APRIL 24 2019

Angular 7 Tutorial Part 2 - Create Base Project Structure & Webpack Config

Other parts available in Angular 7 tutorial series:

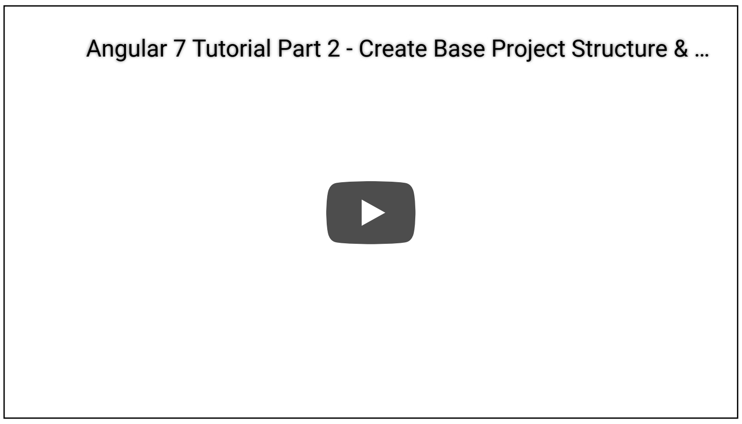
- Part 1 (Windows) - Setup Development Machine on Windows (https://www.youtube.com/watch?v=FNYsvgmblM0)
- Part 1 (Mac) - Setup Development Machine on Mac (https://www.youtube.com/watch?v=NLsey2LlJqo)
- Part 3 - Add Routing & Multiple Pages (/post/2019/04/29/angular-7-tutorial-part-3-add-routing-multiple-pages)
- Part 4 - Login Form, Authentication Service & Route Guard (/post/2019/05/17/angular-7-tutorial-part-4-login-form-authentication-service-route-guard)
- Part 5 - Registration Form & User Service (/post/2019/05/22/angular-7-tutorial-part-5-registration-form-user-service)
- Part 6 - Home Page & Alert Component (/post/2019/05/31/angular-7-tutorial-part-6-home-page-alert-component)

Next part coming soon:

- Part 7 (Optional) - Migrating to an Angular CLI Project

Angular 7 Tutorial Part 2

In part 2 of this Angular 7 tutorial series we're going to setup the base project structure and webpack config for an Angular 7 application.


Angular 7 Tutorial Part 2 - Create Base Project Structure & ...

The complete source code for this part of the tutorial is available on github at https://github.com/cornflourblue/angular-7-tutorial (https://github.com/cornflourblue/angular-7-tutorial) in the `part-2` folder.

Steps:

1. Create project folder
2. Create package.json
3. Install Angular 7 Dependencies
4. Install Angular 7 Dev Dependencies
5. Create tsconfig.json
6. Create webpack.config.js
7. Create src folder
8. Create app folder
9. Create Angular App Component template

Create project folder

You can name it anything you like, I called mine `angular-7-tutorial` , then open the folder in Visual Studio Code. This might seem like an obvious step but I decided to include every detail of the process from the beginning.

Create package.json

Inside the project folder create a file named `package.json` . This file contains project configuration information including package dependencies which get installed when you run npm install. More info available at https://docs.npmjs.com/files/package.json (https://docs.npmjs.com/files/package.json).

Open package.json in VS Code, set the `name` property to the same as the project folder name and set the `version` property to `1.0.0` . It should look something like this:

```
{
    "name": "angular-7-tutorial",
    "version": "1.0.0"
}
```

Install Angular 7 Dependencies

Run the following command from the project folder to install and save the npm packages required to run Angular 7 applications:

`npm install -save @angular/common@7 @angular/compiler@7 @angular/core@7 @angular/forms@7 @angular/platform-browser@7 @angular/platform-browser-dynamic@7 @angular/router@7 core-js@3 rxjs@6 zone.js@0`

The command will download the packages into a `node_modules` folder and add them to the `dependencies` property in the package.json file. So next time to install all of the packages you just need to run `npm install` and it will use the list in package.json.

The package.json should look like this after installing Angular 7 dependencies with the above command:

```
{
    "name": "angular-7-tutorial",
    "version": "1.0.0",
    "dependencies": {
        "@angular/common": "^7.2.13",
        "@angular/compiler": "^7.2.13",
        "@angular/core": "^7.2.13",
        "@angular/forms": "^7.2.13",
        "@angular/platform-browser": "^7.2.13",
        "@angular/platform-browser-dynamic": "^7.2.13",
        "@angular/router": "^7.2.13",
        "core-js": "^3.0.1",
        "rxjs": "^6.4.0",
        "zone.js": "^0.9.0"
    }
}
```

Below are details of each of the dependencies and a link to the documentation for each:

- **@angular/common**
  Common angular directives including NgIf, NgClass, NgForOf and pipes including AsyncPipe, UpperCasePipe, LowerCasePipe. Also includes the Angular HttpClient in the @angular/common/http subfolder.
  Docs: https://angular.io/api/common (https://angular.io/api/common)
- **@angular/compiler**
  Angular template compiler, used by @angular/platform-browser-dynamic to convert templates to JavaScript code that can run in the browser.
- **@angular/core**
  As the name suggests, these are the core services, utilities and functionality required by all Angular applications.
  Docs: https://angular.io/api/core (https://angular.io/api/core)
- **@angular/forms**
  Includes providers and directives for building both template-driven and reactive forms.
  Docs: https://angular.io/api/forms (https://angular.io/api/forms)
- **@angular/platform-browser**
  Includes core functionality for running Angular applications in different supported browsers.
  Docs: https://angular.io/api/platform-browser (https://angular.io/api/platform-browser)
- **@angular/platform-browser-dynamic**
  Includes providers and methods to compile, bootstrap and run Angular apps dynamically in the browser using JIT compilation.
  Docs: https://angular.io/api/platform-browser-dynamic (https://angular.io/api/platform-browser-dynamic)
- **@angular/router**
  Implements routing features which enable navigation between different routes (url paths) in an Angular application and mapping routes to different components.
  Docs: https://angular.io/guide/router (https://angular.io/guide/router)
- **core-js**
  A collection of polyfills that add support for features required by Angular that aren't natively supported yet in several browsers.
  Docs: https://github.com/zloirock/core-js (https://github.com/zloirock/core-js)
- **rxjs**
  Reactive Extensions Library for JavaScript, including an implementation of Observables which are returned by many Angular APIs and used throughout the Angular framework for handling asynchronous events.
  Docs: https://angular.io/guide/rx-library (https://angular.io/guide/rx-library)
- **zone.js**
  Implements *Zones* for JavaScript, used by Angular for running change detection processes when native js operations raise events.
  Docs: https://github.com/angular/zone.js/ (https://github.com/angular/zone.js/)

Install Angular 7 Development Dependencies

Run the following command from the project folder to install and save the npm packages required to develop, compile and bundle Angular 7 applications:

```
npm install --save-dev @types/node@11 angular2-template-loader@0 html-webpack-plugin@3 raw-loader@1 ts-loader@5 typescript@3 webpack@4
webpack-cli@3 webpack-dev-server@3
```

The command will download the packages into a `node_modules` folder and add them to the `devDependencies` property in the package.json file. So next time to install all of the packages you just need to run `npm install` and it will use the list in package.json.

The package.json should look like this after installing Angular 7 development dependencies with the above command:

```
{
    "name": "angular-7-tutorial",
    "version": "1.0.0",
    "dependencies": {
        "@angular/common": "^7.2.13",
        "@angular/compiler": "^7.2.13",
        "@angular/core": "^7.2.13",
        "@angular/forms": "^7.2.13",
        "@angular/platform-browser": "^7.2.13",
        "@angular/platform-browser-dynamic": "^7.2.13",
        "@angular/router": "^7.2.13",
        "core-js": "^3.0.1",
        "rxjs": "^6.4.0",
        "zone.js": "^0.9.0"
    },
    "devDependencies": {
        "@types/node": "^11.13.5",
        "angular2-template-loader": "^0.6.2",
        "html-webpack-plugin": "^3.2.0",
        "raw-loader": "^1.0.0",
        "ts-loader": "^5.3.3",
        "typescript": "^3.4.4",
        "webpack": "^4.30.0",
        "webpack-cli": "^3.3.0",
        "webpack-dev-server": "^3.3.1"
    }
}
```

Below are details of each of the dev dependencies and a link to the documentation for each:

- **@types/node**
  Contains TypeScript type definitions for Node.js.
  Docs: https://www.npmjs.com/package/@types/node (https://www.npmjs.com/package/@types/node)
- **angular2-template-loader**
  A webpack loader that replaces `templateUrl` and `styleUrls` declarations in Angular components with corresponding require statements in order to inline the html and styles. Depends on raw-loader for loading html and css files.
  Docs: https://github.com/TheLarkInn/angular2-template-loader (https://github.com/TheLarkInn/angular2-template-loader)
- **html-webpack-plugin**
  Injects webpack bundled JavaScript into an html template to be run in the browser.
  Docs: https://webpack.js.org/plugins/html-webpack-plugin/ (https://webpack.js.org/plugins/html-webpack-plugin/)
- **raw-loader**
  A webpack loader that enables importing text files such as html templates and css stylesheets. Used to load Angular html component templates and convert them into JavaScript code.
  Docs: https://www.npmjs.com/package/raw-loader (https://www.npmjs.com/package/raw-loader)
- **ts-loader**
  A webpack loader that loads TypeScript files and compiles/transpiles them into JavaScript using the command-line typescript compiler.
  Docs: https://www.npmjs.com/package/ts-loader (https://www.npmjs.com/package/ts-loader)
- **typescript**
  The command-line TypeScript compiler that converts TypeScript files into JavaScript.
  Docs: https://www.typescriptlang.org/docs/home.html (https://www.typescriptlang.org/docs/home.html)
- **webpack**
  A JavaScript module bundler that can be extended with loaders to handle other file types (e.g. TypeScript, HTML, CSS) and plugins to perform other tasks such as injecting generated JS bundles into an HTML template (html-webpack-plugin).
  Docs: https://webpack.js.org/concepts (https://webpack.js.org/concepts)
- **webpack-cli**
  The webpack command line interface, it is required to use webpack from the command line.
  Docs: https://webpack.js.org/api/cli/ (https://webpack.js.org/api/cli/)
- **webpack-dev-server**
  A local development server for running and testing webpack applications, it provides automatic live reloading when a file is updated.
  Docs: https://webpack.js.org/configuration/dev-server/ (https://webpack.js.org/configuration/dev-server/)

Create tsconfig.json

Inside the project folder create a file named `tsconfig.json`. This file specifies the compiler options used by the TypeScript compiler to convert TypeScript files into JavaScript that will be understood by browsers.

Open tsconfig.json in VS Code, set `target` to "es5", `emitDecoratorMetadata` to true and `experimentalDecorators` to true. There are many more compiler options available, for a full list see https://www.typescriptlang.org/docs/handbook/compiler-options.html (https://www.typescriptlang.org/docs/handbook/compiler-options.html).

The tsconfig.json file should look like this:

```
{
    "compilerOptions": {
        "emitDecoratorMetadata": true,
        "experimentalDecorators": true,
        "target": "ES5"
    }
}
```

Below are details of each of the compiler options used:

- **emitDecoratorMetadata**
  This options tells the TypeScript compiler to save type information from Angular components as metadata in the output JavaScript. The type information is used by Angular to support dependency injection.
  For a more detailed explanation: http://nicholasjohnson.com/blog/how-angular2-di-works-with-typescript/ (http://nicholasjohnson.com/blog/how-angular2-di-works-with-typescript/).
- **experimentalDecorators**
  Enables the experimental *Decorator* feature of TypeScript. A Decorator is a special kind of declaration that can be attached to a class, method, accessor, property, or parameter, and are written as `@Decorator`. Angular uses decorators to declare what a class is (e.g. `@Component`, `@NgModule`, `@Directive`) and provide configuration metadata about the class.
  List of all decorators in Angular core package: https://angular.io/api/core#decorators (https://angular.io/api/core#decorators).
- **target**
  Specifies the JavaScript/ECMAScript target version that the TypeScript will be compiled to. "ES5" is currently the latest version which is supported by all modern browsers.
  More info about browser support for JavaScript versions: https://www.w3schools.com/js/js_versions.asp (https://www.w3schools.com/js/js_versions.asp).

Create webpack.config.js

Inside the project folder create a file named `webpack.config.js`. This file specifies the configuration options used by webpack to compile and bundle an Angular app.

Webpack is a JavaScript module bundler that can be extended with loaders to handle other file types (e.g. TypeScript, HTML, CSS) and plugins to perform other tasks such as injecting generated JS bundles into an HTML template (html-webpack-plugin). More info available at the official webpack docs website: https://webpack.js.org/concepts (https://webpack.js.org/concepts).

Open webpack.config.js in VS Code and copy the following config:

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
    entry: './src/main.ts',
    resolve: {
        extensions: ['.ts', '.js']
    },
    module: {
        rules: [
            {
                test: /\.ts$/,
                use: ['ts-loader', 'angular2-template-loader']
            },
            {
                test: /\.(html|css)$/,
                use: 'raw-loader'
            }
        ]
    },
    plugins: [
        new HtmlWebpackPlugin({ template: './src/index.html' })
    ]
}
```

Below are details of each of the webpack config options used:

- **entry**
  Specifies the entry point/s for webpack to begin. Webpack will find all of the direct and indirect dependencies of the entrypoint file to include in the output JavaScript bundle/s for the Angular application. If not set the default value is `./src/index.js` , for this Angular app it is set to `./src/main.ts` .
  Docs: https://webpack.js.org/concepts/entry-points/ (https://webpack.js.org/concepts/entry-points/).
- **resolve.extensions**
  Sets the list of extensions that webpack will use when attempting to resolve the location of module dependencies. Used to find the files for imported Angular modules and components.
  Docs: https://webpack.js.org/configuration/resolve/#resolveextensions (https://webpack.js.org/configuration/resolve/#resolveextensions).
- **module.rules**
  Module rules are configured to tell webpack to transform specified files using loaders before adding them to the output bundle. The `test` property sets which file/s to transform and the `use` property sets which loader/s will do the transforming.
  The first above rule tells webpack to transform TypeScript (`*.ts`) files with the `ts-loader` and `angular2-template-loader` . The second rule tells webpack to transform HTML (`*.html`) & CSS (`*.css`) files with the `raw-loader` .
  Docs: https://webpack.js.org/concepts#loaders (https://webpack.js.org/concepts#loaders)
- **plugins**
  Plugins are used to extend the functionality of webpack to perform a wide range of tasks. The `HtmlWebpackPlugin` above injects the JavaScript bundle output from webpack into the specified html template (`./src/index.html`) to be run in the browser. Docs: https://webpack.js.org/concepts#plugins (https://webpack.js.org/concepts#plugins)

Create src folder

Create a `src` folder in the project folder, this folder will contain all of the source code for the Angular 7 application.

Create app folder

Create an `app` folder in the `src` folder, this folder will contain all of the code for the Angular 7 `AppModule` that you will create shortly.

Angular applications are organised into modules, or in other words, Angular modules are the building blocks of Angular applications. An Angular module is used to group together related Angular components, services, directives etc. The tutorial application will contain a single Angular module (`AppModule`), as applications grow they can be split into multiple modules for maintainability. For more info on Angular Modules see https://angular.io/guide/architecture-modules (https://angular.io/guide/architecture-modules)

Create Angular App Component template

Create a file named `app.component.html` in the `/src/app` folder and copy the below HTML code into it.

The App Component template contains the HTML for the App Component that you will create in the next step. For now it will just contain a simple hello message that will display when we test our Angular 7 app. For more info on Angular templates see https://angular.io/guide/architecture-components#templates-and-views (https://angular.io/guide/architecture-components#templates-and-views)

It should look like this:

```
<h1>Hello Angular 7!</h1>
```

Create Angular App Component

Create a file named `app.component.ts` in the `/src/app` folder and copy the below TypeScript code into it.

An Angular component is a class that contains the logic to control a piece of the UI. A class becomes an Angular component when it is decorated with the `@Component` decorator. For more info on Angular components see https://angular.io/guide/architecture-components (https://angular.io/guide/architecture-components)

The App Component contains the logic required to interact with and support the App Component template, at the moment the App Component template doesn't require any logic so the class is empty.

The `@Component` decorator contains two parameters, the `selector: 'app'` parameter tells Angular to inject an instance of this component wherever it finds the `<app></app>` HTML tag. The `templateUrl: 'app.component.html'` parameter tells Angular where to find the HTML template for this component.

It should look like this:

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

@Component({ selector: 'app', templateUrl: 'app.component.html' })
export class AppComponent {}
```

Create Angular App Module

Create a file named `app.module.ts` in the `/src/app` folder and copy the below TypeScript code into it.

An Angular module is a class used to group together related Angular components, services, directives etc. A class becomes an Angular module when it is decorated with the `@NgModule` decorator. For more info on Angular Modules see https://angular.io/guide/architecture-modules (https://angular.io/guide/architecture-modules).

It should look like this:

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

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

@NgModule({
    imports: [BrowserModule],
    declarations: [AppComponent],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Below are details of each of the `@NgModule` parameters used:

- The `imports: [BrowserModule]` parameter tells Angular to import the `BrowserModule` into this module, which makes the components, directives and pipes in `BrowserModule` available to all components in the `AppModule`. The `BrowserModule` includes core functionality for running Angular applications in different supported browsers.
- The `declarations: [AppComponent]` parameter declares that the `AppComponent` belongs to this module and makes it available to all other components and templates within this module.
- The `bootstrap: [AppComponent]` parameter tells Angular to bootstrap the `AppComponent` when the `AppModule` is bootstrapped. For more info on Angular bootstrapping see https://angular.io/guide/bootstrapping (https://angular.io/guide/bootstrapping).

Create Angular 7 Polyfills File

Create a file named `polyfills.ts` in the `/src` folder and copy the below TypeScript code into it.

Some features used by Angular 7 are not yet supported natively by all major browsers, polyfills are used to add support for features where necessary so your Angular 7 application works across all major browsers.

```
import 'core-js/features/reflect';
import 'zone.js/dist/zone';
```

Create Angular 7 Main (Bootstrap/Launch) File

Create a file named `main.ts` in the `/src` folder and copy the below TypeScript code into it.

The main file is the entry point used by Angular to bootstrap/launch the application's root module (`AppModule`) in the browser.

It calls `bootstrapModule(AppModule);` on `platformBrowserDynamic()` to compile and bootstrap `AppModule` in the browser with JIT compilation. For more info on Angular bootstrapping see https://angular.io/guide/bootstrapping (https://angular.io/guide/bootstrapping).

```
import './polyfills';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Create Angular 7 Main Index Html File

Create a file named `index.html` in the `/src` folder and copy the below HTML code into it.

The main index.html file is the initial page loaded by the browser that kicks everything off. Webpack bundles all of the javascript files together and injects them into the body of the index.html page so the scripts get loaded and executed by the browser.

Angular injects the `AppComponent` into the `<app>Loading...</app>` tag because it matches the `selector` defined in the component metadata. The `Loading...` message is the default text displayed until Angular has finished booting up.

```
<!DOCTYPE html>
<html>
<head>
    <base href="/" />
    <title>Angular 7 Tutorial</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <!-- bootstrap css -->
    <link href="//cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.3/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <app>Loading...</app>
</body>
</html>
```

Add start script to package.json

Open package.json in VS Code and the following start script below the version property:

```
"scripts": {
  "start": "webpack-dev-server --mode development --open"
}
```

The start script is run with the command `npm start`. This script will start the `webpack-dev-server` in development mode and automatically open the browser with the `--open` argument.

The updated package.json should now look like this:

```
{
    "name": "angular-7-tutorial",
    "version": "1.0.0",
    "scripts": {
        "start": "webpack-dev-server --mode development --open"
    },
    "dependencies": {
        "@angular/common": "^7.2.13",
        "@angular/compiler": "^7.2.13",
        "@angular/core": "^7.2.13",
        "@angular/forms": "^7.2.13",
        "@angular/platform-browser": "^7.2.13",
        "@angular/platform-browser-dynamic": "^7.2.13",
        "@angular/router": "^7.2.13",
        "core-js": "^3.0.1",
        "rxjs": "^6.4.0",
        "zone.js": "^0.9.0"
    },
    "devDependencies": {
        "@types/node": "^11.13.5",
        "angular2-template-loader": "^0.6.2",
        "html-webpack-plugin": "^3.2.0",
        "raw-loader": "^1.0.0",
        "ts-loader": "^5.3.3",
        "typescript": "^3.4.4",
        "webpack": "^4.30.0",
        "webpack-cli": "^3.3.0",
        "webpack-dev-server": "^3.3.1"
```

```
}
```

Start Angular 7 Application!

Run the command `npm start` from the project root folder (where the package.json is located) to launch the Angular 7 application.

Subscribe or Follow Me For Updates

Subscribe to my YouTube channel or follow me on Twitter or GitHub to be notified when I post new content.

- Subscribe on YouTube at https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg/ (https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg?sub_confirmation=1)
- Follow me on Twitter at https://twitter.com/jason_watmore (https://twitter.com/jason_watmore)
- Follow me on GitHub at https://github.com/cornflourblue (https://github.com/cornflourblue)

Tags: Angular 7 (/posts/tag/angular-7), Angular 2 (/posts/tag/angular-2), TypeScript (/posts/tag/typescript), Webpack (/posts/tag/webpack)

Share:

More Angular 7 Posts

- Angular 7 Tutorial Part 6 - Home Page & Alert Component (/post/2019/05/31/angular-7-tutorial-part-6-home-page-alert-component)
- Angular 7 Tutorial Part 5 - Registration Form & User Service (/post/2019/05/22/angular-7-tutorial-part-5-registration-form-user-service)
- Angular 7 Tutorial Part 4 - Login Form, Authentication Service & Route Guard (/post/2019/05/17/angular-7-tutorial-part-4-login-form-authentication-service-route-guard)
- Angular 7 - Mock Backend Example for Backendless Development (/post/2019/05/02/angular-7-mock-backend-example-for-backendless-development)
- Angular 7 Tutorial Part 3 - Add Routing & Multiple Pages (/post/2019/04/29/angular-7-tutorial-part-3-add-routing-multiple-pages)
- Angular 7 - Custom Modal Window / Dialog Box (/post/2019/04/16/angular-7-custom-modal-window-dialog-box)
- Angular 7 - Communicating Between Components with Observable & Subject (/post/2019/02/07/angular-7-communicating-between-components-with-observable-subject)
- Angular 7 - Role Based Authorization Tutorial with Example (/post/2018/11/22/angular-7-role-based-authorization-tutorial-with-example)
- Angular 7 - JWT Authentication Example & Tutorial (/post/2018/11/16/angular-7-jwt-authentication-example-tutorial)
- Angular 7 - Template-Driven Forms Validation Example (/post/2018/11/10/angular-7-template-driven-forms-validation-example)
- Angular 7 - Reactive Forms Validation Example (/post/2018/11/07/angular-7-reactive-forms-validation-example)
- Angular 7 - User Registration and Login Example & Tutorial (/post/2018/10/29/angular-7-user-registration-and-login-example-tutorial)

What do you think?

6 Responses

Upvote Funny Love Surprised Angry Sad

2 Comments **Jason Watmore's Blog** 1 **Login**

Recommend Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Brian • a month ago

Any reason you're not using the Angular CLI? This will be very difficult to automatically upgrade/maintain

• Reply • Share ›

Jason Watmore Mod Brian • 24 days ago

Hi Brian, I like using webpack directly so the project is lean and only contains the bits it needs. I've found it pretty easy to upgrade to different versions of Angular with this approach, I maintain a number of example projects that I've been upgrading since Angular 2 (e.g. https://jasonwatmore.com/post/2018/10/29/angular-7-user-registration-and-login-example-tutorial).

If you prefer using the Angular CLI it's not hard to migrate the code. I've created a few Angular CLI versions of example projects and pretty much all I had to do was copy the source code into a new Angular CLI project and it worked. I'll be showing how to do this in the last video of the series.

Cheers,
Jason

• Reply • Share ›

ALSO ON **JASON WATMORE'S BLOG**

React - Basic HTTP Authentication Tutorial & Example | Jason Watmore's Blog

4 comments • 9 months ago

Avatar Sasmit Das — how can I deploy in a clod instance .. What changes I need to make

Angular 6 - Basic HTTP Authentication Tutorial & Example | Jason Watmore's Blog

11 comments • 9 months ago

Avatar Guigreg — Hi! Are you sure that you want to store the username and password as plain text in the local storage?

Angular 7 - JWT Authentication Example & Tutorial | Jason Watmore's Blog

39 comments • 7 months ago

Avatar Bilal Elmursi — Wow very nice tutorial

ASP.NET Core 2.1 - Basic Authentication Tutorial with Example API | Jason …

16 comments • 9 months ago

Avatar Jason Watmore — https://en.wikipedia.org/wiki/Basic_authentication

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Disqus' Privacy PolicyPrivacy PolicyPrivacy**

ABOUT

I'm a web developer in Sydney Australia and the technical lead at Point Blank Development (https://www.pointblankdevelopment.com.au), I've been building websites and web applications in Sydney since 1998.

Find me on: (https://twitter.com/jason_watmore) (https://github.com/cornflourblue) (https://www.youtube.com/channel/UCc46Wo9z8S3xSDhw9vdvxtg/)

Support me on Patreon (https://www.patreon.com/jasonwatmore)

MONTHS

2019

- May (/posts/2019/05) (4)
- April (/posts/2019/04) (6)
- March (/posts/2019/03) (1)
- February (/posts/2019/02) (4)
- January (/posts/2019/01) (1)

2018
2017
2016
2015
2014
2013
2012
2011

TAGS

Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular Directive, Angular UI Router, AngularJS, Animation, ASP.NET, ASP.NET Core, ASP.NET Web API, Authentication and Authorization, AWS, Basic Authentication, Bootstrap, C#, Chai, CKEditor, CSS3, DDD, Design Patterns, Dynamic LINQ, ELMAH, ES6, Exchange, Facebook, Fluent NHibernate, Google Analytics, Google API, Google Maps API, Google Plus, Heroku, HTML5, HTTP, IIS, Insecure Content, Instagram API, Ionic Framework, iOS, iPhone, JavaScript, jQuery, JWT, LinkedIn, LINQ, Login, MEAN Stack, Mocha, Modal, MongoDB, Moq, MVC, MVC5, NGINX, ngMock, NHibernate, Ninject, NodeJS, npm, Pagination, Pinterest, Razor Pages, React, Redmine, Redux, Registration, Repository, RxJS, Security, Shell Scripting, Sinon, SinonJS, TDD, Terraform, Twitter, TypeScript, Ubuntu, Umbraco, Unit of Work, Unit Testing, URL Rewrite, Validation, Vue, Vuex, Webpack, Windows Server 2008,