**What is AOT Compilation? - Pros and Cons of Ahead-of-Time!**

AOT compilation stands for “Ahead of Time compilation” and it are used to compiles the angular components and templates to native JavaScript and HTML during the build time instead of run-time.

The compiled HTML and JavaScript are deployed to the web server so that the compilation and render time can be saved by the browser. It is the big advantage to improve the performance of applications.

**Advantages of AOT -**

1.         **Faster download**: - The Angular app is already compiled so it is faster.

2.         **Faster Rendering**: - If the app is not AOT compiled and the compilation process happens in the browser once the application is fully loaded. This has a wait time for all necessary components to be downloaded and then the time taken by the compiler to compile the app. With AOT compilation, this is optimized.

3.         **Lesser Http Requests**: - It is supporting to the lazy loading. Actually, lazy loading is great concepts for sending HTTP request to the server. It is minimise the multiple requests for each associated html and css, there is a separate request goes to the server.

4.         **Detect error at build time**: - In Angular 2, the compilation happens beforehand and most of the errors can be detected at the compile time and this process providing us a better application’s stability.

**Disadvantages of AOT -**

1.         AOT only works only with HTML and CSS and not for other file types. If required other file types that time we will need to follow the previous build step.

2.         We need to maintain AOT version of bootstrap file.

3.         We need to clean-up step before compiling.

**What are Decorators?**

Decorators are functions that adds metadata to class members and functions. It was proposed in ES2016 and implemented in Typescript.

**Explain $event in Angular5?**

In Angular5 $event is a reserved keyword that represents the data emitted by an event (event data).

It is commonly used as a parameter for event based methods.

**What is the difference between observable and promises?**

1. Observable is a more powerful way of handling HTTP asynchronous requests. Whereas, A promise handles a single event when an asynchronous operation completes or fails.
2. An observable is like a stream which allows passing zero or more events where the callback is called for each event. Whereas, A promise eventually calls the success or failed callback even when you don’t need the notification or the result it provides anymore.
3. Observable works with multiple values for a particular time. Whereas, Promises works with and even returns a single value at a time.
4. Observables can be canceled. Whereas, Promises cannot be canceled.
5. Observable supports map, filter, reduce and similar operators. Whereas, Promises have more readable codes with try/catch and async/await.
6. In observable, one operator ‘retry’ can be used to retry whenever needed. Whereas, Promises cannot be retried. A promise should have access to the original function that returned the promise in order to have a retry capability.

**What is lazy loading and How to enable lazy loading in angular 2?**

**Lazy Loading** - Lazy loading enables us to load only the module user is interacting and keep the rest to be loaded at run-time on demand.

Lazy loading speeds up the application initial load time by splitting the code into multiple bundles and loading them on demand.

1.         Each and every Angular2 application must have one main module that is called “AppModule” and your code should be splitted into various child modules based on your applications.

2.         We do not require to import or declare lazily loading module in root module.

3.         Add the route to top level routing and takes routes array and configures the router.

4.         Import module specific routing in the child module.

**How would you Optimize the Angular 2 Application for Better Performance?**

The optimizations are depends on the size of applications, type and other factors but normally we consider following optimizing points i.e.

1.         Consider AOT compilation.

2.         Consider lazy loading instead of fully bundled app if the app size is more.

3.         Keep in mind, your application is bundled and disfeatured.

4.         Keep in mind, your application doesn’t have un-necessary import statements.

5.         Keep in mind, your application’s 3rd party unused library. If exist and not used, removed from your application.

6.         Remove your application dependencies if not required.

**When will ngInit be called? How would you make use of ngOnInit()?**

In Angular 1.x, **ngInit** is called when template is re-rendered. In other words “**ng-init**” is called, when I take turns back to a page.

In Angular2, there is no “**ng-init**” but we can create a ways like this using the directive and ngOnInit class. Angular 2 provides life cycle hook **ngOnInit** by default.

The **ngOnInit** is invoked when the component is initialized and invoked only once when the directive is instantiated. It is a best practice to implement these life-cycle interfaces.

**Differences - Constructor Vs. ngOnInit**

**Angular 2 Constructors:-**

1.      The **constructor** is a default method runs when component is being constructed.

2.      The constructor is a typescript **feature** and it is used only for a class **instantiations** and nothing to do with Angular 2.

3.      The constructor called first time before the **ngOnInit**().

**Angular 2 ngOnInit**:-

1.      The **ngOnInit** event is an Angular 2 life-cycle event method that is called after the first ngOnChanges and the ngOnInit method is use to parameters defined with @**Input** otherwise the constructor is **OK**.

2.      The **ngOnInit** is called after the constructor and ngOnInit is called after the first ngOnChanges.

3.      The **ngOnChanges** is called when an input or output binding value changes.

**WHAT ARE THE FEATURES OF ANGULAR 4.3?**

Features in Angular version 4.3 are:

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

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

**Angular 5 Performance Improvements - Angular 5**

1.     Use of addEventListener for the faster rendering and it is the core functionality.

2.     Update to new version of build-optimizer.

3.     Added some Improvements on the abstract class methods and interfaces

4.     Remove decorator DSL which depends on Reflect for Improve the Performance of Apps and This is the core functionality.

5.     Added an option to remove blank text nodes from compiled templates

6.     Switch Angular to use Static-Injector instead of Reflective-Injector.

7.     Improve the applications testing.

8.     Improve the performance of hybrid applications

9.     Improvements on Lazy loading for Angular

**Some Improvement on HttpClient** – This is used for Applications communicate with backend services over the HTTP protocol!

1.     Improvement on Type-checking the response

2.     Improvement on Reading the full response

3.     Improvement on Error handling and fetching error details

4.     Improvement on Intercepting all requests or responses

5.     Improvement on Logging

6.     Improvement on Caching

7.     Improvement on XSRF Protection

**Added Features - Angular 5**

1.     Added Representation of Placeholders to xliff and xmb in the compiler

2.     Added an Options Arg to Abstract Controls in the forms controls

3.     Added add default updateOn values for groups and arrays to form controls

4.     Added updateOn blur option to form controls

5.     Added updateOn submit option to form controls

6.     Added an Events Tracking Activation of Individual Routes

7.     Added NgTemplateOutlet API as stable in the common controls

8.     Create StaticInjector which does not depend on Reflect polyfill

9.     Added [@.disabled] attribute to disable animation children in the animations

**Router Life Cycle Events – Angular 5**

Added new router life cycle events for Guards and Resolvers -

1.     GuardsCheckStart,

2.     GuardsCheckEnd,

3.     ResolveStart and

4.     ResolveEnd

**Angular 5 Bug Fixes - Angular 5**

1.     Fixed compilation error by using the correct type for providers

2.     Skip PWA test when redeploying non-public commit

3.     Don't strip CSS source maps. This is the compiler related fix

4.     Remove tsickle (language-service) dependency

5.     Support persisting dynamic styles within animation states

6.     Ignore @import in multi-line css

7.     Fix platform-browser-dynamic

8.     Forbid destroyed views to be inserted or moved in VC

9.     Support persisting dynamic styles within animation states

**What's New In Angular 6? What Are Improvements In Angular 6?**

The [**Angular Team**](https://www.code-sample.com/2018/01/whats-new-in-angular-6.html) are working on lots of bug fixes, new features and added/update/remove/ re-introduce/ and may more things.

**1)** Typescript 2.6.x supports

**2)** Added Angular Material and CDK Stable

**3)** Component Dev Kit (CDK) - CDK allows you to build your own library of UI components using Angular Material.

**4)** Improved decorator error messages

**5)** Fix platform-detection example for Universal

**6)** Ivy Renderer - It is a new backward compatible and main focused area - speed improvements, size reduction, and increased flexibility.

**7)** Add afterContentInit and afterContentChecked to render

**8)** Added to supports of nativeElement

**9)** Added Optional generic type for ElementRef

The Example looks like -

@ViewChild('your-element') yourElement:ElementRef;

**10)** Bazel Compiler - Bazel only rebuilds what is necessary.

**11)** Added Test Comment

**12)** Add missing lifecycle tests for projected components

**13)** Closure Compiler - Closure Compiler consistently generates smaller bundles.

**14)** Rename QueryPredicate to LQuery and LQuery to LQueries

**15)** Service Worker - Service worker is a script that runs in the web browser. It also manages caching for an application.

**16)** Added multiple validators for array method of FormBuilder

The Example looks like -

import { Component } from '@angular/core';

import {FormsModule, FormBuilder, FormGroup} from '@angular/forms';

constructor(private fb: FormBuilder) {}

myForm: FormGroup;

ngOnInit() {

  this.myForm = this.fb.group({

      text: ['', Validators.required],

      options: this.fb.array([], [MyValidators.minCount, MyValidators.maxCount])

  });

}

**17)** Handle string with and without line boundary - Now Handle string with and without line boundary (^ & $) on pattern validators. Previously, it works with string not boundaries.

**18)** AbstractControl statusChanges - Previous version, not emits an event when you called “markAsPending” but now emits an event of "PENDING" when we call AbstractControl markAsPending.

**19)** Updates on NgModelChange - Now emitted after value and validity is updated on its control. Previously, it was emitted before updated.

Previously it looks like -

<input [(ngModel)]="name" (ngModelChange)="onChange($event)">

And

onChange(value) {

  console.log(value);   // would log the updated value, not old value

}

Now, its looks like -

<input #modelDir="ngModel" [(ngModel)]="name" (ngModelChange)="onChange(modelDir)">

And

onChange(NgModel: NgModel) {

  console.log(NgModel.value);// would log old value, not updated value

}

**20)** Allow HttpInterceptors to inject HttpClient –

Previously, an interceptor attempting to inject HttpClient directly would receive a circular dependency error, as HttpClient was constructed via a factory which injected the interceptor instances. Users want to inject HttpClient into interceptors to make supporting.

Either HttpClient or the user has to deal specially with the circular Dependency. This change moves that responsibility into HttpClient itself. By utilizing a new class HttpInterceptingHandler which lazily Loads the set of interceptors at request time, it's possible to inject HttpClient directly into interceptors as construction of HttpClient no longer requires the interceptor chain to be constructed.

**21)** Add navigationSource and restoredState to NavigationStart – Currently, NavigationStart there is no way to know if navigation was triggered imperatively or via the location change. These two use cases should be handled differently for a variety of use cases (e.g., scroll position restoration). This PR adds a navigation source field and restored navigation id (passed to navigations triggered by a URL change).

**22)** Add type and hooks to directive def

**23)** Enable size tracking of a minimal CLI render3 application

**24)** Add canonical view query

**25)** Language Service – The 2.6 version of Typescript’s “resolveModuleName” started to require paths passed to be separated by '/' instead of being able to handle '\'.

**What Is Service Workers?**

A Service Worker is a script which runs in the web browsers and manages to the caching for web applications. This script runs in the separates and background and don't need any user interactions.

They can query a local cache and deliver a [cached response,](https://www.code-sample.com/2017/09/angular-4-httpcache-httpinterceptor.html) if it is available in the cached. This makes more reliable and increase the performance.

A Service Worker is a programmable network proxy and it intercept all outgoing HTTP requests and use to allowing you to control how network requests from your page are handled.

The Service Worker is a method that enables applications to take advantage of persistent data in the background processing, including hooks to enable bootstrapping of web applications while offline.

**What Is Service Workers in Angular 5+?**

Angular 5+ start using service workers and the service workers are increased the apps reliability and performance without needing to code against this.

This is the great advantages for angular and Angular’s service worker is designed for -

* Improve the performance regarding the unreliable network connection
* Minimizing the risks of serving outdated content
* It’s Optimize the end user experience

**The main Design Goal of Angular's Service Worker**

* Caching an application
* When users refresh applications, they see firstly latest version cached file.
* The Updates happen in the background process. Do not interrupt to other process.
* When Updates, it’s happen the previous version of the application is served until an update ready to use

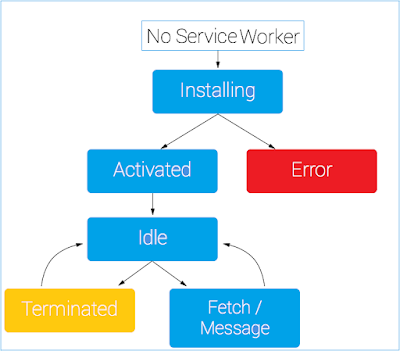
**Prerequisites to Supports Service Workers**

We must have the following Angular and [Angular CLI](https://www.code-sample.com/2017/11/angular-cli-updating-angular-cli.html) versions and also our web application must run in a web browser that supports service workers.

* Angular 5 or later
* Angular CLI 1.6 or later

**What Is Service Worker Life Cycle?**

A service worker has a life cycle that is completely separate from your web apps page.

[](https://1.bp.blogspot.com/-WKlfYcbswVA/WjiyL-VKi0I/AAAAAAAARbg/eZ9EE2bPK8oxTP2Ui5nuAP1EsSyCTOdNwCLcBGAs/s1600/service+worker+lifecycle.png)

To install a service worker for our site, we need to register it, which we do in our pages. To Registering a service worker will cause the browser to start the service worker install step in the background process.

Prerequisites to Supports Service workers-

* Browser support
* You need HTTPS

**How To Register A Service Worker?**

To install a service worker you need to kick start the process by registering it in your page. This tells the browser where your service worker JavaScript file lives.

You can call below **register** () every time a page loads without concern; the browser will figure out if the service worker is already registered or not and handle it accordingly.

if ('serviceWorker' in navigator) {

  window.addEventListener('load', function() {

    navigator.serviceWorker.register('/sw.js').then(function(registration) {

      // If Registration was successful

      console.log('Success Registration - ', registration.scope);

    },

    function(err) {

      // If Registration was failed!

      console.log('Failed Registration - ', err);

    });

  });

}

This code checks to see if the service worker API is available, and if it is, the service worker at /sw.js is registered once the page is loaded.

**How To Install a Service Worker?**

After a controlled page kicks off the registration process, let's shift to the point of view of the service worker script, which handles the install event.

Example looks like -

self.addEventListener('install', function(event) {

  // Perform install steps

});

Inside of our install callback, we need to take the following steps -

* Open a cache
* Cache our files
* Confirm whether all the required assets are cached or not

Example looks like -

var CACHE\_NAME = 'my-site-cache-v1.0';

var urlsToCache = [

  '/',

  '/styles/site.css',

  '/script/site.js'

];

self.addEventListener('install', function(event) {

  // Perform install steps

  event.waitUntil(

    caches.open(CACHE\_NAME)

      .then(function(cache) {

        console.log('Opened cache');

        return cache.addAll(urlsToCache);

      })

  );

});

**How To Cache and return Requests?**

After a service worker is installed and the user navigates to a different page or refreshes, the service worker will begin to receive fetch events.

self.addEventListener('fetch', function(event) {

  event.respondWith(

    caches.match(event.request)

      .then(function(response) {

        // Cache hit - return response

        if (response) {

           return response;

        }

        return fetch(event.request);

      })

  );

});

# What's New in HttpClient and HttpClientModule in Angular 4/5?

**HttpClientModule** and **HttpClient**and it’s available in package @angular/common/http.

The web applications communicate with backend services over the HTTP protocol and the browsers support to the**XMLHttpRequest** interface and the **fetch ()** API to execute HTTP request.

The new **HttpClient** service is included in the **HttpClientModule** and it used to initiate **HTTP** request and responses in angular apps.

The **HttpClient** is more modern and easy to use alternative of HTTP.

Also the **HttpClient** is use the **XMLHttpRequest** browser API to execute HTTP request and it specific the HTTP request type’s i.e.

* Get()
* Post()
* Put()
* Delete()
* Patch()
* Head()
* Jsonp()

**Setup and install the latest version of Angular**- Angular CLI

npm install -g @angular/cli@latest

**Use of HttpClient Services in App Module** - [import HttpClient module in **app.module.ts**]

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    HttpClientModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

**How To Use HttpClient in Angular?**

import { Component, OnInit } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

  //variables

  baseUrl ="https://code-sample.com/";

  users = null;

  //injects HttpClient into your component or service.

  constructor(private http: HttpClient){  }

  //Load User info.

  ngOnInit(): void {

  //Make the HTTP request:

    this.http.get(this.baseUrl +'api/users/').subscribe(data => {

      this.users = data;

    },

    err => {

      console.log("Error- something is wrong!")

    });

  }

}

**Example 2 –**HTTP request for GET/POST Method

import { Component, OnInit } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

  baseUrl ="https://abccom/";

  users = null;

 // Inject HttpClient into your component or service.

  constructor(private http: HttpClient){  }

  //Load User info.

  ngOnInit(): void {

     // Make the HTTP request:

    this.http.get(this.baseUrl +'api/users/').subscribe(data => {

      this.users = data;

    },

    err => {

      console.log("Error- something is wrong!")

    });

  }

  addUser = function(){

    let user ={

      id: 1,

      name:’ Srini’;,

      user\_Id:9979,

      site : 'https://abc.com'

    }

    //Make the HTTP Post Request

    this.http.post(this.baseUrl +'api/addUser/', user)

        .subscribe(

          result => {

            console.log("The User added successfully!");

            console.log(result);

          },

          err => {

            console.log("Error- something is wrong!")

          });

  }

}