Javascript

Friday, January 10, 2025 5:42 AM

Command	Returns
window.navigator.userAgent	User agent string of current browser. Eg: "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:124.0) Gecko/20100101 Firefox/124.0" for mozilla.
indexOf	Finds the index of the string passed in the actual string. Returns -1 if not found. Eg: window.navigator.userAgent.indexOf('MSIE ')

JSON

Friday, January 10, 2025 11:42 AM

JSON.stringify - deserialize
JSON.parse - serialize

```
// Create item:
let myObj = { name: 'Nixon', profession: 'Developer' };
localStorage.setItem(key, JSON.stringify(myObj));
// Read item:
let item = JSON.parse(localStorage.getItem(key));
```

Session_Local_Storage

Friday, January 10, 2025 11:31 AM

Documentation:
Element: input event - Web APIs | MDN
To play online:
Playground | MDN

The window object represents an open window in a browser.
Session and local storage are part of window object.

Session Storage keeps a separate storage area for every given origin.
Available for the duration of the page session.
Session Storage is flushed when the tab or window is closed.
Generally browsers provide 10MB of storage space.

sessionStorage is lost the moment the browser is closed, while localStorage remain till the browser cache is cleared.

To store a session variable

```
sessionStorage/localStorage.setItem('name', 'Rana Hasnain');
sessionStorage/localStorage.getItem('name');
sessionStorage/localStorage.removeItem('location');
sessionStorage/localStorage.clear();
```

Angular Data Bindings

Thursday, January 9, 2025 9:57 AM

Data Binding	Direction	From	To	Syntax	Example	Remarks
String Interpolation	One way	Component	DOM	{{value}}		
Property Binding	One way	Component	DOM	[property]="value"	<app-hero-detail [hero]="selectedHero"> </app-hero-detail>	selectedHero - parent component variable hero - property of child component
Event Binding	One way	DOM	Component	(event)=""handler	<li (click)="selectHero(hero)">	selectHero- component's method. Called when user clicks on hero name.
Model Binding	Two way	Componet/ DOM	DOM/Component	[(ng-model)]="property"	<input [(ngModel)]="hero.name">	Used in template-driven forms. Combines property and event binding in a single notation. Property binding - data value flows from component to input box. Event binding - user's changes flow back to the component.

Angular Directives

Thursday, January 9, 2025 10:30 AM

Directives are classes that add additional behavior to elements.
There are 3 types of Directives

1. Components
2. Attribute Directives
3. Structural Directives

Components

Thursday, January 9, 2025 10:36 AM

Components are fundamental building blocks.
It encapsulates part of user interface - including styles, behavior
Syntax: To generate component
To create and add component to app module(default)
ng g component component-name
ng g component folder-name/component-name
To create and add component to custom module(user defined module)
ng g component folder-name/component-name --module/--m module-name

Modules

Thursday, January 9, 2025 11:32 AM

Imports	Declarations	Providers
Exported declarations of other modules to be available in current module	Used to declare components, directives, pipes that belongs to current module.	Used to make services and values known to DI.
Used to import supporting modules.	Used to make directives from the current module available to other directives in current module.	Used to inject services required by comp/ dir/ pipes in our module.

Modules:
Groups related components, directives, pipes, and services together.
The Angular module must implement a specific feature. The Components, directives, services that implement such a feature, will become part of that Module. It can include pipes too.
Uses of Modules:
Helps in modularity - structure and organize application.
Lazy loading
Maintainability
Syntax:
ng g m module-name

Under app-> creates a new folder with module name
And a file inside -> module-name.module.ts

Providers:
Useclass
Usevalue

Interceptors

Thursday, January 9, 2025 2:37 PM

HttpClient supports a form of middleware - interceptor.
Interceptors are functions which run for each request.
We can have interceptor chain - where each interceptor process the request or response before forwarding it to next interceptor.

Services

Thursday, January 9, 2025 2:37 PM

Dependency Injection

Thursday, January 9, 2025 2:47 PM

Dependency provider
Dependency consumer
Injector facilitates the communication between provider and consumer.
When a dependency is requested, injector checks the registry for instance already exists.
If not exists, a new instance is created and stored in the registry.

Provider:
Step1: Preparing the Provider:
@Injectable decorator - denotes the class can be injected.
Step2: Make it available in the DI by registering at any of the places

- At the component level, within providers - here a new instance of service will be created for each new instance of that component.
- Singleton instance - At the application root level. A single instance is created and shared to any class that requires the instance. Makes visible of this service throughout the application by providing root.
  @Injectable({providedIn: 'root'})
- Module level - the same instance is shared to all components, directives and pipes declared within this module. Registered using providers array.

Consumer:
Option 1: via class constructor.
constructor(private service: HeroService){}
Option 2: via inject method inside class.
private service = inject(HeroService)

HttpClient

Thursday, January 9, 2025 5:15 PM

HttpClient from @angular/common/http

Step 1: In app.module.ts
Import HttpClientModule
Add HttpClientModule in imports

Step 2: in the service class

- Inject the dependency via class constructor
- Import rxjs Observable

Rxjs Operators -
Operators enable transformations of observable values.
Operator is a function takes observable as input and manipulates the input data and return a new observable of the transformed values.

map -> transforms the data
filter -> selective picks based on condition

CLI Commands

Friday, January 10, 2025 4:15 AM

Command	Remark
ng serve --open	To start the portal and open it in browser
npm install --save-dev	To install any package as developer dependencies, these are needed only at development stage and not in production. Example: npm install --save-dev @angular-devkit/build-angular Or in short Example: npm install -D package-name --save-dev - adds the package under "devDependencies" DevDependencies are ignore in either of these cases: npm install --production=true(default) - This will ignore dev dependencies package in prod npm install with environment variable 'NODE_ENV' set as 'production'

Package.json

Friday, January 10, 2025 4:44 AM

~version “Approximately equivalent to version”
- supports backward compatible without incrementing the minor version.
Only accepts patch version.
without incrementing the minor version. ~1.2.3 will use releases from 1.2.3 to < 1.3.0

^version “Compatible with version”
- supports backward compatible without incrementing the major version.
Accepts both patch and minor version.
without incrementing the major version. ^1.2.3 will use releases from 1.2.3 to < 2.0.0

Sections:
Dependencies - packages listed under here are installed in production too.
DevDependencies - packages installed here are not installed in production and available only in Development.
Production/ NonProduction - set via environment variable 'NODE_ENV' set as 'production'

To install any package as developer dependencies, these are needed only at development stage and not in production.
Example: npm install --save-dev @angular-devkit/build-angular
Or in short
Example: npm install -D package-name

--save-dev - adds the package in **package.json** under "devDependencies"

DevDependencies are ignore in either of these cases:
npm install --production=true(default) - This will ignore dev dependencies package in prod
npm install with environment variable 'NODE_ENV' set as 'production'

Main.ts

Friday, January 10, 2025 5:35 AM

The first file will be executed when the angular runs.
From here the application is bootstrapped.
App.Module will be executed.

```
if (environment.production) {
  enableProdMode();
}
```

enableProdMode():
From @angular/core
When production mode is enabled, Angular will enhance the application performance by disabling verifies that a change detection pass does not result in additional changes to any bindings (also known as unidirectional data flow).

Reference link:
https://lukaonik.medium.com/what-is-the-difference-between-production-and-development-mode-in-angular-3eed82b9cf73

Route guard

Friday, January 10, 2025 11:02 AM

Generate route guard

Step 1:
ng generate guard auth

Route Guards:
To prevent user from accessing certain parts of the application under certain circumstances.
Use cases are CanActivate, CanDeactivate, Resolve, CanLoad, CanActivateChild, CanMatch

CanActivate Route guard:
This runs before we navigate to a route allowing us to cancel the navigation.
This guard decides whether a route can be activated.

Interface CanActivate - @angular/router
Contains one method canActivate
2 parameters -route: ActivatedRouteSnapshot, state: RouterStateSnapshot
Returns: Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree

ActivatedRouteSnapshot, RouterStateSnapshot - route/ query parameter
UrlTree - represents the parsed url

A route can have more than one canActivate guard
If all guards returns true, navigation to the route will continue.
If any one of the guard returns false, navigation will be cancelled.
If any one of the guard returns UrlTree, current navigation will be cancelled and a new navigation will be kicked off to the urlTree returned from the guard.

Update the route definition as
{path:'product',component:'ProductComponent',canActivate:[AuthGuardService]}

```
 {
     path: '', component: HomePageComponent, canActivate: [AuthGuard],
     data: { expectedRole: [RoleNames.HM, RoleNames.CP, RoleNames.MS, RoleNames.MU] }
   }

{
    path: 'admin/user', component: UserManagementComponent, canActivate: [AdminauthGuard],
    data: { expectedRole: [RoleNames.MU] }
  }
```

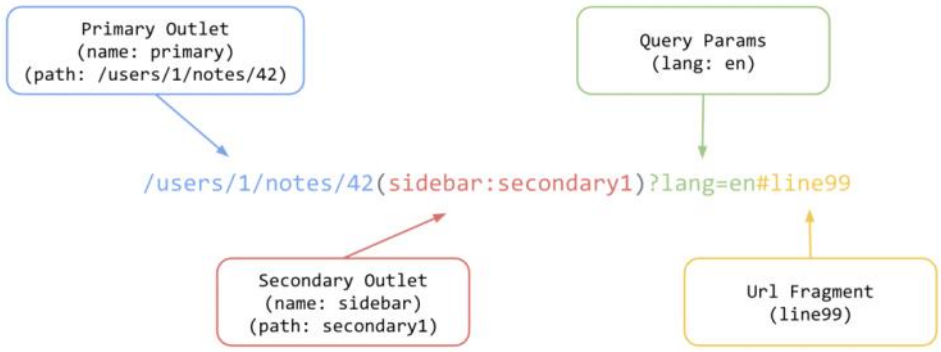
parseUrl - parses a string into a UrlTree
Available in Router from @angular/router
Inject router in constructor

This returns the UrlTree
return this.router.parseUrl('norolesfound');

```
const url = '/users/1/notes/42(sidebar:secondary1)?lang=en#line99';
const tree = this.router.parseUrl(url); // '/users/1/notes/42(sidebar:secondary1)?lang=en#line99'
const fragment = tree.fragment; // line99
const queryParams = tree.queryParams; // lang=en
const primary: UrlSegmentGroup = tree.root.children[PRIMARY_OUTLET]; // gets the UrlSegmentGroup
for the primary router outlet
const sidebar: UrlSegmentGroup = tree.root.children['sidebar']; // gets the UrlSegmentGroup for the
secondary router outlet (sidebar)
const primarySegments: UrlSegment[] = primary.segments; // returns all UrlSegments for the primary
outlet. ['users','1','notes','42']
const sidebarSegments: UrlSegment[] = sidebar.segments; // returns all UrlSegments for the secondary
outlet. ['secondary1']
```

Routing

Friday, January 10, 2025 4:35 PM




```
const url = '/users/1/notes/42(sidebar:secondary1)?lang=en#line99';
const tree = this.router.parseUrl(url); // '/users/1/notes/42(sidebar:secondary1)?lang=en#line99'
const fragment = tree.fragment; // line99
const queryParams = tree.queryParams; // lang=en
const primary: UrlSegmentGroup = tree.root.children[PRIMARY_OUTLET]; // gets the UrlSegmentGroup for the primary router outlet
const sidebar: UrlSegmentGroup = tree.root.children['sidebar']; // gets the UrlSegmentGroup for the secondary router outlet (sidebar)
const primarySegments: UrlSegment[] = primary.segments; // returns all UrlSegments for the primary outlet. ['users','1','notes','42']
const sidebarSegments: UrlSegment[] = sidebar.segments; // returns all UrlSegments for the secondary outlet. ['secondary1']
```

Primary router
Router outlet without name is primary router.
This allows components to display dynamically inside a root component.

Router State -
This describes which components to display for a given url
Is of type Route.
It is defined by importing RouterModule, use the forRoot method by passing router state config object.

routerModule.forRoot([{},{},{},..])

Or

```
const routes: Route[] = [
{ path: 'home', component: HomeComponent },
  { path: 'notes',
    children: [
      { path: '', component: NotesComponent },
      { path: ':id', component: NoteComponent }
    ]
  }
```

];
routerModule.forRoot(routes);

forChild vs forRoot
return modules containing all of the router directives and route configurations.
There can be only one forRoot at root app module
Feature modules(lazy loading modules) should use forChild

<router-outlet></router-outlet>
This router-outlet has no name. This router outlet is called [PRIMARY_OUTLET]
The router will place the routed components just after the router-outlet, within an <ng-component> element.
This happens automatically, so you don't actually type the <ng-component> element

Secondary Router

Consider you have 2 components to be loaded in the root component based on 2 corresponding urls

Secondary router is created with the name router
<router-outlet name="sidebar"></router-outlet>
<ng-component>Chat components go here</ng-component>

While defining router state, this name should be specified

```
const ROUTES = [
  { path: 'home', component: HomeComponent },
  { path: 'chat', component: ChatComponent, outlet: 'sidebar' }
]
```

Primary Outlets - localhost:4200/home - here only primary outlet is active
Secondary outlets are enclosed within brackets - localhost:4200/home(sidebar:chat) - here both primary and secondary outlet is active. Note here - changing the primary or secondary portion of a url doesnot affect the other outlet. They are routed individually.

Example:
/users/1/notes/42(sidebar:secondary1)

This loads the root component with 2 outlets.
1 for NoteComponent
And the other for secondary1Component

Here changing from secondary1 to secondary2 doesnot affect the primary outlet url /users/1/notes/42

For more details:
3 Pillars of Angular Routing - Router States and URL Matching
Routing in Angular - Secondary Outlets Primer
3 Pillars of Angular Routing - Angular Router Series Introduction

Options of forRoot:
initialNavigation: true/ false
False - the initial navigation is not performed. The location listener is set up before the root component gets created.
True - the initial navigation starts before the root component is created. The bootstrap is blocked until the initial navigation is complete. Use this when Server side rendering is enabled.

Client Side Routing

Friday, January 10, 2025 5:37 PM

Angular Routing Strategy:
PathlocationStrategy - HTML5 routing. Not supported by old browsers. Eg: http://localhost:4200/#/product
HashlocationStrategy - Hash style routing. Eg: http://localhost:4200/product

Angular is a SPA
Hence we cannot

- Refresh the page
- Goto particular view by typing url
- Share the url with someone
- Using browser Back button to move to prev page
- Support for SEO

Hence Client side routing is used to overcome the above issues.
The Client-side routing simply mimics server-side routing by running the process in the browser. It changes the URL in the browser address bar and updates the browser history, without actually sending the request to the server.

Hash Style Routing:
Uses anchor tag technique as like domain-name/#/navigation-path
To have hash location strategy in place, pass useHash as true in the options of forRoot method of router.
RouterModule.forRoot(routes, { useHash: true, initialNavigation: false })
useHash makes use of # in the url.
Supported by all browsers.
Will not support server side rendering.

In the below urls, all the 3 paths only www.example.com url is sent to the server.
www.example.com/
www.example.com/#/about
www.example.com/#/contact

PathLocationStrategy:
Is default strategy in Angular.
To configure this strategy, add <base href="/"> in the head section of the index.html of our application.

Other Options of forRoot:
initialNavigation: true/ false
False - the initial navigation is not performed. The location listener is set up before the root component gets created.
True - the initial navigation starts before the root component is created. The bootstrap is blocked until the initial navigation is complete. Use this when Server side rendering is enabled.

Observables

Friday, January 10, 2025 7:09 PM

Observable is reacting extension of javascript library RxJS
To manage asynchronous data - to handle streams of data.
An observable is a data stream - that emit multiple values over time.
Subscribe can listen to these emitted values.

httpClient:
Is a build-in service class available in angular/common/http package.
Returns observable<returndatatype> type, where returndatatype can be any type including array.

Observable operations:
Returns observable type.
map -> transforms each data and generates a new observable.
Filter -> based on condition data fetched and returned as new observable.

Post - takes 3 parameters
API Endpoint URL
Body - the data to submit
Options - optional
Headers - defines httpheaders for http POST request
Observe - Defines whether we want complete response or body only or events only. Response/ body(default)/ events
Params - Defines parameters in URL.
reportProgress - Defines whether we want progress report or not.
responseType - Defines response type such as arraybuffer, blob, json(default) and text.
withCredentials - a Defines whether we want to pass credentials or not.

Promise vs Observables

Sunday, January 12, 2025 6:52 AM

Remarks	Promise	Observable
Helps for	Asynchronous functionality	Asynchronous functionality
Handling Events	One at a time	A sequence of aync events over a period of time. Use SetTimeout for repeated fetch of data
Emit value	Emit a single value at a time	Emits multiple values over a period of time
When it is Execution	Are lazy: not executed until it is subscribed using suscribe() method.	Are not lazy, immediately executed after execution
Is Cacellation allowed?	Are not cancellable.	Can be cancelled with unsubscribe() method of cancellable subscriber. This stops the listener from receiving further values.
Operations	Don't provide any operations.	Provide the map, foreach, filter, reduce, retry, retrywhen.
Error handling	Push errors to child promises	Deliver errors to subscribers. Can handle error using catchError and decide whether to recover, continue or retry. It can also be completed successfully after an error is handled.

Let's assume service updates the data for every 1000ms i.e., 1 second
this.obs = new Observable((observer)=>observer.next(this.dataService.getData(true));
Const intervalId = setInterval(()=>{observer.next(this.dataService.getData(false));},1000});
Note:
clearInterval(intervalId) - to stop the timer

toPromise()

Sunday, January 12, 2025 7:24 AM

toPromise() lives on prototype of observable
It is a util method that is used to convert an observable into promise.
Inside this method - we subscribe to observable and resolve the promise with the last emitted value, when the observable completes.

toPromise() is deprecated
Instead use lastValueFrom()
There is an another function firstValueFrom which helps to get the first emitted value and doesn't want to listen for upcoming streams.