# MISC

## angular and node.js, npm

Node.js or Node is an open-source, cross-platform, **JavaScript runtime environment(JSRE)** that executes JavaScript code outside of a web browser.

npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. Nvm is a Nodejs version manager. It lets you easily install and switch between versions. It retains globally installed packages for each version.

Why does Angular need Node.js and npm? Angular is written in TypeScript, which browsers don’t understand and hence TypeScript needs to be transpiled to JavaScript. Node.js compile Javascript, not TypeScript, but Node.js can be used with other packages to transpile TypeScript to Javascript; Angular also needs many Javascript packages, so Angular needs npm, the package manager of node.js.

## Differences between AngularJS and Angular

AngularJS was released 6 years before Angular, in 2010 by Google. ReactJS and VueJS arrived on the market and made clear they wanted to compete for the best performance framework. This is when Google’s team realized and decided it was time to rewrite Angular completely. The release of Angular 2 revealed some drastic changes. Language, Architecture and Data binding were the key ones.

### Language: JavaScripts vs. TypeScript

Angular team’s first decision was to shift from JavaScript to TypeScript to avoid JavaScript’s pitfalls. Also, a way of presenting a more organized and maintainable code. The static typing of TypeScript improves performance, avoids runtime pitfalls and transpiles to JavaScript.

### Architecture: MVC vs. Component-based architecture

AngularJS’s architecture supports the MVC (Model View Controller) architecture. Logic, output and view happen in separate files. On the other hand, Angular relies on components, directives with predefined templates. Such a modern structure allows us to scale and maintain large applications.

### Data Binding: Two-way data binding algorithm

AngularJS two-way data binding is a unique way that immediately propagates to the view of any data changes that can affect the model, and changes made by users in views immediately reflect the model. A better Angular algorithm makes it five times faster than AngularJS for data binding and component-based architecture.

## Quick introduction to NPM

### What is npm?

npm is 2 things:

* Npm is a software repository for JavaScript
* npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js.

### install npm

People often install npm through the Node.js installer, which includes both node.js and npm.

### How to use npm

npm manages downloads of dependencies of your project.

#### package.json

All npm packages contain a file, usually in the project root, called **package.json** - this file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies. It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package.

npm installs every dependency into the **./node\_modules** directory, so in order to uninstall every package a given repo depends on, just delete all in the **./node\_modules** directory. For example, if your package.json looks like the below, **npm install** will install version 5.2.10 of Mongoose and version 4.16.4 of Express.

{

"dependencies": {

"mongoose": "5.2.10",

"express": "4.16.4"

}

}

Every repo has a separate copy of every dependency. If you clone two repos that depend on express@4.16.4, you will have two copies of the same version of Express.

#### Installing all dependencies described in packages.json

If a project has a packages.json file, by running

npm install

it will install everything the project needs, in the node\_modules folder, creating it if it’s not existing already.

#### Installing a single package

You can also install a specific package by running

npm install <package-name>

Often you’ll see more flags added to this command:

--save installs and adds the entry to the package.json file dependencies (default as of npm 5)

--save-dev installs and adds the entry to the package.json file devDependencies

The difference is mainly that devDependencies are usually development tools, like a testing library, while dependencies are bundled with the app in production.

#### Versioning

In addition to plain downloads, npm also manages versioning, so you can specify any specific version of a package, or require a version higher or lower than what you need.

Many times you’ll find that a library is only compatible with a major release of another library.

Or a bug in the latest release of a lib, still unfixed, is causing an issue.

Specifying an explicit version of a library also helps to keep everyone on the same exact version of a package, so that the whole team runs the same version until the package.json file is updated.

In all those cases, versioning helps a lot, and npm follows the semantic versioning (semver) standard.

#### Running Tasks

The package.json file supports a format for specifying command line tasks that can be run by using

npm run <task-name>

For example: a package.json has:

{

"scripts": {

"start-dev": "node lib/server-development",

"start": "node lib/server-production"

},

}

It’s very common to use this feature to run Webpack:

{

"scripts": {

"watch": "webpack --watch --progress --colors --config webpack.conf.js",

"dev": "webpack --progress --colors --config webpack.conf.js",

"prod": "NODE\_ENV=production webpack -p --config webpack.conf.js",

},

}

So instead of typing those long commands, which are easy to forget or mistype, you can run:

$ npm run watch

$ npm run dev

$ npm run prod

# Angular workspace

## Setting up the local environment and workspace

To install Angular on your local system and set up a workspace, you need the following:

1. Node.js and npm

Angular uses Typescript while browsers understand only Javascript, so Angular needs Node.js plus some other packages to transpile Typescript to Javascript.

To install both node.js and npm, use an installer from node.js website.

1. Angular CLI: npm install -g @angular/cli

You use the Angular CLI to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

To install the Angular CLI, open a terminal window and run the following command:

npm install -g @angular/cli

1. Create a workspace and initial application: ng new my\_app

You develop apps in the context of an Angular workspace.

To create a new workspace and initial starter app: run the CLI command ng new and provide the name my-app, and then accept the default setup.

ng new my-app

1. Run the application: ng serve --open

The Angular CLI includes a server, so that you can build and serve your app locally.

To run your application, navigate to the workspace folder, such as my-app and then ng serve --open.

cd my-app

ng serve --open

The ng serve command launches the server, watches your files, and rebuilds the app as you make changes to those files.

The --open (or just -o) option automatically opens your browser to http://localhost:4200/.

If your installation and setup was successful, you should see a page similar to the following.

## Angular project structure

The followings are what inside an Angular project folder (generated by ng new):

🗂e2e/

End to end testing folder located at the top level of the project’s structure. It contains source files to perform end-to-end tests to the root-level application (AppComponent).

You can use ng e2e command on the CLI to test. The app will be built in watch mode and the Protractor test runner will be launched. Once that is done, you will see the output in the terminal and the chrome browser will open up, test all the scenarios declared in the e2e file and quickly close.

🗂src

📝Index.html — The main HTML page where the views are rendered and are served when users visit your page.

📝main.ts — The main.ts is the main entry point of an Angular application. It uses the JIT compiler to compile the application and bootstraps the application’s root module (AppModule) to run in the browser.

📝polyfills.css — Provides polyfill scripts for browser support.

📝styles.css — All the styles declared here will be globally available. The extension of the file reflects the style preprocessor you have configured for the project (in this case CSS).

🗂app/

It contains the component files in which your application logic and data are defined.

📝 App.component.css

A stylesheet automatically generated to declare styles for the root AppComponent. The extension of the file reflects the style preprocessor you have configured for the project(in this case CSS).

📝App.component.html

A template generated to define the HTML root of the AppComponent.

📝App.component.spec.ts

The file responsible for performing *unit tests* for the root AppComponent.

When you run the ng test command on the CLI, the app will be built in watch mode and the Karma test runner will be launched. The test’s output is revealed in the terminal and a Jasmine HTML Reporter will be shown on a chrome browser which will open up.

📝App.component.ts

Defines the logic for the app’s root component (AppComponent). If you add more components, the app.component.html becomes the root of the views hierarchy.

📝App.module.ts

Defines the root module (AppModule) that tells Angular how to assemble the application. It declares AppComponent by default but can declare more components as you add them to the project.

🗂assets/

A folder automatically generated to store images and other assets files to be copied as you wish to add to your project.

🗂environments/

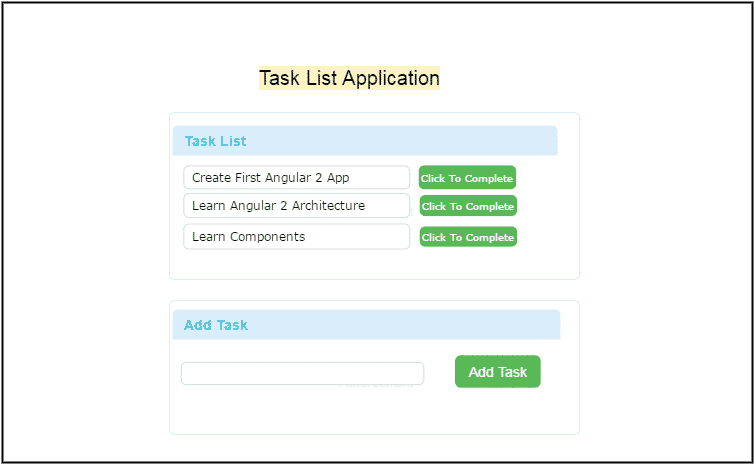
Contains build configuration (JSON configuration information) options for particular target environments. It is responsible for telling the build system which files to change when you run ng build or ng serve.

# Angular building blocks

## Angular Architecture

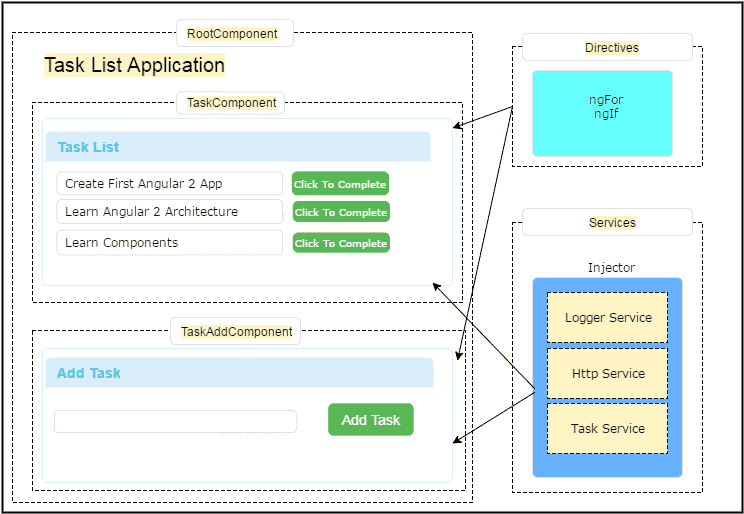
The  Architecture of an Angular Application is based on the idea of Components.  An Angular application starts with a Top-level component called Root Component. We then add child components forming a tree of loosely coupled components.

**A Typical Angular Application**



Consider a Simple Task List Application shown above. The Application shows a list of Tasks and at the bottom, you have the option to add a new task.

The above application a simple application, but if you look at the Angular code, you will see that it comprises several parts. You can see that in the image below. You can see that is consists of three components, Services that gets injected into the components and directives that help to manipulate the DOM



Our application has three Components. At the top we have rootComponent. Under the rootComponent, we have two other components. One is TaskComponent, which displays the list of Tasks and TaskAddComponent, which allows us to create new tasks.

The [Angular Components](https://www.tektutorialshub.com/angular/angular-components/) are plain javaScript classes and defined using [**@component Decorator**.](https://www.tektutorialshub.com/angular/angular-components/#Component-decorator) This Decorator provides the component with the View to display ( known as [**Template**](https://www.tektutorialshub.com/angular/angular-components/#Template-View)) & [**Metadata**](https://www.tektutorialshub.com/angular/angular-components/#Metadata)about the class

The Components uses [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/) to get the data from the Component to our View (Template). We use the special HTML markup knows as the [Angular Template Syntax](https://www.tektutorialshub.com/angular/angular-data-binding/#Template-Expression) to achieve this.

At the right side, we have an [Angular Services](https://www.tektutorialshub.com/angular/angular-services/). The Angular Services provides services to our Component, like fetching data from database using Task Service, logging application events using logger Services and making an HTTP request to the back-end server using HTTP service.

The Responsibility to provide the instance of the Service to the Components falls on [**Angular Injector**](https://www.tektutorialshub.com/angular/angular-injector-injectable-inject/). It injects services into the components using [**Dependency Injection**](https://www.tektutorialshub.com/angular/angular-dependency-injection/)

We also have [**Directives**](https://www.tektutorialshub.com/angular/angular-directives/), which help us to manipulate the structure ([**structural directives**](https://www.tektutorialshub.com/angular/angular-directives/#Structural-Directives)) or style ([**attribute directive**](https://www.tektutorialshub.com/angular/angular-directives/#Attribute-Directives)) our application. The directives help us to transform the DOM as per our needs.

## Angular component

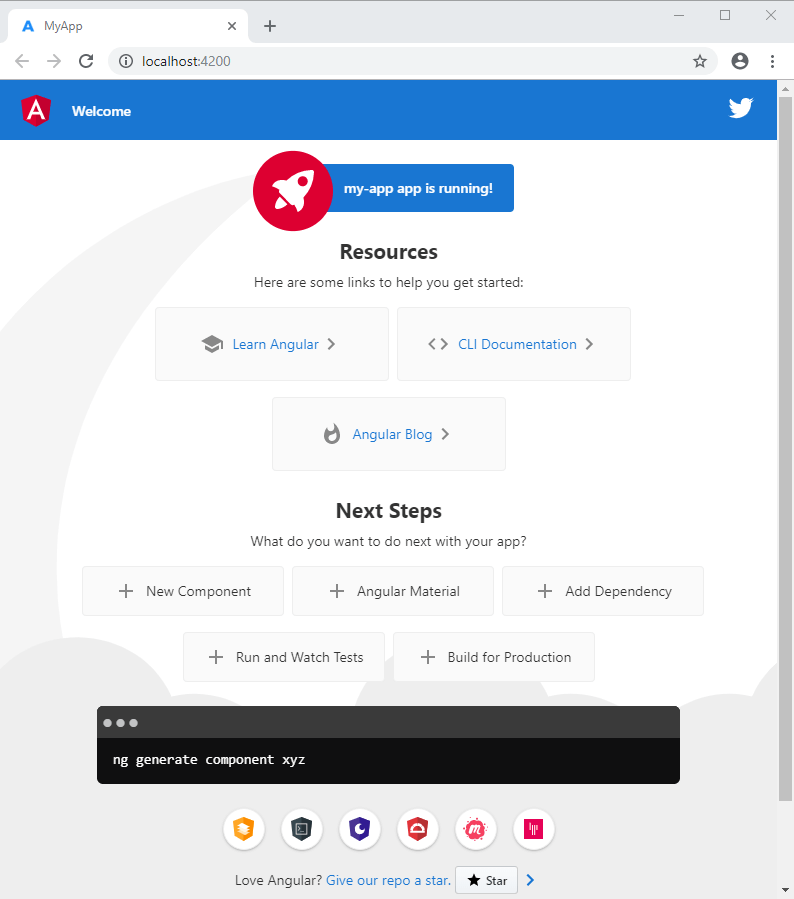
## Angular component

You build Angular applications with components. Components define areas of responsibility in the UI that let you reuse sets of UI functionality.

A component consists of three things: class, template, styles

* A component **class** that handles data and functionality.
* An HTML **template** that determines the UI.
* Component-specific **styles** that define the look and feel.

Angular project structure



Next steps

content\_copy

ng new my-app

The ng new command prompts you for information about features to include in the initial app. Accept the defaults by pressing the Enter or Return key.

The Angular CLI installs the necessary Angular npm packages and other dependencies. This can take a few minutes.

The CLI creates a new workspace and a simple Welcome app, ready to run.