GANATAN

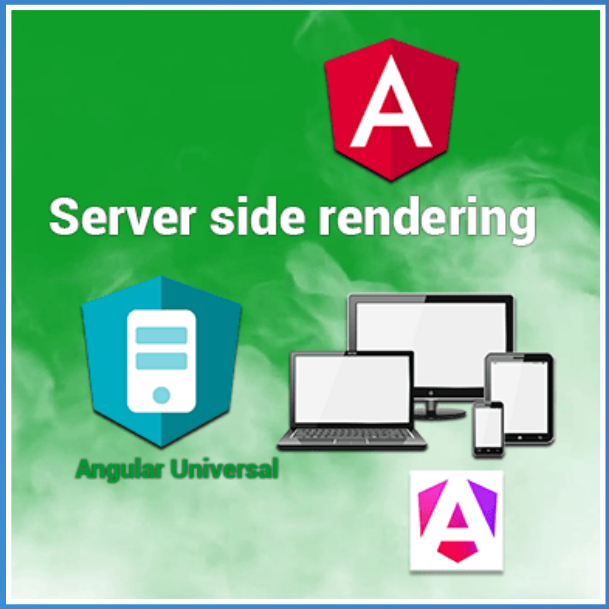
Server side rendering avec Angular

Guide Complet

Mise à jour du 1 mars 2025

Plus un **site web** est **connu** plus son nombre de visiteurs sera élevé.
Le fait d'être connu ou non dépendra de son **référencement**.

Si votre site est écrit avec **javascript** le référencement ne fonctionnera pas.



Pour résoudre ce problème vous devez utiliser le concept de **Server Side Rendering** ou **SSR**

Dans ce tutorial nous allons créer une application web avec **Angular version 19.2.0**

Puis nous appliquerons le **Server Side Rendering** dans cette Application Web.

Comment le faire ?

Pour débuter notre projet voici un résumé de ce que nous allons faire.

- **Référencement et SSR la Théorie**
  Tout d'abord il nous faut comprendre ce qu'est le référencement.
  Référencement, SEO et SSR, tout est expliqué en détails
- **Initialisation du projet from scratch**
  Nous utiliserons Angular CLI pour la mise en place d'un projet,
  En utilisant les best practices nous déploierons le SSR.
- **Utilisation d'un projet complexe**
  Nous utiliserons un projet complet que nous adapterons pour utiliser le SSR.
- **Mise à jour du projet**
  Vérifier les dépendances utilisées et les mettre à jour.
- **Déploiement**
  Comment déployer et tester notre application sur internet.
- **Code source**
  Le code complet du projet est disponible sur Github.

La théorie

Internet c'est quoi
- Des milliards d'humains
- Des millions de sites web sur une toile d'araignée
- Et surtout le moyen de les réunir

Commencons par les humains

Ils sont des milliards, ils mènent leur vie et se posent chaque jour des milliards de questions.

Internet c'est le bien connu **www**.

Ou **World Wide Web** littéralement la **toile d'araignée mondiale**.
On y trouve des millions de site web qui sont censés rèpondre à toutes vos questions.

Donc ça se résume à ça.
Les questions d'un côté, les réponses de l'autre.

On imagine déjà l'offre et la demande

Pour faire rencontrer tout ce beau monde il nous faudra des intermédiaires.

Ce seront les moteurs de recherche ou Search engine (SE dans SEO !)

Des moteurs de recherche il y en a un paquet.
Mais si on fait un classement ça donne cette liste

- **Google : 91,88%**
- Bing : 3,19%
- Yandex : 1,52%
- Yahoo! : 1,33%
- Baidu : 0,76%
- DuckDuckGo : 0,64%

C'est ce que l'on appelle une hégémonie.
Ainsi va le monde on va s'adapter.

En tout cas le résumé pour notre offre et notre demande ce sera cette image.

Vous l'aurez compris.
Un site web connu c'est un site qui a un bon référencement.
Et surtout qui a un bon référencement avec Google. CQFD et pas le choix.

Javascript et le référencement

Si vous êtes développeur ou vous voulez le devenir il vous faudra utiliser des outils pour construire un site web.

Actuellement 3 outils se démarquent pour créer un site.
- Angular
- React
- Vuejs

Ce sont des outils qui utilisent le langage Javascript.
Et le problème c'est que Javascript est le problème.

Si vous utilisez Javascript le référencement de google ne fonctionnera pas.
Je vous expliquerai plus loin dans ce tutoriel pourquoi en détail.

Donc il nous faut trouver une solution.
Dans le cas d'Angular la soution s'appelle Le Server Side Rendering ou SSR.
Et nous utiliserons un outil dédié à cela angular Universal.

Passons maintenant à la pratique.

Qu’allons nous faire ?

Je vous propose deux possibiltés

- Création d'un projet SSR from scratch
- Utilisation d'un projet existant et rajout du SSR

Commençons par la solution la plus rapide mais la moins détaillée.

Création d'une application From Scratch

Si vous êtes pressé par la peine d'aller plus loin.

je vous donne les commandes qui permettent d'obtenir une application SSR en quelques minutes

J'avoue que c'est un peu bref mais cela a le mérite de fonctionner (et plutôt bien même).

Si vous voulez plus d'explications sur ce qui a été fait je vous conseille de passer à la suite et l'utilisation d'un projet existant.

Création d'une application from scratch

```
# Désinstallez Angular CLI (au cas ou une ancienne version d'Angular aurait été installée)
npm uninstall -g @angular/cli

# Installez Angular CLI version spécifique (la dernière si possible)
npm install -g @angular/cli@18.0.3

# Créez un répertoire demo (le nom est ici arbitraire)
mkdir demo

# Allez dans ce répertoire
cd demo

# Générez un projet appelé angular-starter avec choix manuel des options (répondez oui à tout)
ng new angular-starter

# Positionnez vous dans le projet
cd angular-starter

# Exécutez l'application en mode normal
npm run start

# Testez l'application dans votre navigateur
http://localhost:4200

# Installation des dépendances nécessaires au SSR
ng add @angular/ssr

# Compilez l'application en mode SSR
npm run build

# Exécutez l'application en mode SSR
npm run serve:ssr:angular-starter

# Testez l'application dans votre navigateur
http://localhost:4000
```

Qu’allons nous faire ?

Je vous propose la deuxième possibilité pour les plus curieux et les plus patients.
Le projet Angular de base que nous utiliserons dispose déjà des caractéristiques suivantes

- Généré avec **Angular CLI**
- Le **Routing**
- Le **Lazy Loading**
- Le framework CSS **Bootstrap**

Tous les sources utilisés sont indiqués en fin de tutoriel.

L' application finale est à l'adresse suivante

- https://angular.ganatan.com

Avant de commencer

Pour être visité par un grand nombre d'utilisateurs, un site web se doit de remplir deux conditions essentielles.

- S'afficher le plus rapidement possible.
- Etre bien référencé par les moteurs de recherche.

La technique qui permet de le faire porte un nom.

- Le **Rendu Côté Serveur** ou **Server Side Rendering** en anglais.

Nous allons appliquer cette technique dans un projet Angular.

Pour cela nous emploierons la technologie préconisée par les équipes de Google

- **Angular Universal.**

Cette technologie permettra d'améliorer le **référencement naturel** ou **SEO** (**Search Engine Optimization**) de notre site.

Création du projet Angular

Pour pouvoir continuer ce tutoriel nous devons bien evidemment disposer de certains éléments

- **Node.js** : La plateforme javascript
- **Git** : Le logiciel de gestion de versions.
- **Angular CLI** : L'outil fourni par Angular.
- **Visual Studio code** : Un éditeur de code.

Vous pouvez consulter le tutoriel suivant qui vous explique en détails comment faire

- https://www.ganatan.com/tutorials/demarrer-avec-angular

Nous allons utiliser un projet existant

```
# Créez un répertoire demo (le nom est ici arbitraire)
mkdir demo

# Allez dans ce répertoire
cd demo

# Récupérez le code source sur votre poste de travail
git clone https://github.com/ganatan/angular-modules.git

# Allez dans le répertoire qui a été créé
cd angular-modules

# Exécutez l'installation des dépendances (ou librairies)
npm install

# Exécutez le programme
npm run start

# Vérifiez son fonctionnement en lançant dans votre navigateur la commande
http://localhost:4200/
```

Théorie

Les pages web générées avec des frameworks javascript utilisent javascript.

Les moteurs de recherche ont à l'heure actuelle du mal à interpréter le javascript.

Nous allons vérifier cette notion de façon pratique.

Nous allons éxécuter notre application avec le script correspondant.

```
# Exécution de l'application
npm run start

# Affichage du site dans le navigateur
http://localhost:4200/
```

Avant de commencer

Nous allons vérifier le code source produit dans la page correspondante.

En utilisant le navigateur **Chrome** il faut taper **Ctrl + U** pour voir le code html.

On remarque que le code "**Features**" qui s'affiche dans le navigateur n'apparait pas dans le code.

```
<!doctype html>
<html lang="en">

<head>
  <script type="module" src="/@vite/client"></script>

  <meta charset="utf-8">
  <title>AngularStarter</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <!-- Google tag (gtag.js) -->
  <script async="" src="https://www.googletagmanager.com/gtag/js?id=YOUR-ID"></script>
  <script>
    window.dataLayer = window.dataLayer || [];
    function gtag() { dataLayer.push(arguments); }       gtag('js', new Date());
    gtag('config', 'YOUR-ID');
  </script>

  <link rel="stylesheet" href="styles.css">
</head>

<body>
  <app-root></app-root>
  <script src="polyfills.js" type="module"></script>
  <script src="scripts.js" defer></script>
  <script src="main.js" type="module"></script>
</body>

</html>
```

En cliquant sur **main.js** nous ouvrons ce fichier qui contient le texte "Features"

Exécutons une compilation avec *npm run build.*

Le répertoire **dist/angular-starter** contient le fichier **main.js.**

Ouvrons ce fichier avec notre éditeur VS code , puis effectuons une recherche (Ctrl + F) du texte "Features".

Il contient le texte "Features".

Le fichier main.js est un fichier javascript, il sera donc mal interprété par les moteurs de recherche.

Nous verrons plus loin dans ce tutoriel qu'une fois le **SSR appliqué** le code apparait **directement** dans le **code HTML** et sera ainsi bien interprété par les moteurs de recherche.

Installation

L'outil que nous utiliserons pour appliquer le SSR à notre projet est
- **@angular/ssr** version **19.2.0**

L'outil précédent utilisé jusqu'à la version 16 d'Angular était
- **Angular universal** version **16.2.0**

La dernière version de cet outil est disponible ci-dessous
- https://github.com/angular/universal/releases

@angular/ssr permet de générer des pages statiques via un processus appelé **Server side rendering (SSR)**.

La procédure à suivre est détaillée sur le site officiel d'Angular.

https://angular.io/guide/ssr

Nous allons utiliser une simple commande CLI

```
# Installation
ng add @angular/ssr
```

Angular universal

Pour rappel angular CLI utilise via la directive **ng add** le principe des **schematics** pour modifier notre code et l'adapter à la nouvelle fonctionnalité (ici le ssr).

De nombreuses opérations ont été effectuées automatiquement sur notre projet.

Si nous avions dû réaliser cette opération manuellement voici les différentes étapes que nous aurions dû suivre.

- Installation des nouvelles **dépendances** nécessaires
- Modification du fichier **angular.json**
- Création du fichier src/app/**app.config.server.ts**
- Création du fichier src/**main.server.ts**
- Création du fichier **server.ts**
- Modification du fichier **tsconfig.app.json**
- Modification du fichier src/app/**app.config.ts**
- Modification du fichier **package.json**

Installation des dépendances.

```
# Installer les nouvelles dépendances dans package.json
npm install --save @angular/platform-server
npm install --save @angular/ssr
npm install --save express
npm install --save @types/express
npm install --save @types/node
```

Modification du fichier **angular.json.**

Rajout de propriétés liées au SSR

- **server**
- **prerender**
- **ssr**

angular.json

```
        "builder": "@angular-devkit/build-angular:application",
        "options": {
          "outputPath": "dist/angular-starter",
          "index": "src/index.html",
          "browser": "src/main.ts",
          "polyfills": [
            "zone.js"
          ],
          "tsConfig": "tsconfig.app.json",
          "assets": [
            "src/favicon.ico",
            "src/assets"
          ],
          "styles": [
            "node_modules/@fortawesome/fontawesome-free/css/all.min.css",
            "node_modules/bootstrap/dist/css/bootstrap.min.css",
            "src/assets/params/css/fonts.googleapis.min.css",
            "src/styles.css"
          ],
          "scripts": [
            "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
          ],
          "server": "src/main.server.ts",
          "prerender": true,
          "ssr": {
            "entry": "server.ts"
          }
        },
```

Création du fichier **app.config.server.ts**

src/app/app.config.server.ts

```
import  { mergeApplicationConfig, ApplicationConfig } from '@angular/core';
import  { provideServerRendering } from '@angular/platform-server';
import  { appConfig } from './app.config';

const serverConfig: ApplicationConfig = {
  providers: [
    provideServerRendering()
  ]
};

export const config = mergeApplicationConfig(appConfig, serverConfig);
```

Création du fichier **main.server.ts**

src/main.server.ts

```
import  { bootstrapApplication } from '@angular/platform-browser';
import  { AppComponent } from './app/app.component';
import  { config } from './app/app.config.server';

const bootstrap = () => bootstrapApplication(AppComponent, config);

export  default bootstrap;
```

Création du fichier **server.ts**

Le port utilisé par défaut est 4000 nous pouvons le changer si nécessaire dans ce fichier.

server.ts

```
import { APP_BASE_HREF } from '@angular/common';
import { CommonEngine } from '@angular/ssr';
import express from 'express';
import { fileURLToPath } from 'node:url';
import { dirname, join, resolve } from 'node:path';
import bootstrap from './src/main.server';

// The Express app is exported so that it can be used by serverless Functions.
export function app(): express.Express {
  const server = express();
  const serverDistFolder = dirname(fileURLToPath(import.meta.url));
  const browserDistFolder = resolve(serverDistFolder, '../browser');
  const indexHtml = join(serverDistFolder, 'index.server.html');
  const commonEngine = new CommonEngine();
  server.set('view engine', 'html');
  server.set('views', browserDistFolder);
  // Example Express Rest API endpoints
  // server.get('/api/**', (req, res) => { });
  // Serve static files from /browser
  server.get('*.*', express.static(browserDistFolder, {
    maxAge: '1y'
  }));

  // All regular routes use the Angular engine
  server.get('*', (req, res, next) => {
    const { protocol, originalUrl, baseUrl, headers } = req;
    commonEngine
      .render({
        bootstrap,
        documentFilePath: indexHtml,
        url: `${protocol}://${headers.host}${originalUrl}`,
        publicPath: browserDistFolder,
        providers: [{ provide: APP_BASE_HREF, useValue: baseUrl }],
      })
      .then((html) => res.send(html))
      .catch((err) => next(err));
  });
  return server;
}

function run(): void {
  const port = process.env['PORT'] || 4000;

  // Start up the Node server
  const server = app();
  server.listen(port, () => {
    console.log(`Node Express server listening on http://localhost:${port}`);
  });
}
run();
```

Modification du fichier

- **tsconfig.app.json**

tsconfig.app.json

```
/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "outDir": "./out-tsc/app",
    "types": [
      "node"
    ]
  },
  "files": [
    "src/main.ts",
    "src/main.server.ts",
    "server.ts"
  ],
  "include": [
    "src/**/*.d.ts"
  ]
}
```

src/app/app.config.ts

```
import  { ApplicationConfig } from '@angular/core';
import  { provideRouter } from '@angular/router';

import  { routes } from './app.routes';
import  { provideClientHydration } from '@angular/platform-browser';

export  const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideClientHydration()]
};
```

Modification du fichier **package.json**

package.json

```
"scripts": {
    ...
    "serve:ssr:angular-starter": "node dist/angular-starter/server/server.mjs"
    ...
}
```

Mise à jour

Nous pouvons en profiter pour mettre à jour les dépendances du fichier package.json et adapter les descripteurs de version.

Les dépendances suivantes
- @angular/ssr

Peuvent être mises à jour avec les versions
- 19.0.0

Le fichier contiendra au final les dépendances suivantes.

```
  "dependencies": {
    "@angular/animations": "19.2.0",
    "@angular/common": "19.2.0",
    "@angular/compiler": "19.2.0",
    "@angular/core": "19.2.0",
    "@angular/forms": "19.2.0",
    "@angular/platform-browser": "19.2.0",
    "@angular/platform-browser-dynamic": "19.2.0",
    "@angular/platform-server": "19.2.0",
    "@angular/router": "19.2.0",
    "@angular/ssr": "19.2.0",
    "@fortawesome/fontawesome-free": "6.7.2",
    "bootstrap": "5.3.3",
    "express": "4.21.2",
    "rxjs": "7.8.2",
    "tslib": "2.8.1",
    "zone.js": "0.15.0"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "19.2.0",
    "@angular/cli": "19.2.0",
    "@angular/compiler-cli": "19.2.0",
    "@types/express": "5.0.0",
    "@types/jasmine": "5.1.7",
    "@types/node": "22.13.8",
    "angular-eslint": "19.2.0",
    "eslint": "9.21.0",
    "jasmine-core": "5.6.0",
    "karma": "6.4.4",
    "karma-chrome-launcher": "3.2.0",
    "karma-coverage": "2.2.1",
    "karma-jasmine": "5.1.0",
    "karma-jasmine-html-reporter": "2.1.0",
    "typescript": "5.7.3",
    "typescript-eslint": "8.25.0"
  }
```

Debuggage

Le code de note application va provoquer une erreur sur la compilation.

Donc le code avant après.

Il concerne le fichier **app.component.ts**

app.component.ts (avant)

```
import  { Component, OnInit } from '@angular/core';
import  { CommonModule } from '@angular/common';
import  { RouterLink, RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterLink, RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

 export class AppComponent implements OnInit {
  title = 'angular-routing';
  footerUrl = 'https://www.ganatan.com';
  footerLink = 'www.ganatan.com';
  ngOnInit(): void {

    const navMain = document.getElementById('navbarCollapse');
    if (navMain) {
      navMain.onclick = function onClick() {
        if (navMain) {
          navMain.classList.remove("show");
        }
      }
    }
  }

}
```

app.component.ts (après)

```
import  { Component, OnInit } from '@angular/core';
import  { CommonModule } from '@angular/common';
import  { RouterLink, RouterOutlet } from '@angular/router';

import  { Inject, PLATFORM_ID } from '@angular/core';
import  { isPlatformBrowser } from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterLink, RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

 export class AppComponent implements OnInit {
  title = 'angular-routing';
  footerUrl = 'https://www.ganatan.com';
  footerLink = 'www.ganatan.com';

  constructor(
    @Inject(PLATFORM_ID) private platformId: object) {
  }

  ngOnInit(): void {

    if (isPlatformBrowser(this.platformId)) {
      const navMain = document.getElementById('navbarCollapse');
      if (navMain) {
        navMain.onclick = function onClick() {
          if (navMain) {
            navMain.classList.remove("show");
          }
        }
      }
    }
  }

}
```

Conclusion

Il ne reste plus qu'à tester tous les scripts précédents et finaliser par le SSR.

```
# Développement
npm run start
http://localhost:4200/

# Tests
npm run test

# SSR Compilation
npm run build

# SSR execution
npm run serve:ssr:angular-starter

http://localhost:4000/
```

Enfin nous allons vérifier le code source produit dans la page correspondante à la compilation SSR..

En utilsant le navigateur **Chrome** il faut taper **Ctrl + U** pour voir le code html.

On remarque que le code "**Features**" s'affiche cette fois dans le navigateur.

La page sera dès lors bien interprétée par les moteurs de recherche.

Remarque
Certaines versions d'Angular ne permettent pas de vérifier le résultat SSR dans votre navigateur.
Pourtant SSR fonctionne côté serveur avec les robots google.

Pour le vérifier utilisez le logiciel curl

- Par exemple sur localhost

  curl http://localhost:4000/ > ssr-results.txt

- Sur la version Démo

  curl https://angular.ganatan.com/ > ssr-results.txt

Puis vérifiez le contenu du fichier ssr-results.txt

Vous verrez dans le code HTML apparaitre le texte voulu.

Autre preuve Utilisez SEOQUAKE (SEO Toolbox) pour vérifier le SEO sur angular.ganatan.com/

J'applique ainsi le SSR sur www.ganatan.com et angular.ganatan.com

Code source

Le code source utilisé **en début de tutoriel** est disponible sur github
https://github.com/ganatan/angular-react-modules

Le **code source** de ce tutoriel est disponible sur GitHub.
Utilisez git pour récupérer ce code et vérifier son fonctionnement.

Il vous suffit de vous rendre à l'adresse suivante
https://github.com/ganatan/angular-ssr

Et n'oubliez pas les P'tits Loups si le code source vous plait vous savez ce qu'il vous reste à faire.

Un star sur github peut changer un homme

Les étapes suivantes vous permettront **d'obtenir une application prototype**

- Etape 7 : Progressive Web App avec Angular
- Etape 8 : Search Engine Optimization avec Angular
- Etape 9 : HttpClient avec Angular

La dernière étape permet **d'obtenir un exemple d'application**

- Créer une application Web complète avec Angular

Le **code source de cette application finale** est disponible sur GitHub
https://github.com/ganatan/angular-app