Tutorial 1 - QUICKSTART

Overview

The QuickStart application has the structure of a real-world Angular application and displays the simple message:



**Build this application!**

* [Prerequisite](https://angular.io/docs/ts/latest/quickstart.html#prereq): Install Node.js and npm.
* [Step 1](https://angular.io/docs/ts/latest/quickstart.html#create-and-configure): Create and configure the project.
* [Step 2](https://angular.io/docs/ts/latest/quickstart.html#ngmodule): Create your application.
* [Step 3](https://angular.io/docs/ts/latest/quickstart.html#root-component): Create a component and add it to your application.
* [Step 4](https://angular.io/docs/ts/latest/quickstart.html#main): Start up your application.
* [Step 5](https://angular.io/docs/ts/latest/quickstart.html#index): Define the web page that hosts the application.
* [Step 6](https://angular.io/docs/ts/latest/quickstart.html#build-and-run): Build and run the application.
* [Step 7](https://angular.io/docs/ts/latest/quickstart.html#make-changes): Make some live changes.
* [Wrap up and Next Steps](https://angular.io/docs/ts/latest/quickstart.html#wrap-up)

# **Prerequisite: Install Node.js and npm**

**Verify that you are running at least node v4.x.x and npm 3.x.x** by running node -v and npm -v in a terminal/console window. Older versions produce errors.

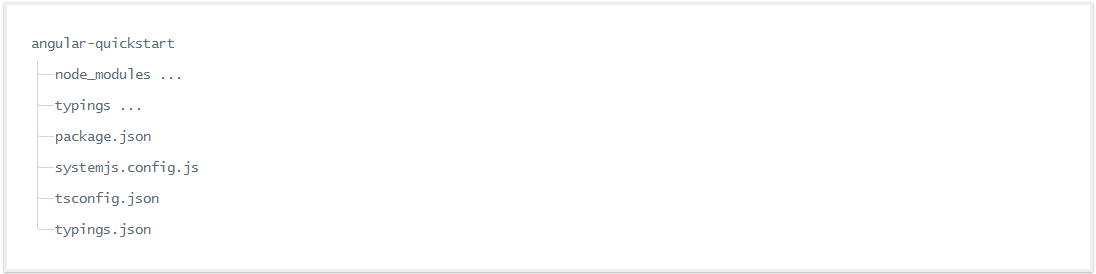
Note: To upgrade **npm on Windows**, go to this link and see Usage section

<https://github.com/felixrieseberg/npm-windows-upgrade>

# **Step 1: Create and configure the project**

* Make a copy of angular2-seed
* Rename the copy to angular-quickstart
* cd angular-quickstart
* npm install

You should now have the following structure:



If the typings folder doesn't show up after running npm install, you'll need to install it manually with the command:

npm run typings install

# **Step 2: Create your application**

You compose Angular applications into closely related blocks of functionality with [NgModules](https://angular.io/docs/ts/latest/guide/ngmodule.html). Angular itself is split into separate Angular Modules. This makes it possible for you to keep payload size small by only importing the parts of Angular that your application needs.

Every Angular application has at least one module: the root module, named AppModule here.

* Create an app subfolder off the project root directory
* Create file app/app.module.ts with content:



This is the entry point to your application.

Since the QuickStart application is a web application that runs in a browser, your root module needs to import the [BrowserModule](https://angular.io/docs/ts/latest/api/platform-browser/index/BrowserModule-class.html) from @angular/platform-browser to the imports array.

This is the smallest amount of Angular that is needed for a minimal application to run in the browser.

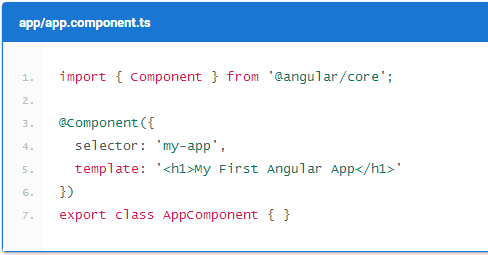
The QuickStart application doesn't do anything else, so you don't need any other modules. In a real application, you'd likely import [FormsModule](https://angular.io/docs/ts/latest/api/forms/index/FormsModule-class.html) as well as [RouterModule](https://angular.io/docs/ts/latest/api/router/index/RouterModule-class.html) and [HttpModule](https://angular.io/docs/ts/latest/api/http/index/HttpModule-class.html). These are introduced in next tutorial.

# **Step 3: Create a component and add it to your application**

Every Angular application has at least one component: the *root component*, named AppComponent here.

Components are the basic building blocks of Angular applications. A component controls a portion of the screen—a *view*—through its associated template.

Create the component file app/app.component.ts with the following content:



The QuickStart application has the same essential structure as any other Angular component:

* **An import statement**. Importing gives your component access to Angular's core [@Component decorator function](https://angular.io/docs/ts/latest/api/core/index/Component-decorator.html).
* **A @Component decorator** that associates metadata with the AppComponent component class:
* a selector that specifies a simple CSS selector for an HTML element that represents the component.
* a template that tells Angular how to render the component's view.
* **A component class** that controls the appearance and behavior of a view through its template. Here, you only have the root component AppComponent. Since you don't need any application logic in the simple QuickStart example, it's empty.

You export the AppComponent class so that you can import it into the application that you just created.

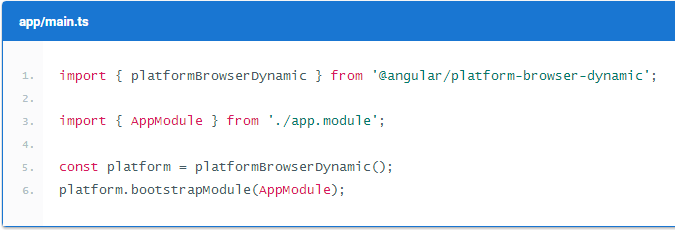
Edit the file app/app.module.ts to import your new AppComponent and add it in the declarations and bootstrap fields in the NgModule decorator:



# **Step 4: Start up your application**

Now you need to tell Angular to start up your application.

Create the file app/main.ts with the following content:



This code initializes the platform that your application runs in, then uses the platform to bootstrap your AppModule.

### **Why create separate *main.ts*, app module and app component files?**

App bootstrapping is a separate concern from creating a module or presenting a view. Testing the component is much easier if it doesn't also try to run the entire application.

# **Step 5: Define the web page that hosts the application**

* Copy index.html from the **Files** folder to your project root folder

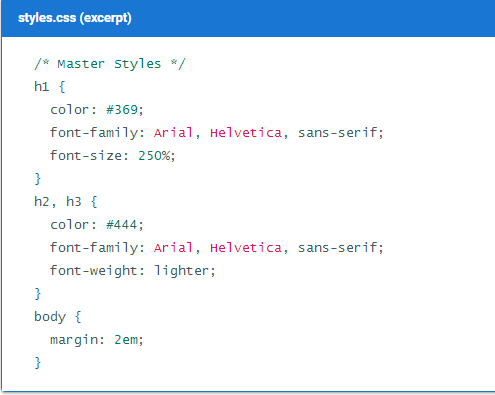
The noteworthy sections here are:

* JavaScript libraries: core-js polyfills for older browsers, the zone.js and reflect-metadata libraries needed by Angular, and the SystemJS library for module loading.
* Configuration file for SystemJS, and a script where you import and run the app module which refers to the main file that you just wrote.
* The [<my-app>](https://angular.io/docs/ts/latest/quickstart.html#my-app) tag in the <body> which is where your app lives!

## Add some style

Styles aren't essential, but they're nice, and index.html assumes that you have a stylesheet called styles.css.

Create a styles.css file in the *project root* folder, and start styling, perhaps with the minimal styles shown below. (You can also copy over this file from **Files** folder provided)



# **Step 6: Build and run the application**

Open a terminal window and enter this command:

npm start

That command runs the following two parallel node processes:

* The TypeScript compiler in watch mode.
* A static file server called lite-server that loads index.html in a browser and refreshes the browser when application files change.

In a few moments, a browser tab should open and display the following:

Output of QuickStart app

# **Step 7: Make some live changes**

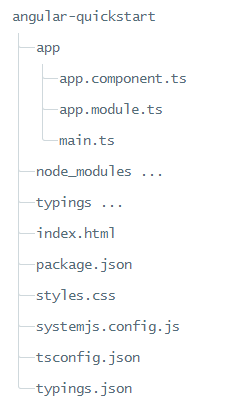
Try changing the message in app/app.component.ts to "Hello World! It’s my SECOND Angular App".

The TypeScript compiler and lite-server will detect your change, recompile the TypeScript into JavaScript, refresh the browser, and display your revised message.

Close the terminal window when you're done to terminate both the compiler and the server.

**Wrap up and next steps**

The final project folder structure looks like this:



**What next?**

This first application doesn't do much. It's basically "Hello, World" for Angular.

You wrote a little Angular component, created a simple index.html, and launched with a static file server.

You also created the basic application setup that you'll re-use for other sections in this guide. From here, the changes you'll make in the package.json or index.html files are only minor updates to add libraries or some css stylesheets. You also won't need to revisit module loading again.

To take the next step and build a small application that demonstrates real features that you can build with Angular, carry on to the [next tutorial](https://angular.io/docs/ts/latest/tutorial).

|  |
| --- |
| Why Angular2 ? - Its advantages |
| Angular 2 Architecture |
| ECMA 6 and Typescript |
| **ECMA 6 Features -** |
| Classes |
| Arrow Function |
| Template String |
| Inheritance |
| Delegation |
| Constants and Block Scoped Variables |
| Spread Syntax (Spread Element) and Rest parameters |
| Destructuring |
| Modules |
| **TypeScript Features -** |
| Typings |
| Linting |
| TypeScript Features |
| TypeScript Classes |
| Interfaces |
| Shapes |
| Type Inference |
| Type Keyword |
| Decorators |
| Property Decorators |
| Class Decorators |
| Parameter Decorators |
|  |
| Modules - NgModule |
| Components |
| Component Lifecycle / Lifecycle Hooks |
| Passing Data into a Component |
| Responding to Component Events |
| Projection in Component |
| Using other component inside a component |
| Animation |
| Directives |
| Structural Directives |
| Data Binding |
| Metadata |
| Template |
| Forms |
| Services |
| Pipes |
| Dependency Injection |
| HTTP |
| Promise |
| Observables |
| Observables vs Promises |
| Component Routing (ngComponentRouter) |
| RouteParams |
| Lazy Loading |
| Change Dection |
| Zones |
| Unit Testing and E2E Testing |
| Responsive Design |
| Theming |
| Localization |
| Internationalization |
| Accessibility |
| Angular CLI/Webpack? |
| Aot and JIT Compiler |