Angular

1. Introduction:

What is Angular?

Angular is a development platform, built on TypeScript created by Google for developing web applications.

Angular includes:

- A component-based framework for building scalable web applications.
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication and more.
- A suite of developer tools to help develop, build, test, and update the code.

Features of Angular

- Angular has a good structure to create and maintain for large applications.
- Angular support for TypeScript and JavaScript.
- Angular comes with the Angular CLI tool.
- Angular uses modules, components and directives.
- Angular is supported by all the popular mobile browsers.
- Angular uses @Route Config{(…)} for routing configuration.

Why we prefer Angular?

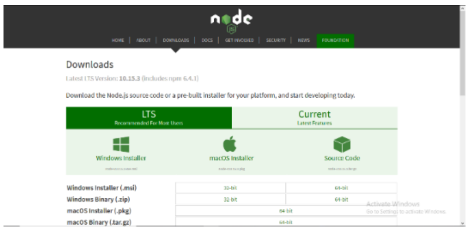
- Angular is particularly well-suited for building SPAs, so single webpage is created and content is dynamically updated as the user interacts.
- Angular has modular architecture, dependency injection, and TypeScript support make it suitable for building complex and maintainable applications.
- Angular has two-way data binding and dynamic features make it suitable for building interactive e-commerce websites with real-time updates and user interactions.
- With frameworks like Ionic or NativeScript, Angular is used to build cross-platform mobile applications for iOS and Android operating systems.
- Angular is used widely for building social media applications with dynamic content rendering, and interactive features.
- Angular has a big community support. It is developed and maintained by Google. Also, it is based on typescript which is created by Microsoft.
- Since it is an open-source language, therefore developers are allowed to use its components, all methods and functionality for free.
- Angular has a big community and many resources are available to learn.
- Angular CLI helps in simplification processes like creating, building, and deploying the application.
- Angular is used to build dynamic mobile-based applications.

2. Installations:

In order to work with Angular, we must have node.js, Node Package Manager (NPM) & Angular CLI installed in the system.

Installing Node On Windows (WINDOWS 10):

Step 1: Downloa the Node.js '.msi' installer. Visit the official Node.js website i.e https://nodejs.org/en/download/



Downloads

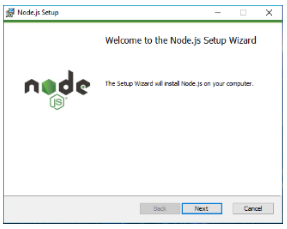
Latest LTS Version: 10.15.3 (includes npm 6.4.1)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	

Step-2: Run the Node.js installer. Double click on the .msi installer.

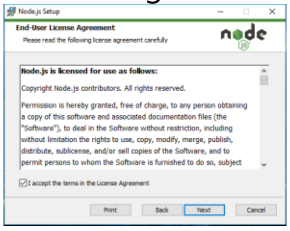
The Node.js Setup wizard will open. Select "Next"



Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

After clicking "Next", End-User License Agreement (EULA) will open. Check "I accept the terms in the License Agreement". Select "Next"



End-User License Agreement

Please read the following license agreement carefully

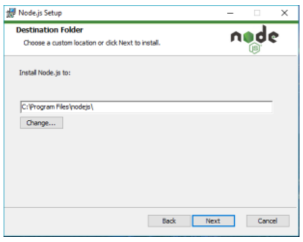
Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

☑ I accept the terms in the License Agreement

Set the Destination Folder where you want to install Node.js & Select "Next"



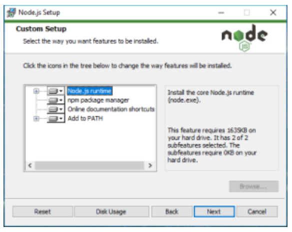
Destination Folder

Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\

Without changing in custom setup Select "Next"



Custom Setup

Select the way you want features to be installed.

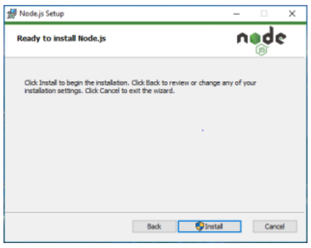
Click the icons in the tree below to change the way features will be installed.

Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH

Install the core Node.js runtime (node.exe).

This feature requires 1639KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 0KB on your hard drive.

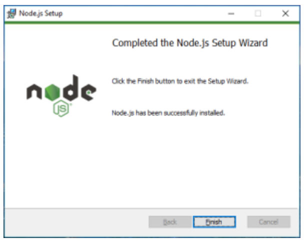
The installer may prompt you to "install tools for native modules". Select "Install"



Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Do not close or cancel the installer until the install is complete.
Complete the Node.js Setup Wizard. Click "Finish"



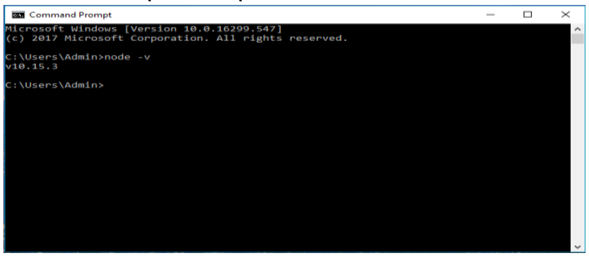
Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

Step 3: Verify that Node.js was properly installed or not.
To check that node.js was completely installed on your system or not, you can run the following command in command prompt or Windows Powershell and test it.

C:\Users\Admin> node -v



```
Microsoft Windows [Version 10.0.16299.547]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>node -v
v10.15.3

C:\Users\Admin>
```

If node.js was completely installed on your system, the command prompt will print the version of the Node JS installed.
Step 4: Updating the Local npm version.
You can run the following command, to quickly update the npm

npm install npm --global // Updates the 'CLI' client

Angular CLI Installation:
After installation of the node.js & npm, we need to install the Angular CLI, as follows:

Open the terminal/Command prompt & type the below command

npm install -g @angular/cli

For checking the version of the angular CLI installed, type the below command in terminal/Command prompt.

ng version

3. Create & Run First Angular Application:

Create the application:

Now, we will create the Angular application using the Angular CLI.

You may navigate to any of the directories where you want to create the project.

Use Angular CLI command as given below:

```
ng new Project-Name
```

it will ask about routing feature as below give either y or n.

```
? would you like to add Angular routing? (Y/N)
```

Again, it will ask about style sheet as below. If you want CSS just click enter or else select any by pressing up and down arrows.

```
? which stylesheet format would you like to use?
```

Finally, application has been created with required and supporting files and folders.

Run the application:

To navigate the folder where the project has been created type the below command

```
cd Project-Name
```

to run the app give below Angular CLI command

```
ng serve –open
```

The *ng serve* command will launch the server & will open your browser to *http://localhost:4200/*.

Common issues while creating and run angular application at first time:

Issue1: "cannot be loaded because running scripts is disabled on this system".
Solution: give below command in terminal

```
set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted
```

Issue2: when app.module.ts file has not been created in application.
Solution: give --no-standalone in command while creating new application.

```
ng new projectname --no-standalone
```

Useful commands in Angular:

npm uninstall -g @angular/cli	→	To uninstall angular
npm install -g @angular/cli@latest	→	To install latest angular cli

4. Angular Application Structure:

- **src folder:** This is the folder which contains the main code files related to your angular application.
- **app folder:** The app folder is root module folder contains app.module.ts, app.component.html, app.component.css, app.component.ts and app.component.spec.ts files.
- **app.module.ts:** This is also a typescript file which includes all the dependencies for the website. This file is used to define the needed modules to be imported, the components to be declared and the main component to be bootstrapped.
- **app.component.html:** This file contains the html file related to app component. This is the template file which is used by angular used for other components template.
- **app.component.css:** This file contains the cascading style code for app component.
- **app.component.ts:** This is the typescript file which includes the view logic behind the app component.
- **app.component.spec.ts:** This file is a unit testing file related to app component. This file is used along with unit tests. It is run from Angular CLI by the command ng test.
- **assets folder:** This folder is for resource files which are used in the application such as images, locales, translations etc.
- **favicon.ico:** This file specifies a small icon that appears next to the browser tab of a website.
- **index.html:** This is the entry file which holds the root container for the angular application.
- **main.ts:** As defined in angular.json file, this is the main ts file that will first run. This file bootstraps (starts) the AppModule from app.module.ts , and it can be used to define global configurations.
- **styles.css:** This is a global css file which is used by the angular application. It provides styles to all components in application.

- **angular.json:** It is very important configuration file related to your angular application. It defines the structure of your app and includes any settings associated with your application. Here, you can specify environments on this file (development, production). This is the file where we add Bootstrap file to work with Angular.

- **package.json:** This is npm configuration file. It includes details about your website's package dependencies along with details about your own website being a package itself.
- **package-lock.json :** This is an auto-generated and modified file that gets updated whenever npm does an operation related to node_modules or package.json
- **.gitignore:** This file is related to the git to ignore what ever do you want.
- **.editorconfig:** This is a simple file which is used to maintain consistency in code editors to organize some basic things such as indentation and whitespaces.
- **tsconfig.json:** This is a typescript compiler configuration file.
- **tsconfig.app.json:** This is used to override the tsconfig.json file with app specific configurations.
- **tsconfig.spec.json:** This overrides the tsconfig.json file with app specific unit test configurations.

5. Modules & Components

Modules and components are fundamental building blocks that help organize and structure the application.

Modules:

Modules in Angular are containers for organizing and managing related components, directives, services, pipes and other application code.

Angular modules play a crucial role in dependency injection, helping to manage the application's runtime behavior.

Each Angular application has at least one module, known as the root module, which is typically named AppModule.

Modules help to encapsulate and organize the different parts of an application.

Modules facilitate modularity, code reusability, and maintainability.

Modules are defined using the @NgModule decorator.

Modules can import other modules to compose larger applications from smaller.

Components:

Components encapsulate the logic, data, and presentation of a part of the user interface (UI).

Each component consists of a TypeScript class (with metadata) and an HTML template.

Components enable the development of complex applications by breaking them down into smaller, manageable pieces.

Components are defined using the @Component decorator.

Components consist of a TypeScript class that contains the component's properties and methods.

Components have a template (HTML) that defines the structure and layout of the UI associated with that component.

Data binding and communication between components are facilitated by properties and events.

Example for Module:

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

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

Example for Component:

```
// app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<h1>Hello, {{ name }}!</h1>',
})
export class AppComponent {
  name = 'Angular';
}
```

How to create new module and its components in angular?

The Angular CLI (Command Line Interface) provides a convenient way to create modules.

Open a terminal or command prompt and navigate to your Angular project directory.

Use the ng generate module command to create a new module.

```
ng generate module your-module-name
```

This command will create a new folder for your module with the specified name and generate necessary files like **your-module-name.module.ts**.

After creating the module, you can add components, services, directives, and other elements to it as needed.

```
ng generate component your-module-name/your-component-name
```

This will create a new component within the specified module.

Integrate the Module:

Once your module is created, you can integrate it into other modules or the root module by adding it to the imports array of the importing module like app module.

// app.module.ts (or another module where you want to use your new module)

```
import { NgModule } from '@angular/core';
import { YourModuleNameModule } from './path-to-your-module/your-module-name.module';
```

```
@NgModule({
  imports: [YourModuleNameModule],
  // Other module configurations
})
export class AppModule { }
```

Integrate the Component:

Once integration of your-module is done with other module then only we can integrate components and services of that module into the other module.

For this component integration we should export component in

your-module.module.ts file as below.

```
//your-module.module.ts
@NgModule({
  declarations: [YourComponent],
  imports: [CommonModule],
  exports: [YourComponent]
})
export class YourModule { }
```

once component is included in exports we can use this component in any module where we imported this component's module.

How to use one component into another component?

If it is HTML template:

Component's template we can use in other component's template by its selector.

```
//app.component.html
<your-component></your-component>
```

Component's typescript file content can use in other component's typescript by its class name. That's why we do export class in every typescript file.

```
            //app.component.ts
export class AppComponent { }
```

How to delete component in angular?

Deleting a component in Angular involves removing the component files, updating the module file, and potentially removing references from other parts of your code.

Here are step-by-step instructions:

Delete the component files (component TypeScript file, HTML file, CSS file, and the test files) from the src/app directory.

Open app.module.ts or another module file where the component is declared.

Remove the import statement for the deleted component.

Remove the component from the declarations array.

Check other files for any references to the deleted component and remove them.

After performing these steps, the component should be successfully deleted from your Angular project. Ensure that you test your application to verify that the deletion didn't introduce any issues.

6. Routing in Angular Application

In Angular, routing refers to the process of navigating between different views or components in a single-page application (SPA). Angular provides a built-in router module that allows you to define routes and map them to specific components.

RouterModule:

Angular's routing functionality is provided by the RouterModule, which is part of @angular/router package. This module needs to be imported into your Angular application. While creating application routing module will be created and RouterModule automatically imported in app module.

Routes:

Routes are defined in the Angular application by configuring the RouterModule with an array of route definitions. Each route definition specifies a path and the component to be displayed when that path is matched.

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  // Additional routes can be defined here
];
```

RouterOutlet:

In your main component template (usually app.component.html), you place a <router-outlet></router-outlet> tag. This is where Angular will render the component corresponding to the current route.

```
<router-outlet></router-outlet>
```

Navigation:

Navigation between different views/components is typically done using the Router service provided by Angular.

You can navigate declaratively in your HTML templates using directives like routerLink.

```
<!-- Declarative navigation -->
<a routerLink="/home">Home</a>

<a routerLink="/about">About</a>
```

You can navigate imperatively in your TypeScript code using methods like navigate() or navigateByUrl(). We can discuss more about this in later topics.

```
// Imperative navigation
import { Router } from '@angular/router';
constructor(private router: Router) { }
navigateToHome() {
        this.router.navigate(['/home']);
}
```

Route Parameters:

Routes can also contain parameters, which allow you to pass data dynamically to components. Parameters are specified in the route path and can be accessed in the component using ActivatedRoute service. We can discuss more about this in later topics.

```
// Route configuration
{ path: 'user/:id', component: UserComponent }

// Accessing route parameters in component
import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {
  this.route.params.subscribe(params => {
    console.log(params['id']);
  });
}
```

Example:

Without any bootstrap and angular material, create angular application that demonstrate modules, components and routing.

Step1: Create two modules kitchen and bedroom.
Step2: Create mixi, fridge components in kitchen module.
Step3: Create bed component in bedroom module.
Step4: Export bed component in bedroom.module.ts file.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { BedComponent } from './bed/bed.component';

@NgModule({
  declarations: [
    BedComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    BedComponent
  ]
})
export class BedroomModule { }
```

Step5: Export mixi and fridge components in kitchen.module.ts file.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MixiComponent } from './mixi/mixi.component';
import { FridgeComponent } from './fridge/fridge.component';

@NgModule({
  declarations: [ MixiComponent, FridgeComponent ],
  imports: [
    CommonModule
  ],
  exports: [
    MixiComponent,
    FridgeComponent
  ]
})
export class KitchenModule { }
```

Step6: import kitchen module and bedroom module in app.module.ts file.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { KitchenModule } from './kitchen/kitchen.module';
import { BedroomModule } from './bedroom/bedroom.module';

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

Step7: create routes for bed, mixi & fridge components in app-routing.module.ts file.

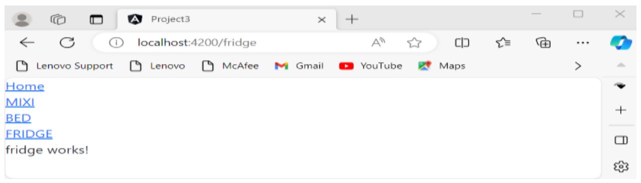
```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { MixiComponent } from './kitchen/mixi/mixi.component';
import { FridgeComponent } from './kitchen/fridge/fridge.component';
import { BedComponent } from './bedroom/bed/bed.component';
const routes: Routes = [
  {
    path: 'mixi', component: MixiComponent
  },
  {
    path: 'fridge', component: FridgeComponent
  },
  {
    path: 'bed', component: BedComponent
  }
];
@NgModule({
        imports: [RouterModule.forRoot(routes)],
        exports: [RouterModule]
})
export class AppRoutingModule { }
```

Step8: call routes in app.component.html with normal html anchor tag.

```
<a href="#">Home</a><br>
<a href="mixi">MIXI</a><br>
<a href="bed">BED</a><br>
<a href="fridge">FRIDGE</a>

<router-outlet></router-outlet>
```

Output:



```
localhost:4200/fridge
Home
MIXI
BED
FRIDGE
fridge works!
```

How to install bootstrap, jquery, popper in angular?

Step1: Run the following command in your terminal to install these packages

```
npm install bootstrap jquery popper.js
```

Step2: Open angular.json file, which is in the root directory of your Angular project. In the styles array, add the Bootstrap CSS file path:

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.css"
],
```

In the scripts array, add the jQuery, Popper.js, and Bootstrap JavaScript file paths:

```
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/popper.js/dist/umd/popper.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

Step3: If you need to use Bootstrap components that require JavaScript initialization, like modals or tooltips, you might need to import jQuery and Popper.js into your Angular components. In your app.module.ts, you could import jQuery and Popper.js

```
import * as jQuery from 'jquery';
import 'popper.js';
```

Note: It's generally not recommended to import jQuery directly into Angular components, as Angular encourages the use of its own tools and methods for DOM manipulation. However, if you're using Bootstrap, which relies on jQuery for some of its features, you may need to import it in specific cases.

What is best way for UI design in angular?

There are several approaches and tools commonly used by Angular developers for UI design. Angular Material is a popular choice for UI design in Angular applications. It provides a comprehensive set of Material Design components that are well-tested, accessible, and easy to use. Angular Material's components follow Google's Material Design guidelines, making them a great option for building modern and visually appealing user interfaces.