`<template>some template content</template>` Now:
`<ng-template>some template content</ng-template>`

3. Better URL Serialization

In Angular 6, issues like the one above have been fixed and:

URI fragments will now serialize the same as query strings.
In the URL path, (portion prior to the query string and/or fragment), the plus sign (+) and ampersand (&) will appear decoded.
In the URL path, parentheses values ((and)) will now appear percent-encoded as %28 and %29 respectively.
In the URL path, semicolons will be encoded in their percent encoding %3B.
It's also important to know that parentheses and semicolons denoting auxiliary routes will, in any case, show up in their decoded shape except for parentheses and semicolons used as values in a segment or key/value pair for matrix params which will show up encoded.

4. Support Added for Custom Elements

With this support, developers can simply register Angular Components as Custom Elements. Once registered, these components can be used just like built-in HTML elements. They are HTML Elements, so why not?

```
//my-name.ts

import { Component, Input, NgModule } from '@angular/core';
import { createCustomElement } from '@angular/elements';

@Component({
  selector: 'my-name',
  template: `<h1>Hello my name is {{name}}</h1>`
})

export class MyName {
  @Input() name: string = 'Prosper!';
}

@NgModule({
```

```
  declarations: [ MyName ],
  entryComponents: [ MyName ]
})

export class MyNameModule {}

//app.component.ts
import { Component, NgModuleRef } from '@angular/core';
import { createCustomElement } from '@angular/elements';

import { MyName } from './my-name.ngfactory';

@Component({
  selector: 'app-root',   templateUrl:
'./app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  constructor(private injector: Injector, ngModuleRef: NgModuleRef) {
const ngElementConfig = createCustomElement(MyName, injector);
customElements.define('my-name', NgElementConfig);
  }
}
```

5. Forms Validation in Angular 6

Before now, *ngModelChange* was always emitted before the underlying form control was updated.

If you had a handler for the *ngModelChange* event that checked the value through the control, the old value would be logged instead of the updated value. This is not the case if you pass the value through the $event keyword directly.

Check out this example:

Passing the value through the $event keyword directly

```
  <input [(ngModel)]="name" (ngModelChange)="onChange($event)">
onChange(value) {
   console.log(value); // logs updated value
                                                                    }
```

Using a Handler

```
 <input #modelDir="ngModel" [(ngModel)]="name"
 ngModelChange)="onChange(modelDir)">
onChange(ngModel: ngModel) {
  console.log(ngModel.value); // logs old value
```

In Angular 6, *ngModelChange* is now emitted after the value is updated on its form control.

```
(


}
```

"In Angular 6, *ngModelChange* is now emitted after the value is updated on its form control."

6. Multiple Validators for Form Builder Array

In previous versions of Angular, you could set only one validator on a *FormArray* field with the *FormBuilder.array* method.

In Angular 6, you can set multiple validators with the *FormBuilder.array* method:

```
...
ngOnInit() {
  this.speakerForm = this.formBuilder.group({
    text: ['', Validators.required],
    options: this.formBuilder.array([], [MyValidators.correctProposalCount,
MyValidators.totalProposals])
  });
}
```

7. Token Marking for Runtime Animation Context

In Angular 6, it's now possible to determine which animation context is used for a component at runtime. A token is provided as a marker to determine whether the component is running a *BrowserAnimationsModule* or NoopAnimationsModule context at runtime.

8. Hello Schematics

Schematics is a new scaffolding library that's used by the Angular CLI to generate custom templates. The Angular team has always been keen on improving developer productivity, which explains the birth of schematics.

With Schematics, you can easily create Angular libraries like so:

First, install the necessary schematic tools:

npm i -g ng-lib-schematics @angular-devkit/core @angular-devkit/schematics-cli
Next, create a new angular-cli project:

ng new avengers --skip-install // avengers is the name of the new library I'm trying to create
Finally, you can just run schematics like so:

schematics ng-lib-schematics:lib-standalone --name avengers
A new lib directory will be generated for you inside the src folder. The lib directory ships with a sample demo and the build tools necessary for a typical Angular package.

Check out this deep and excellent guide to Schematics.

9. Deprecations and Other Updates Angular 6 ships with *Rxjs* 6.0.0.
@Injectable now supports tree-shakeable tokens.
Service workers now properly handle invalid hashes in all scenarios.
The router sometimes hits a race condition while a route is being instantiated and a new navigation request arrives. This issue has been solved in Angular 6. Avoid overriding *ngInjectableDef* in the decorator if present on the type

3. What is the new in Angular v7?

1.TypeScript 3.1 Support

2. Updated Domino to v2.1.0

3. RxJS 6.3

4. Added support for Node 10, still support 8

5. Ivy Renderer
Speed Improvements, Size Reduction, Increased Flexibility

6. Angular Compatibility Compiler (ngcc)

The **ngcc -Angular node_module compatibility** compiler – The ngcc is a tool which "**upgrades**" node_module compiled with **non-ivy ngc** into **ivy compliant** format.

This compiler will convert **node_modules** compiled with **Angular Compatibility Compiler (ngcc)**, into node_modules which appear to have been compiled with **TSC compiler transformer (ngtsc)** and this compiler conversion will allow such "legacy" packages to be used by the Ivy rendering engine.

TSC transformer which removes and converts **@Pipe, @Component, @Directive and @NgModule** to the corresponding **definePipe, defineComponent, defineDirective and defineInjector.**

7. DoBootstrap

Angular 7 added a new lifecycle hook that is called ngDoBootstrap and an interface that is called DoBootstrap.

```
//ngDoBootstrap - Life-Cycle Hook Interface
class AppModule implements DoBootstrap {
  ngDoBootstrap(appRef: ApplicationRef) {
    appRef.bootstrap(AppComponent);
  }
                                          }
```

8. Updated XMB Placeholders

Updated XMB placeholders() to include the original value on top of an example. Now placeholders can by definition have one example() tag and a text node. The text node is used by TC as the “original” value from the placeholder, while the example should represent a dummy value. Let’s take a look examples below:

A message like Name: {{yourName}} would generate this xmb message in current and new behaviour:

Current Behaviour :

```
<msg id=123>Name: <ph name="INTERPOLATION"><ex>{{yourName}}</ex></ph></msg>
```

New Behaviour :

```
<msg id=123>Name: <ph
name="INTERPOLATION"><ex>{{yourName}}</ex>{{yourName}}</ph></msg
>
```

9. Added UrlSegment[] to CanLoad interface.

Current Behaviour :
The Route object as passed to implementations of CanLoad which only provides minimal information on the page which should be navigated to.

New Behaviour :
Additionally to Route an UrlSegment[] is passed to implementations of CanLoad as a second parameter. It contains the array of path elements the user tried to navigate to before canLoad is evaluated.

10. Other Angular 7 Important Features
Produce a warning when router navigation triggered outside Angular zone with NgZone enabled on dev mode.

Enable Shadow DOM v1 and slots

Adds a new optional parameter to the renderer.selectRootElement method:

Before:

renderer.selectRootElement(rootSelectorOrNode);
renderer.selectRootElement(rootSelectorOrNode, preserveContents?:boolean);
preserveContents defaults to false to preserve existing renderer behavior. If set to true, the renderer will not remove the existing contents of a root element when bootstrapping a component.

This is primarily useful when combined with the ViewEncapsulation.ShadowDom Renderer – the nodes are preserved, and developers can use <slot> elements to re-project the light DOM nodes.

Added support to extend angularCompilerOptions

Current Behaviour :

TypeScript only supports merging and extending of compilerOptions.

New Behaviour :

This is an implementation to support extending and inheriting of angularCompilerOptions from multiple files.

Bazel :

Added additional parameters : strip_export_pattern and allow_module_identifiers to ts_api_guardian_test.

Ivy :

Allow combined context discovery for components, directives, and elements Resolve references to vars in .d.ts files :
Previously, if ngtsc encountered a Variable Declaration without an initializer, it would assume that the variable was undefined, and return that result.

However, for symbols exported from external modules that resolve to .d.ts files, variable declarations are of the form:

export declare let varName: Type;
This form also lacks an initializer, but indicates the presence of an importable symbol which can be referenced. This feature changes the static resolver to understand variable declarations with the declare keyword and to generate references when it encounters them.

Support animation *@triggers* in templates.
Support bootstrap in *ngModuleDef*.

4. What is the new in Angular v8?

1. Angular Ivy (Angular's next-generation compilation and rendering pipeline)
For over a year, the Angular Team has been telling about their new Ivy renderer, so it is obviously the major part of the Angular 8 update.

Ivy renderer is basically the new compiler of Angular framework. Though our Angular developers were really hoping for the final release of the new compiler, but unfortunately the Ivy renderer is only available as an opt-in preview in Angular 8 update.

Compared with the current Angular compiler, Ivy renderer will offer the following benefits:

Template type checking will be improved, which will allow catching more errors at build time
Payload time will be decreased, which will help browser download and parse applications faster
Rebuild times will also get faster

2. Router Location
A bunch of new stuff has been added to the location services in the new Angular 8 update.

Platformlocation, for example, provides access to the protocol, port, and hostname.

There is also a new *getstate*() method, which will help developers get the history.state.

All of these updates are really useful if you're using *ngUpgrade*.

Lazy Loading with *Import*() Sytax
The Angular 8 update includes a new way to declare lazy-loading routes using the TypeScript syntax – *import*().

In the latest Angular version, this will be the preferred way of declaring a lazy-loading route and the Ivy renderer will only support this.

Here's how you can change the lazy-loading route for Angular 8 for example:

Old way:

loadChildren: '.admin/admin.module#AdminTestModule'

New Way:

loadChildren: () => import('./races/races.module').then(m => m.RacesModule)

As you can see, the schematic offered by CLI automatically migrates your declarations, making it easier to run ng update *@angular.cli.*

3. Forms

FormArray.clear

The *FormArray* in the new Angular 8 now offers a clear method so that it becomes quicker to remove all the control it contains.

In earlier version of Angular, developers had loop over these controls and remove them one by one.

```
// `customers` is initialized with 2 customers const
customers = fb.array([customer1, customer2]);
customers.clear(); // customers is now empty
```

markAllAs Touched

The AbstractControl class in Angular now has a new method – *markAllAsTouched*, in addition to the existing *markAsTouched*, *markAsPending*, and *markAsDirty* methods.

As the name suggests, the *markAllAs* Touched method will mark an *AbstractControl* as touched and also its descendants.

AbstractControl is basically the parent class of *FormControl, FormArray*, and *FormGroup*, making the method available on all of the reactive form entities.

Here's the syntax to use this new method:

```
form.markAllAsTouched();
```

4. Support for TypeScript

Angular is finally going to include updates to the latest version of Angular's dependencies, which are mainly TypeScript and RxJS.

In fact, it is now mandatory to upgrade to TypeScript 3.4 in the Angular 8 update.

Though it may look like just a small improvement, but if you look at the positive side then this update is extremely good since the TypeScript team always introduces new features in their every new release.

5. Service Worker Registration Strategy

The service worker registration in the Angular 8 update now has a new option to specify when the registration should take place. Earlier, the service worker had to wait for the app to be stable in order to avoid slowing the start of the application.

For example, if you had run a recurring asynchronous task in the previous version of Angular, the service worker would never get registered as Angular would consider application to be not

stable due to the recurring task.

With Angular 8, however, you can use the new registrationStrategy option to handle the registration of the service worker.

For instance, suppose you want to register the service worker after 5 seconds. Here's how you would do it in Angular 8.

```
providers: [
ServiceWorkerModule.register('/ngsw-worker.js', {
enabled: environment.production,
registrationStrategy: 'registerDelay:5000'
}),
// …
]
```

Bypass a Service Worker
In the new Angular 8 update, you can now also bypass the service worker for a particular request using the new ngsw-bypass header.

```
this.http.get('api/users', { headers: { 'ngsw-bypass': true } });
```

Multiple Apps on Sub-Domains
In the earlier version of Angular, developers had no option to have multiple apps use @angular/service-worker on different sub-domains because that would cause the service worker to overwrite the caches of other sub-domains.

Now in Angular 8, this glitch has been fixed.

6. Support for Bazel
Bazel in Angular 8 is an opt-in option for now, but it is expected to be included in the Angular CLI in version 9.

Bazel basically offers fast build time, which means you can build your application through an incremental build and deploy the last changes only.

7. No More *@angular/http*
The Angular 8 will no longer support *@angular/http*. So, if you have an existing Angular application built using the previous version, then you need to make appropriate adjustments to the code and replace *@angular/http with @angular/common/http.*

8. Builders API
The Angular 8 update has now added support to use Builders API, also referred as Architect API.

Builder is basically just a function with a bunch of commands that you pass to *createbuilder*() method through the *@angular-devkit/architect package*.

These builders are basically used for the following main operations:

Serve
Build
Test
Lint
e2e
In addition, developers can now also create their own custom builders in addition to the built-in builders from here.

5. What is the new in Angular v9?

1.Default Ivy

Ivy is the code name for angular's next-generation compilation and rendering pipeline. Angular 9's compiling and rendering engine is known as Ivy. Application build with IVY is more efficient. The new version of Angular use View Engine and View Engine produced a large size bundle. But with the help of Ivy this bundle has considerably reduced thereby helping Angular overcome its bundle issues.
3. Smaller bundle size

In the new feature, the buddle sizes are decrees by 30 to 40 percent based on the app size. When you compile any component using view engine in Angular JS, for example, home. component, you get two JavaScript files. The first one is example.component.js which consists of the compiled typescript code and example.component.ngfactory.js file which is a static representation of the template code. That is why here, mapping of these two files which consume a much longer build time. Angular team decided that the second file which is .ngfactory.js file be removed by just adding the template code within the JavaScript code itself. Ivy makes use of function calls rather than iterating over each element as in the View Engine.

4. Better Debugging

Debugging is one of the features of Ivy compiler and runtime. It is difficult to develop a program that is completely bug-free. Developers will not need to be debugged through the framework; they can do it on the component itself which helps you debug your code quickly.

5. Faster Compilation

Earlier, when you run ng build command the whole application gets recompiled. Because the Angular components are aware of all the dependencies of themselves.

For example, component contain *ngIf then the component would also know what this *ngIf in order to compile it. If you make changes to any of the dependencies then all component that contain this *ngIf needs to be recompiled. The team came up with an idea with locality principle which brought forth single file compilation.components that contain *ngIf don't need to know

dependencies. In this case, some component is recompiled, nothing else needed to recompile. Ivy calls the constructor of the *ngIf which knows the exact dependencies of it.

6. AOT and Ivy

AOT along with Ivy speed up the creation of an application to active AOT in your application. Open *angular.json* file and set aot : true.

```
“options ”: {
“outputPath”: “dist/project”,
“index”: “src/index.html”,
“main”: “src/main.ts”,
“polyfills”: “src/polyfills.ts”,
“tscongif”: “tsconfig.app.json”,
“aot”: true,
```

If you like to build your application efficiently, then you will have to run ngcc after every installation of third party packages. To do this, insert the below script in the *package.json* file.

```
“name”: “project”
“version”: “0.0.0”,
“scripts”: {
“postinstall”: “rgcc --properties es2015 browser module main --first-only  --create-ivy-entry-points”,
“ng”: “ng”,
```

The script is executed each time when you will install a new module.

7. What is AOT?

As mentioned earlier, the Angular AOT compiler compiles your HTML and Typescript into JavaScript before the browser downloads and executes it.

When you use ng serve or ng build then JIT compiler will come into action. To use Aot compiler the follows:

```
ng serve -aot                    //to build and serve
ng build -aot                    //to build
```

8. Adds improved type checking

The Angular compiler checks the expressions and bindings within the templates of the application and reports any type errors it finds. The bugs are caught and fixed early in the

development process. The Angular 9 currently has three modes of doing this. Apart from the default flag, the other flags that the angular supports are:

fullTemplateTypeCheck — By activating this flag, the compiler will check everything within your template (ngIf, ngFor, ng-template, etc)
strictTemplates — By activating this flag, the compiler will apply the strictest Type System rules for type checking.

6. What is the new in Angular v10?

1. Warnings on CommonJS Imports

A dependency packed with CommonJS in your projects can affect the speed of app loading and lead to slower functionality. However, with the Angular 10, developers have got warnings regarding using dependencies packed with CommonJS.

2. Optional Stricter Settings

Angular 10 provides a stricter project setup for making a new workspace using ng new.

ng new –strict

After allowing this flag, it starts with the new project using some new settings that enhances maintainability, allows the CLI for performing advanced optimizations on the app, and helps catch bugs properly beforehand. Permitting this flag helps app configuration as side-effect free to make sure more developed tree-shaking.

3. New Date Range Picker

Angular 10 has come with the all-new date range picker. To use it, you can utilize elements of both mat date range picker and mat date range input.

4. Enhanced Community Engagement

Angular features a large global community of developers who constantly provide value offerings for the angular projects throughout the spectrum. The organization along with the launch of Angular 10 also revealed its upcoming strategy of doing a huge investment for including the community to create the platform better.

Due to the constant efforts of including Angular community assistance in providing a better toolset and framework, problem counts could be reduced naturally. The organization will make big investments for more accurate working with the Angular development community.

5. Boost in ngcc Performance

You can see a boost in performance that is accomplished by lowering the entry point's size. Moreover, hiding of dependencies is implemented inside the entry point exhibit and is read from there as compared to being computed every time.

Earlier although an entry point didn't require processing, ngcc might analyze the entry point files for computing dependencies, which might take lots of time for large_node modules.

To boost performance, the computation of basePaths has been created lazily. What earlier was done when the finder was instantiated is currently just done in case required in TargetedEntryPointFinder.

5. Async Locking Timeout

Async locking timeout's configuration is still probable, which may include assistance in the ngcc.config.js file, adjusting the retry delays and retry attempts in the AsyncLocker.

An examination has been combined for monitoring for the timeout, using the ngcc.config.js file to lower the timeout and prevent the more maximized application of the test.

6. Removals and Corrections

The main Angular developer team has executed much deprecation and removed some things from this platform.

For example, FESM5 or ESM5 bundles have been taken out from the Angular Package Format and this lowered the file size by at least 119 MB in case you utilize package managers like npm or Yarn to install Angular.

Moreover, the assistance for some old browsers like Internet Explorer Mobile, Internet Explorer 9, 10 has also been taken out.

7.Compiler Update

A compiler interface has been added in the most recent Angular 10 to cover the actual ngtsc compiler. Angular 10 has also included name spans for method calls and property reads. Alongside this, ng-content selectors, Dependency data, and Angular language service have also been added to the metadata.

It also accepts the procreation of the accurate cost period in the ExpressionBinding of a micro syntax expression to ParsedProperty, which, as a result, might procreate the period to the template ASTs (both Ivy and VE).

8. TSLint v6, TSLib 2.9, and Typescript 3.9

Angular 10 boasts Typescript 3.9. Typescript language creates on JavaScript by including syntax for annotations and type declarations used by Typescript compiler to type-check the developers' code. As a result, this cleans readable JavaScript that runs on several runtimes.

Typescript helps save files with its great editing performance across editors. Typescript 3.9 helps the team in working on stability, polish and performance. Aside from the error-checking, the latest angular version strengthens things like editing experience, accelerating the compiler, rapid fixes, and completions.

9. Generic with ModuleWithProviders

The utilization of generic keywords has become compulsory with the *ModuleWithProviders* in Angular 10. This has been added for making the *ModuleWithProviders* patterns function with the rendering sequence and Ivy compiling.

7. What is the new in Angular v11?

1. Faster Builds

The first thing Angular has been consistently doing with new versions is the commitment to optimizing for speed. In this new version, Angular gets even faster than you know, from the dev to even the build cycle. This was possible through a few changes and updates on things like compilation, which now uses TypeScript 4.0 and faster processes like the ngcc update, now up to four times faster.

2. Angular ESLint Updates

So before this new version, Angular always implemented linting with TSLint by default, but TSLint is now deprecated by the creators, who recommend migrating to ES Lint completely.

With a lot of Angular community members' help together with James Henry, a third-party migration path was built with typescript-eslint, angular-eslint and tslint-to-eslint-config, and this has been tested to ensure a smooth transition for Angular devs. So, moving forward, the use of TSLint and even Codelyzer have been deprecated, meaning the default Angular implementation for linting will no longer be available but you can incorporate angular-eslint in a project and migrate from TSLint.

3. Internet Explorer Updates

This new version is also removing all support for old versions of Internet Explorer versions 9 and 10 and even the mobile version. The only IE version now still being supported is 11, and deprecated APIs were also removed too.

4. Webpack 5 Support

This new version ships with an opt-in experimental webpack support. That means that you can opt in to use version 5 of webpack in your project going forward. Angular plans to lead this path and allow for faster builds with really persistent caching of the disk and even smaller bundle sizes with the cjs tree-shaking. Do remember that this is still experimental, so do not

use it in production yet. You can take it for a spin by enabling it inside your new project by adding this line below to your *package.json* file:

```
"resolutions": {
    "webpack": "5.4.0"
                                                    }
```

JSON
You have to use yarn though, as npm does not work yet

5. Improved Logging and Reporting

This new version of Angular also ships with new changes in the way reporting is done on the builder phase during development. New changes have now been made on the CLI to make logs and even reports generally easy to read.

6. Updated Language Service Preview

The language service Angular used before now to provide tools that help make building with Angular fun was based on the view engine. Now that Ivy is mainstream, the team wants you to see a preview of how things will work with a better engine and renderer. This gives you the best Angular experience hands down and it will be able to infer generic types inside your templates just as a TypeScript compiler will do. Look at the image below for a good instance of inferring the iterable type of string.
7. Updated Hot Module Replacement (HMR) Support

Angular has had the support for hot module replacement for a bit now; however, using it had requirements that would involve config changes that made it not really great to include in new Angular projects. So the CLI had to be updated this new version to enable HMR from the jump as you run ng serve like this:
ng serve --hmr Angular
After your app runs, you will see the confirmation message saying that HMR is active. You should also remember it is only available for the development server. So as you build you will begin to see the latest changes to your components, templates and even your styles instantaneously update as you run the app without an actual page refresh at all. This is good—it reaches things like data users type inside input boxes in forms.

8. Automatic Font Inlining

More optimization updates, now the very first contentful paint of your app is set up with automatic inlining. This means that as you run ng serve, Angular will now download and inline fonts that are used or linked in your project so that everything loads up even faster. This update comes right out of the box with Angular 11, so update your version

9. Component Test Harnesses

With Angular 9 there was this component test harness that provided a readable and robust API base for testing Angular material components with the supported API at test. With this new

version 11, we now have harnesses for all components, so even more test suites can now be built by developers.

This comes with a lot of updates, performance improvements and even new APIs. Now the parallel function makes dealing with async actions inside tests easier because you can run multiple async interactions with your components in parallel. A function like the manual change detection will now give you access to even better control of detection by just disabling auto change detections inside your unit tests.

8. What is the new in Angular v12?

1. Ivy Everywhere: Angular v12
   The View Engine has finally been deprecated in Angular version 12. The community has been working over ongoing deliveries towards the objective of combining the Angular ecosystem on Ivy. They call this methodology "Ivy Everywhere"

2. Migrating from legacy i18n message IDs
   As of now, there are different legacy message-id designs being utilized in the i18n framework. These legacy message-ids are delicate as issues can emerge dependent on whitespace and the organizing formats and ICU expressions. To tackle this issue the Angular team is relocating away from them. The new standard message-id design is considerably more tough and natural. This configuration will diminish the superfluous translation invalidation and related retranslation cost in applications where translations don't coordinate due to whitespace changes for instance. Since Angular version 11, new tasks are naturally designed to utilize the new message ids and they presently have the tooling to move existing ventures with existing translations.

3. Protractor: planning for future
   The team has been working with the community to decide the eventual fate of the Protractor. They are right now exploring the feedback shared in the RFC and sorting out the best future for Protractor. This is one of the latest features of Angular 12. The team have chosen to exclude it in new tasks and, all things considered, furnish alternatives with famous 3rd party solutions in the Angular CLI. The team is presently working with Cypress, WebdriverIO, and TestCafe to assist angularjs development company with receiving elective solutions. More data to come as this develops.

4. Nullish Coalescing
   The nullish coalescing operator (??) has been assisting engineers with composing cleaner code in TypeScript classes for some time now. One of the main highlights of this feature is that you can bring the force of nullish coalescing to Angular templates in v12! Presently, in templates, engineers can utilize the new syntax structure to improve the complex conditionals. For instance:

   {{age !== null && age !== undefined ? age : calculateAge() }}
   Becomes:
   {{ age ?? calculateAge() }}

5. Improvements in styling
   In Angular v12, Components will have support for inline Sass in the styles field of the @Component decorator. Already, Sass was just accessible in outside resources because of the Angular compiler. You can empower this component in your current applications simply by adding "inlineStyleLanguage": "scss" to angular.json. Else, it will be accessible to new tasks utilizing SCSS.
6. Angular CDK and Angular Material have adopted Sass' new module framework internally. If your application utilizes Angular CDK or Angular Material, you'll need to ensure you've changed from node-sass to the sass npm package. The node-sass package is unmaintained and no longer stays aware of new feature additions to the Sass language.
7. While updating your application to the angular latest version, It will consequently change to the new Sass API by refreshing your application with ng update. This order will refactor any Sass @import expressions for Angular CDK and Angular Material code over to the new @use API, as both Angular CDK and Angular Material uncover another Sass API surface intended for utilization with the new @use language structure. Here's an illustration of the before and after.

8. Deprecating support for IE11
   Angular is an evergreen platform, implying that it keeps up with the advancing web ecosystem. Eliminating support for legacy browsers permits us to focus on giving modern solutions and better help to engineers and clients. The team has also included a new deprecation warning message as another Angular 12 feature — and eliminate support for IE11 in Angular v13.

9. Support from the Community
   The Angular team has been working diligently for improving the Angular experience for everybody by adding to the structure. They are continually attempting to improve the Angular learning experience for developers as well. As a part of Angular 12 new features, they have rolled out some significant improvements to their documentation. They have also updated the angular.io contributor's guide that will help individuals hoping to improve the docs.

10. Strict by default
    Strict mode improves maintainability and assists you with catching the bugs early in the process. Moreover, strict mode applications are simpler to statically examine and can help the ng update command refactor code more securely and accurately when you are updating to the latest versions of Angular.
    Use the following command to create a new workspace and application:
    ng new [project-name]

    Run the following command when you want to create a new app within an existing non-strict workspace in the strict mode:
    ng generate application [project-name] –strict

11. HTTP improvements
    Angular 12 features a number of upgrades around its HTTP support. Let's have a look:
    Metadata for requests and interceptors

To start with, the HttpClient would now be able to be utilized to store and recover custom metadata for requests. This is especially helpful for HTTP interceptors. This capacity can be utilized through the new HttpContext.
Presently, the distinctive HTTP strategies given by HttpClient incorporate another specific circumstance: HttpContext alternative, which we can use to pass in a map.
appendAll on HttpParams
The HttpParams class presently has another appendAll technique, which can be utilized to effortlessly add a bunch of boundaries without a moment's delay: appendAll(params: {[param: string]: string|string[]}): HttpParams
HttpStatusCode
Angular 12 features its own list of intelligible names for HTTP status codes, in the form of a const enum. Because of this new element, we can now utilize HttpStatusCode instead of introducing our own solution to have human-readable names

12. ng API improvements
    The ng troubleshooting API that we can use from the program dev tool is an improved feature of Angular 12. There are a few functionalities that have been implemented namely getDirectiveMetadata and esetProfiler to debug APIs for structural inspection of applications. The getDirectiveMetadata can be utilized to recover data about parts and directives. The esetProfiler can be used to trace template creation durations, lifecycle hooks processing, and template updates. The API can give an insight into the working of the applications at runtime.

13. New Dev Tools
    A couple of days after the release of Angular 12, the Angular group has reported the accessibility of Angular Dev Tools for Google Chrome. The embedded profiler can preview and record the change detection events which can be monitored as to which detection cycle and components took the longest time. Earlier the Angular community had semiofficial Dev Tools which were not compatible with Ivy. So this is a win-win situation for all.

14. Angular Universal
    One of the major improvements of the Angular 12 feature is Inline critical CSS that is by default in the nguniversal/common. The Angular universal now supports proxy configuration in ssr-dev-server builder. It has also updated schematics to use defaultConfiguration. This version supports an SSR engine called Clover along with a new engine which seems promising. The new engine aims to simplify things to generate application shells without an extra build and remove the requirement for multiple builds for SSR/prerender. The version also supports TLS for the dev-server.

15. Typescript 4.2
    One of the major updates in the feature of Angular 12 is the support of Typescript 4.2. The stable version was released on the 23rd of February. There are some exciting features and breaking changes in this version. Let’s find out the features of Typescript 4.2 and how this version will impact developers and Angular 12.

16. Changes in Tuple Types

Tuples are helpful datatypes in programming whenever developers need to return different outcomes from a function. As they're constant, tuples can use as keys for dictionary-type word reference. One of the vital highlights of TypeScript is that tuples can have discretionary and optional components and rest components, which is fantastic for readability and tooling.

17. Abstract Construct Signatures
    Abstraction is one of the major features of TyepScript 4.2. TypeScript permits designers to utilize the theoretical modifier to check a class as abstract. To make occasions to extract properties and practices of abstract classes, engineers need to follow an essential OOP idea, which is to broaden the abstract class utilizing a subclass and startup objects. Mixins are one of the bleeding edge techniques for TypeScript, which is additionally identified with a similar subject.
    Initially, while using mixins with abstract classes you could find some bugs. However, after version 4.2 developers can use the abstract modifier on constructor signatures.

18. Improvements for in operator
    It is a mistake to utilize an essential variable on the correct side of an in-operator, an overall guideline in programming. Nonetheless, until version 4.2, it was not there in TypeScript. However, with the new update, TypeScript is exacting to rules. So, the in-operator won't go to permit crude information types on the correct side.
    “test var” in 31 // will immediately lead to an error.

19. Improved Type Alias
    In past forms, TypeScript didn't resolve types effectively. In most scenarios, TypeScript will return the right types however not the substantial valid alias, prompting an absence of readability and repeat a similar code over and again. Currently, the re-aliasing of such kinds reliably receives and shows the new type alias

20. Improvements to the compile process
    When creating complex projects, at times it is difficult and tedious while looking for TypeScript file definitions. Mostly, it is a process of experimentation. However, with Typescript version 4.2, it is feasible to get names of documents that TypeScript finds as a piece of the project. Also, Typescript discovers explanations behind the documents to be important for the compilation process.

21. Webpack 5.37 support
    Angular 12 features the production-ready experimental Webpack 5 support that was introduced in Angular 11. Webpack is the vital piece of the Angular CLI puzzle, and it plays a major part in bundle size, constructs execution, and so on. Webpack 5 is a significant delivery which is as it should be. It incorporates various breaking changes and features. The webpack 5 helps in Improving the build performance, long-term caching, compatibility with the Web platform, bundle size with better code generation.

22. Breaking changes in Angular 12 Animation:
    In Angular version 12 features the DOM components that are presently accurately eliminated when the root view is taken out. If you are utilizing SSR and utilize the

application's HTML for delivery, you will be required to guarantee that you save the HTML to a variable prior to destroying the application.
It is additionally conceivable that tests could be unintentionally depending on the old conduct by attempting to discover a component that was not taken out in a past test. However, if that is the situation the failing tests ought to be updated to guarantee they have proper setup code which instates components they depend on.

Common:
Methods of the PlatformLocation class, specifically onPopState as well as onHashChange,used to return void. Presently those techniques return works that can be called to eliminate event handlers.
One of the main Angular 12 improvements is that the strategies for the HttpParams class currently acknowledged string | number | boolean rather than string for the value of a parameter. If you expanded this class in your application, you'll need to update the signatures of your strategies to mirror these changes.

Core:
Angular no longer helps in maintaining Node v10. Previously the ng.getDirectives work tossed an error when in case a given DOM node had no Angular setting related to it. This conduct was conflicting with other troubleshooting utilities under ng namespace, which took care of the present circumstance without raising an exemption. As a major change in Angular 12, the ng.getDirectives work for such DOM nodes would bring about an empty array back from that function.
The type of the APP_INITIALIZER token has been changed to more precisely mirror the types of return values that are taken care of by Angular. Before this, each initializer callback was composed to return any, this is currently Promise <unknown> | Observable <unknown> | void.
In the unlikely event that your application utilizes the Injector.get or TestBed.inject API to infuse the APP_INITIALIZER token, you may have to refresh the code to represent the stricter type.

Compiler - CLI:
One of the most interesting features of Angular 10 was the use of the new IVY extractor to run ng x i18n in one of your applications. One of the new features of Angular 12 is that it no longer generates legacy i18n message-ids through linked libraries. For downstream applications providing translations for these messages, the localize-migrate command-line tool will help migrate their message ids.

Form: Previously ‘min’ and ‘max’ attributes characterized on the <input type=" number"> were disregarded by the Forms module. Currently, the presence of these characteristics would trigger min/max validation logic (if formControl, formControlName or on the other hand ngModel orders are additionally present on a given input) and comparing form control status would mirror that.

Platform browser:
XhrFactory has been moved from @angular/normal/http to @angular/normal.
/// Before
import {XhrFactory} from '@angular/common/http';

```
/// After
import {XhrFactory} from '@angular/common';
```

Router:
Strict invalid checks will give an account of part conceivably being invalid. The type of the RouterLinkActive.routerLinkActiveOptions input was extended to permit all the more adjusted control. Code that recently read this property may be updated to represent the new type.

7. What is the new in Angular v13?

1. Typescript Update

Previously 4.4.2 version of Typescript was supported, now Typescript 4.4 support has been added in Angular 13. Typescript 4.4.2 and older than that is no longer supported. This update can be seen in the package.json files.

Control Flow Analysis, performance boost, new flags, and better IntelliSense are some of the key highlights Typescript 4.4 version.

2. Removal of View Engine

This is one of the most notable Angular 13 features leading to faster compilation and increased productivity in the framework. The view-engine angular 13 feature has been completely removed to reduce the complexity of the Angular 13 codebase.

Angular 13 has completely shifted to Ivy which makes it easier for developers to improvise the dynamic components easily.

To ensure that this transition goes well, the framework has converted all internal tools to Ivy beforehand.

3. NodeJS Support

Versions older than v12.20.0 are no longer supported by the Angular framework. Web developers might face certain issues while installing different packages if working with the older versions.

16.14.2 is the current stable version of NodeJs. For ensuring seamless deployment of your project, it is recommended to install the latest versions of NodeJs.

4. Angular CLI Improvements

Angular CLI stands out to be one of the most crucial aspects of the overall angular architecture. By reducing complexities on a broad scale, Angular CLI helps standardize the process of handling the challenges of the present web development ecosystem.

With the release of Angular 13, significant changes to the Angular CLI are introduced for performance improvement.

- Built-in support of persistent build cache
- Easy enabling and disabling options for the build cache by the angular.json file
- Up to 68% increase in speed of build-cache leading to faster deployment activities.

These are the notable Angular CLI improvements that are introduced in the new Angular 13 features and updates.

5. Validation Improvements

They have now introduced the new type 'Form Control Status' in angular forms. The main aim of this is to have a better check on form controls.

Form control status includes all possible values like ‘Valid’, ‘Invalid’, ‘Pending’, and ‘Disabled’.

For example, instead of string, the type of AbstractControl.status is now FormControlStatus.

These are some of the changes observed regarding the Updating Validators:

abstractControl.setValidators

abstractControl.addValidators

abstractControl.removeValidators

abstractControl.hasValidtors

It is now easier to enable and disable validations like min, max, email, etc.

6. Removal of IE 11 Support

This stands out to be one of the significant Angular 13 features. Angular 13 no longer supports IE11. CSS code paths, build passes, polyfills, special JS, and other parameters that were previously required for IE 11 have now been completely dropped off.

As a result, Angular has grown faster, and it is now easier for Angular to use new browser features like CSS variables and web animations using native web APIs.

Developers need to focus more on Web APIs, web notifications, WebRTC, WebGL, etc.,

7. Changes in the Angular Package Format (APF)

The Angular Package Format (APF) describes how Angular Framework packages and View Engine information should be formatted and assembled. Some substantial improvements are seen in the new edition of APF.

- Older output formats that include View Engine-specific Metadata have been dropped off.
- Standardization of modern JS formats such as ES2020.
- Libraries built with the latest version of the APF no longer require the use of ngcc.

8. Framework Modifications and Dependency Updates

RxJS 7.4 is now available as the version for apps created with ng-new. Existing apps using RxJS v6.x need to be manually updated it using the npm install rxjs@7.4

8. What is the new in Angular v14?

1. Angular gets simplified with Standalone Components

This is one of the major advancements that can be seen with the release of Angular 14. With the addition of standalone components, the process of writing Angular Apps becomes much simpler and easier than before.

Standalone components eliminate the need of using NgModules, and we can now get the developer preview of a new way of writing components, one without the NgModule i.e Standalone Components.

For now, this feature is available in the developer preview, meaning that it might change later until it becomes fully stable. This change could majorly alter the process of writing Angular Apps in the future.

2. Typed Forms

When it comes to Angular, there are two distinctive approaches for handling the forms. They can be either created by the template-driven approach or by using the reactive approach.

This newly introduced feature of Typed Forms is only applicable to reactive forms. The values inside form controls, groups, and arrays are type safe. It improves the overall “ type” safety of the applications developed using Angular.

The updated schematics enable progressive migration to Typed forms, allowing you to gradually add typing to your existing forms.

3. Streamlined Page Title accessibility

While developing apps your page title distinctively represents the content of your page. In the previous release of Angular 13, the process of adding title was streamlined with the new *Route.title* property in the Angular Router.

Now with Angular 14, there are no more additional imports required when adding a title to your page.

4. New primitives in the Angular CDK

The Component Dev Kit (CDK) from Angular provides a comprehensive set of tools for creating Angular components. The CDK Menu and Dialog have now been promoted to stable version in Angular 14!

The addition of new CDK primitives in Angular 14 allows for the creation of more accessible custom components.

Some additional Angular 14 features

1. Angular DevTools is now available offline

The Angular DevTools debugging extension can now be used in offline mode as well. Firefox users can find the extension under Mozilla's Add-ons for Firefox users.

2. Angular CLI enhancements

Angular 14 has got an amazing feature of autocomplete! Typos are bound to often while typing your code, resulting in command-line errors. To fix this, *ng completion* in version 14 now includes real-time type-ahead autocompletion!

Detailed analytics information using the *ng analytics* and improved ways to control the cache information using the *ng cache* are some of the additional features.

3. Optional injectors

When building an embedded view using the Angular 14 version, you may now specify an optional injector
through *ViewContainerRef.createEmbeddedView* and *TemplateRef.createEmbeddedView.*

4. Built-in improvements

Angular 14 adds support for TypeScript 4.7 and now targets ES2020 by default, allowing the CLI to deploy smaller code without downgrading.

Another noteworthy Angular 14 feature is that you can now link to protected component members directly from your templates. This offers more control over the public API surface of the reusable components.

5. Extended developer diagnostics

This new Angular 14 feature provides an extendable framework that enables better insights into your templates and provides suggestions for possible improvements in them.

9. What is the new in Angular v15?