Npm Packages

The **Angular CLI**, Angular applications, and Angular itself depend upon features and functionality provided by libraries that are available as **npm** packages.

You can download and install these npm packages with the **npm client**, which runs as a node.js application.

The **yarn client** is a popular alternative for downloading and installing npm packages. The Angular CLI uses `yarn` by default to install npm packages when you create a new project.

Node.js and npm are essential to Angular development. [Get them now](https://docs.npmjs.com/getting-started/installing-node "Installing Node.js and updating npm") if they're not already installed on your machine. **Verify that you are running node `v4.x.x` or higher and npm `3.x.x` or higher** by running the commands `node -v` and `npm -v` in a terminal/console window. Older versions produce errors. Consider using [nvm](https://github.com/creationix/nvm) for managing multiple versions of node and npm. You may need [nvm](https://github.com/creationix/nvm) if you already have projects running on your machine that use other versions of node and npm.

package.json

Both `npm` and `yarn` install packages identified in a **package.json** file.

The CLI `ng new` command creates a default `package.json` file for your project. This `package.json` specifies *a starter set of packages* that work well together and jointly support many common application scenarios.

You will add packages to `package.json` as your application evolves. You may even remove some.

This guide focuses on the most important packages in the starter set.

dependencies and *devDependencies*

The `package.json` includes two sets of packages, dependencies and devDependencies.

The *dependencies* are essential to *running* the application. The *devDependencies* are only necessary to *develop* the application.

{@a dependencies}

Dependencies

The `dependencies` section of `package.json` contains:

- **Angular packages**: Angular core and optional modules; their package names begin `@angular/`.
- **Support packages**: 3rd party libraries that must be present for Angular apps to run.
- **Polyfill packages**: Polyfills plug gaps in a browser's JavaScript implementation.

Angular Packages

@angular/animations: Angular's animations library makes it easy to define and apply animation effects such as page and list transitions. Read about it in the Animations guide.

@angular/common: The commonly needed services, pipes, and directives provided by the Angular team. The `HttpClientModule` is also here, in the '@angular/common/http' subfolder.

@angular/core: Critical runtime parts of the framework needed by every application. Includes all metadata decorators, `Component`, `Directive`, dependency injection, and the component lifecycle hooks.

@angular/compiler: Angular's *Template Compiler*. It understands templates and can convert them to code that makes the application run and render. Typically you don't interact with the compiler directly; rather, you use it indirectly via `platform-browser-dynamic` when JIT compiling in the browser.

@angular/forms: support for both template-driven and reactive forms.

@angular/http: Angular's old, soon-to-be-deprecated, HTTP client.

@angular/platform-browser: Everything DOM and browser related, especially the pieces that help render into the DOM. This package also includes the `bootstrapStatic()` method for bootstrapping applications for production builds that pre-compile with AOT.

@angular/platform-browser-dynamic: Includes Providers and methods to compile and run the app on the client using the JIT compiler.

@angular/router: The router module navigates among your app pages when the browser URL changes.

@angular/upgrade: Set of utilities for upgrading AngularJS applications to Angular.

{@a polyfills}

Polyfill packages

Many browsers lack native support for some features in the latest HTML standards, features that Angular requires. "Polyfills" can emulate the missing features. The Browser Support guide explains which browsers need polyfills and how you can add them.

The default `package.json` installs the **core-js** package which polyfills missing features for several popular browser.

Support packages

rxjs: Many Angular APIs return *observables*. RxJS is an implementation of the proposed Observables specification currently before the TC39 committee that determines standards for the JavaScript language.

zone.js: Angular relies on zone.js to run Angular's change detection processes when native JavaScript operations raise events. Zone.js is an implementation of a specification currently before the TC39 committee that determines standards for the JavaScript language.

{@a dev-dependencies}

DevDependencies

The packages listed in the *devDependencies* section of the `package.json` help you develop the application on your local machine.

You don't deploy them with the production application although there is no harm in doing so.

@angular/cli: The Angular CLI tools.

@angular/compiler-cli: The Angular compiler, which is invoked by the Angular CLI's `build` and `serve` commands.

@angular/language-service: The Angular language service analyzes component templates and provides type and error information that TypeScript-aware editors can use to improve the developer's experience. For example, see the Angular language service extension for VS Code

**@types/... **: TypeScript definition files for 3rd party libraries such as Jasmine and node.

codelyzer: A linter for Angular apps whose rules conform to the Angular style guide.

**jasmine/... **: packages to support the Jasmine test library.

**karma/... **: packages to support the karma test runner.

protractor: an end-to-end (e2e) framework for Angular apps. Built on top of WebDriverJS.

ts-node: TypeScript execution environment and REPL for node.

tslint: a static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors.

typescript: the TypeScript language server, including the *tsc* TypeScript compiler.

So many packages! So many files!

The default `package.json` installs more packages than you'll need for your project.

A given package may contain tens, hundreds, even thousands of files, all of them in your local machine's `node_modules` directory. The sheer volume of files is intimidating,

You can remove packages that you don't need but how can you be sure that you won't need it? As a practical matter, it's better to install a package you don't need than worry about it. Extra packages and package files on your local development machine are harmless.

By default the Angular CLI build process bundles into a single file just the few "vendor" library files that your application actually needs. The browser downloads this bundle, not the original package files.

See the Deployment to learn more.