GANATAN

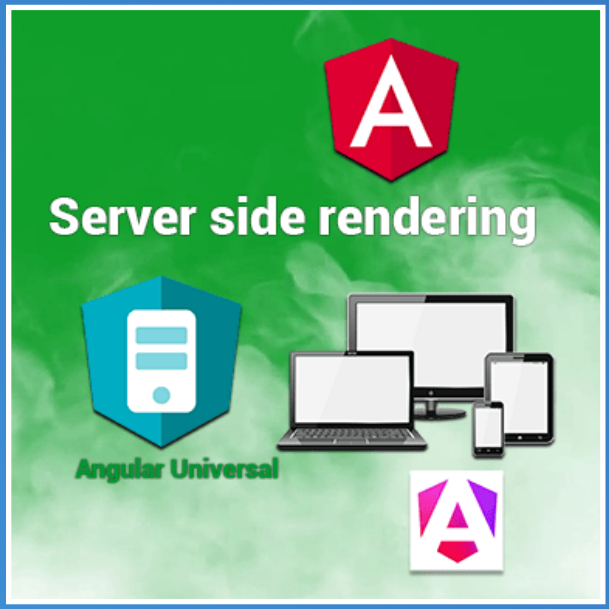
Server side rendering with Angular

Complete Guide

Update on July 31, 2025

The more **famous** a **website** is, the higher the number of visitors it will have. Whether you are known or not will depend on your **SEO** .

If your site is written with **javascript** SEO will not work.



To solve this problem you need to use the concept of **Server Side Rendering** or **SSR**

In this tutorial we will create a web application with **Angular version 20.1.3**

Then we will apply **Server Side Rendering** in this Web Application.

How to do it?

To begin our project, here is a summary of what we are going to do.

- **SEO and SSR Theory**
  First of all we need to understand what SEO is.
  SEO, SSR and referencing, everything is explained in detail
- **Initializing the project from scratch**
  We will use Angular CLI for setting up a project,
  Using best practices we will deploy the SSR.
- **Using a complex project**
  We will use a complete project that we will adapt to use the SSR.
- **Project Update**
  Check the dependencies used and update them.
- **Deployment**
  How to deploy and test our application on the internet.
- **Source code**
  The full code of the project is available on Github.

The theory

What is the Internet?
- Billions of humans
- Millions of websites on a spider's web
- And above all the means of bringing them together

Let's start with humans

There are billions of them, they lead their lives and ask themselves billions of questions every day.

Internet is the well-known **www** .

Or **World Wide Web** literally the **global spider's web** .
There are millions of websites that are supposed to answer all your questions.

So it comes down to this.
Questions on one side, answers on the other.

We can already imagine the supply and demand

To bring all these wonderful people together, we will need intermediaries.

These will be the search engines or Search engines (SE in SEO!)

There are a lot of search engines.

But if we make a classification it gives this list

- **Google: 91.88%**
- Bing: 3.19%
- Yandex: 1.52%
- Yahoo!: 1.33%
- Baidu: 0.76%
- DuckDuckGo: 0.64%

This is what is called hegemony.

This is how the world goes, we will adapt.

In any case, the summary for our supply and demand will be this image.

You will have understood.

A well-known website is a site that has good SEO.

And especially who has a good SEO with Google. QED and no choice.

Javascript and SEO

If you are a developer or want to become one, you will need to use tools to build a website.

Currently, 3 tools stand out for creating a site.
- Angular
- React
- Vuejs

These are tools that use the Javascript language.

And the problem is that Javascript is the problem.

If you use Javascript, Google SEO will not work.

I will explain why in detail later in this tutorial.

So we need to find a solution.

In the case of Angular the solution is called Server Side Rendering or SSR.

And we will use a tool dedicated to this angular Universal.

Now let's move on to practice.

What are we going to do?

I offer you two possibilities.

- Creating an SSR project from scratch
- Using an existing project and adding the SSR

Let's start with the quickest but least detailed solution.

Creating an Application From Scratch

If you are in a hurry to go further.

I give you the commands that allow you to obtain an SSR application in a few minutes

I admit that it is a little brief but it has the merit of working (and rather well even).

If you want more explanations on what was done I advise you to move on and use an existing project.

Creating an application from scratch

```
# Uninstall Angular CLI (in case an older version of Angular was installed)
npm uninstall -g @angular/cli

# Install Angular CLI specific version (latest if possible)
npm install -g @angular/cli@20.1.3

# Create a demo directory (the name is arbitrary here)
mkdir demo

# Go to this directory
cd demo

# Generate a project called angular-starter with manual choice of options (answer yes to all)
ng new angular-starter

# Position yourself in the project
cd angular-starter

# Run the application in normal mode
npm run start

# Test the application in your browser
http://localhost:4200

# Installing the necessary dependencies for the SSR
ng add @angular/ssr

# Compile the application in SSR mode
npm run build

# Run the application in SSR mode
npm run serve:ssr:angular-starter

# Test the application in your browser
http://localhost:4000
```

What are we going to do?

I offer you the second possibility for the most curious and the most patient.
The basic Angular project we will be using already has the following features

- Generated with **Angular CLI**
- The **Routing**
- **Lazy Loading**
- The **Bootstrap** CSS Framework

All sources used are indicated at the end of the tutorial.

The final application is at the following address

- https://angular.ganatan.com

Before you start

To be visited by a large number of users, a website must meet two essential conditions.

- Show up as quickly as possible.
- Be well referenced by search engines.

The technique that allows this to be done has a name.

- **Server Side Rendering** .

We will apply this technique in an Angular project.

For this we will use the technology recommended by the Google teams

- **Angular Universal.**

This technology will improve the **natural referencing** or **SEO** (**Search Engine Optimization**) of our site.

Creating the Angular project

To be able to continue this tutorial we obviously need to have certain elements

- **Node.js** : The JavaScript platform
- **Git** : The version control software.
- **Angular CLI** : The tool provided by Angular.
- **Visual Studio Code** : A code editor.

You can check out the following tutorial which explains in detail how to do it

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

We will use an existing project

```
# Create a demo directory (the name is arbitrary here)
mkdir demo

# Go to this directory
cd demo

# Get the source code on your workstation
git clone https://github.com/ganatan/angular-modules.git

# Go to the directory that was created
cd angular-modules
cd frontend-angular

# Run the dependencies (or libraries) installation
npm install

# Run the program
npm run start

# Check its operation by launching the command in your browser
http://localhost:4200/
```

Theory

Web pages generated with javascript frameworks use javascript.
Search engines currently have difficulty interpreting JavaScript.

We will verify this notion in a practical way.

We will run our application with the corresponding script.

```
# Running the application
npm run start

# Displaying the site in the browser
http://localhost:4200/
```

Before you start

We will check the source code produced in the corresponding page.

Using the **Chrome** browser you have to type **Ctrl + U** to see the html code.

We notice that the " **Features** " code that is displayed in the browser does not appear in the 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>
```

By clicking on **main.js** we open this file which contains the text "Features"

Let's run a build with *npm run build.*

The **dist/angular-starter** directory contains the **main.js** file **.**

Let's open this file with our VS Code editor, then do a search (Ctrl + F) for the text "Features".

It contains the text "Features".

The main.js file is a javascript file, so it will be misinterpreted by search engines.

We will see later in this tutorial that once the **SSR is applied** the code appears **directly** in the **HTML code** and will thus be well interpreted by search engines.

Installation

The tool we will use to apply SSR to our project is

- **@angular/ssr** version **20.1.3**

The previous tool used until Angular version 16 was

- **Angular universal** version **16.2.0**

The latest version of this tool is available below

- https://github.com/angular/universal/releases

@angular/ssr allows generating static pages through a process called **Server side rendering (SSR)** .

The procedure to follow is detailed on the official Angular website.

https://angular.io/guide/ssr

We will use a simple CLI command

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

Angular universal

As a reminder, Angular CLI uses the principle of **schematics** via the **ng add** directive to modify our code and adapt it to the new functionality (here the ssr).

Many operations were performed automatically on our project.

If we had to do this operation manually, here are the different steps we would have had to follow.

- Installing the necessary new **dependencies**
- Editing the **angular.json** file
- Creating the src/app/ **app.config.server.ts** file
- Creating the src/ **main.server.ts** file
- Creating the **server.ts** file
- Editing the **tsconfig.app.json** file
- Editing the src/app/ **app.config.ts** file
- **Modifying** the **package.json** file

Installing dependencies.

```
# Install new dependencies in 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
```

Editing the **angular.json file.**

Added SSR related properties

- **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"
          }
        },
```

Creating the **app.config.server.ts** file

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);
```

Creating the **main.server.ts** file

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;
```

Creating the **server.ts** file

The default port used is 4000 we can change it if necessary in this file.

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()]
};
```

Editing the **package.json** file

package.json

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

Update

We can take advantage of this to update the dependencies of the package.json file and adapt the version descriptors.

The following dependencies
- @angular/ssr

Can be updated with versions
- 19.0.0

The file will ultimately contain the following dependencies.

```
  "dependencies": {
    "@angular/common": "20.1.3",
    "@angular/compiler": "20.1.3",
    "@angular/core": "20.1.3",
    "@angular/forms": "20.1.3",
    "@angular/platform-browser": "20.1.3",
    "@angular/platform-server": "20.1.3",
    "@angular/router": "20.1.3",
    "@angular/ssr": "20.1.3",
    "@fortawesome/fontawesome-free": "7.0.0",
    "bootstrap": "5.3.7",
    "express": "5.1.0",
    "rxjs": "7.8.2",
    "tslib": "2.8.1",
    "zone.js": "0.15.1"
  },
  "devDependencies": {
    "@angular/build": "20.1.3",
    "@angular/cli": "20.1.3",
    "@angular/compiler-cli": "20.1.3",
    "@types/express": "5.0.3",
    "@types/jasmine": "5.1.8",
    "@types/node": "24.1.0",
    "angular-eslint": "20.1.1",
    "eslint": "9.32.0",
    "jasmine-core": "5.9.0",
    "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": "5.8.3",
    "typescript-eslint": "8.38.0"
  }
```

Debuggage

The code in this application will cause an error on compilation.

So the code before after.

It concerns the **app.component.ts** file

app.component.ts (before)

```
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 (after)

```
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

All that remains is to test all the previous scripts and finalize with the 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/
```

Finally we will check the source code produced in the page corresponding to the SSR compilation.

Using the **Chrome** browser, you have to type **Ctrl + U** to see the html code.

We notice that the " **Features** " code is displayed this time in the browser.

The page will then be well interpreted by search engines.

Noticed

Some versions of Angular do not allow you to check the SSR result in your browser.

Yet SSR works server side with google robots.

To check this use the curl software

- For example on localhost

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

- On the Demo version

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

Then check the contents of the ssr-results.txt file

You will see the desired text appear in the HTML code.

Other proof Use SEOQUAKE (SEO Toolbox) to check SEO on angular.ganatan.com/

So I apply SSR on www.ganatan.com and angular.ganatan.com

Code source

The source code used **at the beginning of the tutorial** is available on github
https://github.com/ganatan/angular-react-modules

The **source code** for this tutorial is available on GitHub.
Use git to fetch this code and verify that it works.

You just need to go to the following address
https://github.com/ganatan/angular-ssr

And don't forget the Little Wolves, if you like the source code you know what you have to do.

A star on github can change a man

The following steps will **get you a prototype application**

- Step 7: Progressive Web App with Angular
- Step 8: Search Engine Optimization with Angular
- Step 9: HttpClient with Angular

The last step is **to obtain a sample application**

- Build a Complete Web Application with Angular

The **source code for this final application** is available on GitHub

https://github.com/ganatan/angular-node-java-ai