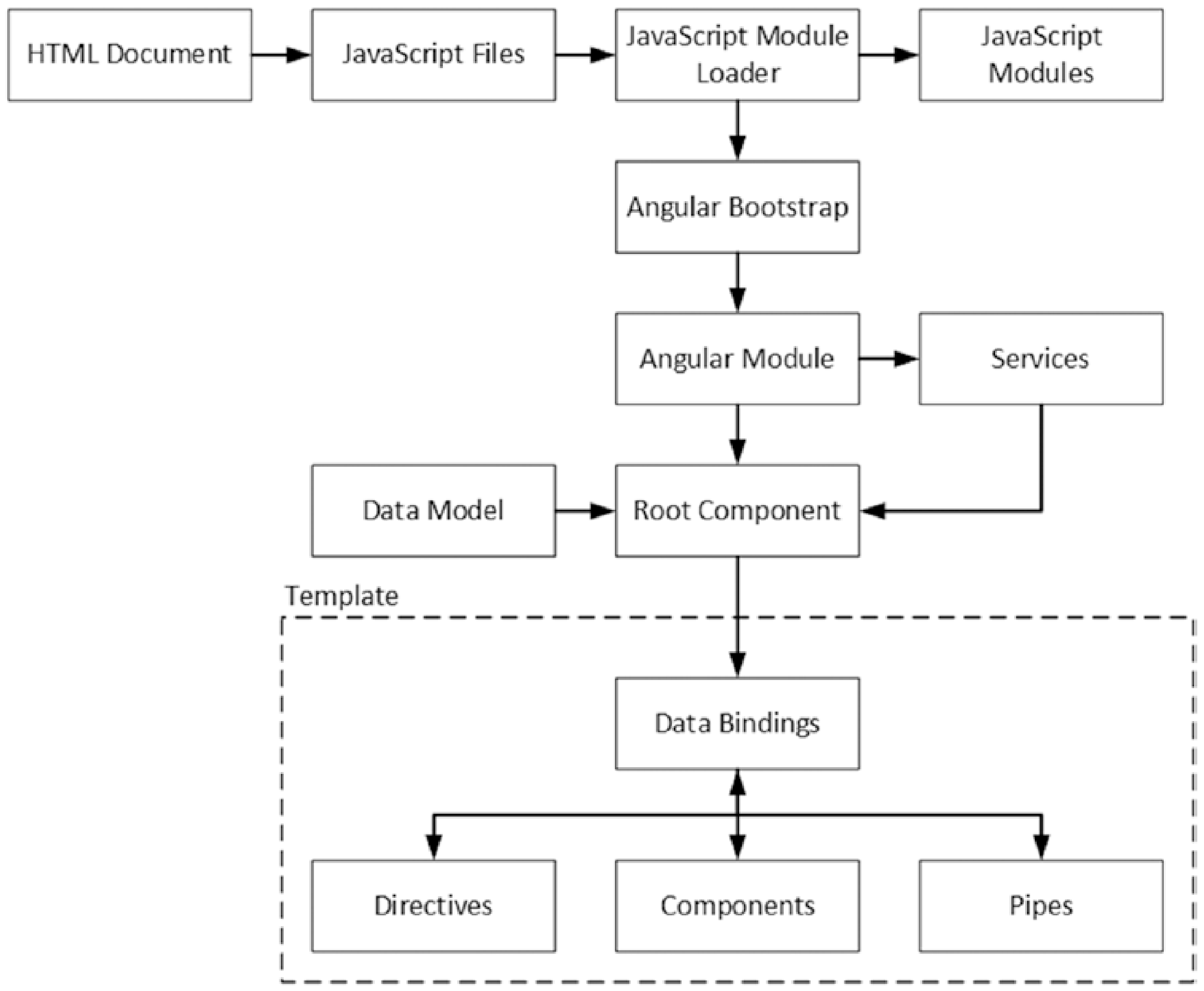
Angular 2 architecture: -



HTML Document: -

This is the HTML document that is requested by the browser and that contains all the content required to load and start the Angular application.

JavaScript Files: -

These are regular JavaScript files that are added to the HTML document using script elements. The JavaScript files are used to add missing features required by Angular that are not supported by the browser and for the JavaScript module loader.

Module Loader: -

This is a JavaScript component that is responsible for loading the Angular modules and the other application building blocks.

JavaScript Modules:-

These are JavaScript files that are packaged into a specific format that allows dependencies between packages to be managed by the module loader to ensure that the JavaScript code in the application is loaded correctly and executed in the right order. The Angular framework is delivered as a set of JavaScript modules, as are some of the required support libraries.

Angular Bootstrap:-

This is a TypeScript file that configures Angular and specifies the Angular module, which is loaded to start the application. See the “Bootstrapping the Application” section for details.

Angular Module:

All Angular projects have a root module that describes the application and can contain additional modules to group related features together and make the project more manageable.

Data Model:-

The model provides the data in the application and the logic required to access it.

Root Component:-

The root component is the entry point into the application and is responsible for generating the dynamic content that is displayed to the user.

Template:-

The template is the HTML content that is displayed by the root component.

Data Bindings:-

Data bindings are annotations in the template that tell Angular how to insert dynamic content, such as data values, and how to respond to user interactions.

Directives:-

Directives are classes that transform HTML elements to generate dynamic content.

Components:-

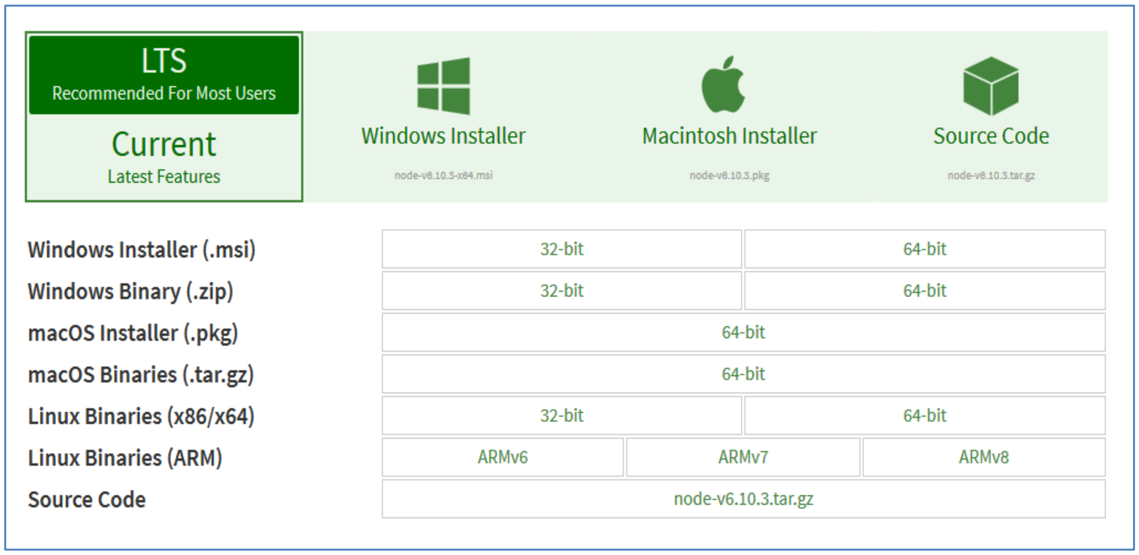
Components add content to the application to provide application features.

Pipes:-

Pipes are classes that are used to format data values for display to the user.

Setting Up the Development Environment: -

1. Install node.js into machine. Download URL is https://nodejs.org/en/download/ .



Windows Installer (.msi)	32-bit		64-bit
Windows Binary (.zip)	32-bit		64-bit
macOS Installer (.pkg)	64-bit		
macOS Binaries (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit		64-bit
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v6.10.3.tar.gz		

2. Once download and install verify the version no.

```
node –v
```

3. Node.js has been helped by the Node Package Manager (NPM), which provides easy access to an enormous catalog of development packages, including Angular. NPM takes care of downloading packages and managing the dependencies between them.

```
npm install -g
```

4. For the application development we can use the sublime/atom/vscode. All are open source and easy to work.
5. Install the typings with the following commands

```
npm run typings --install dt ~ core-js --save --global
npm run typings --install dt ~ node --save --global
```

how to test the examples?

1. Open the command prompt and run the following command.

```
npm start
```

2. open the browser and type http://localhost:3000 to see the output.

Creating and Preparing the Examples: -

1. Creating the Package File

   NPM uses a file called package.json to get a list of the software packages that are required for a project.

```
{
        "dependencies": {
                "@angular/common": "2.2.0",
                "@angular/compiler": "2.2.0",
                "@angular/core": "2.2.0",
                "@angular/forms": "2.2.0",
                "@angular/platform-browser": "2.2.0",
                "@angular/platform-browser-dynamic": "2.2.0",
                "reflect-metadata": "0.1.8",
                "rxjs": "5.0.0-beta.12",
                "zone.js": "0.6.26", "core-js": "2.4.1",
                "classlist.js": "1.1.20150312",
                "systemjs": "0.19.40",
                "bootstrap": "4.0.0-alpha.4"
            },
        "devDependencies": {
                "lite-server": "2.2.2",
                "typescript": "2.0.3",
                "typings": "1.4.0",
                "concurrently": "3.1.0"
            },
        "scripts": {
                "start": "concurrently \"npm run tscwatch\" \"npm run lite\" ",
                "tsc": "tsc",
                "tscwatch": "tsc -w",
                "lite": "lite-server",
                "typings": "typings"
            }
}
```

Dependencies:-
This is a list of NPM packages that the web application relies on to run. Each package is specified with a version number. The dependencies section in the listing contains the core Angular packages, libraries that Angular depends on, polyfill libraries that add modern features for old browsers, and the Bootstrap CSS library.

devDependencies:-
This is a list of NPM packages that are relied on for development but that are not required by the application once it has been deployed. This section contains packages that will compile TypeScript files, provide a development HTTP server, and allow multiple commands to be run at the same time using NPM.

Scripts:- This is a list of scripts that can be run from the command line. The scripts section in the listing starts the TypeScript compiler and the development HTTP server.

2. Installing the NPM Packages

   To process the package.json file to download and install the packages that it specifies, run the following command inside the example folder:

```
npm install
```

   Ignore all warnings. After completion of this file we can see the node_modules folder in our example.

3. Configuring the TypeScript Compiler

   Angular applications are written in TypeScript, which is a superset of JavaScript.
   but requires that TypeScript files are processed to generate backward-compatible JavaScript that can be used by browsers.

The TypeScript compiler requires a configuration file to control the kind of JavaScript files that it generates. I created a file called tsconfig.json in the example folder.

```
{
        "compilerOptions": {
                "target": "es5",
                "module": "commonjs",
                "moduleResolution": "node",
                "emitDecoratorMetadata": true,
                "experimentalDecorators": true
        },
        "exclude": [ "node_modules" ]
}
```

4. Installing the TypeScript Type Information

The TypeScript compiler relies on descriptions of the standard JavaScript APIs, known as type definitions, for the enhancements that TypeScript provides to the JavaScript language specification.

Run the following commands in the examples folder.

```
npm run typings -- install dt ~ core-js --save --global
npm run typings -- install dt ~ node --save –global
```

Once the commands have completed, the todo folder will contain typings/globals/core-js and typings/global/node folders that contain the type definitions and a typings.json file that contains details of the type information that was downloaded.

5. Create the HTML File

```
<!DOCTYPE html>
<html>
        <head>
                <title>ToDo</title>
                <meta charset="utf-8" />
                <link href="node_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
        </head>
        <body>
                <h1>Angular2</h1>
        </body>
</html>
```

6. Starting the Server

```
npm start
```

7. Open the browser and type the localhost:3000. You can see the hello world message in the browser.

Component: -

Angular component is responsible for managing a template and providing it with the data and logic it needs.

Components make it possible to define self-contained blocks of functionality, which makes projects more manageable and allows for functionality to be more readily reused.

The @Component decorator is applied to a class, which is registered in the application's Angular module.

Components provide all the functionality of directives, with the addition of providing their own templates.

An Angular application must contain at least one component, which is used in the bootstrap process. Aside from this, you don't have to add additional components, although the resulting application becomes unwieldy and difficult to manage.

Ex: -

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

@Component({
        selector: "hello-world",
        templateUrl: "app/helloworld.component.html"
})

export class HelloWorldComponent {
        getMessage() {
                return "Hello World";
        }
}
```

Component Usage: -

• Providing Angular with an entry point into the application, as the root component

• Providing access to the application's data model so that it can be used in data bindings

• Defining the layout that contains the form and the table

• Checking that the form data is valid when a new product is created

• Maintaining state information used to prevent invalid data being used to create data

• Maintaining state information about whether the table should be displayed

Component: -

An Angular component is responsible for managing a template and providing it with the data and logic it needs.

@Component: -

Component decorator allows you to mark a class as an Angular component and provide additional metadata that determines how the component should be processed, instantiated and used at runtime.

Components are the most basic building block of an UI in an Angular application.

A component must belong to an NgModule in order for it to be usable by another component or application.

Metadata Properties:

- animations - list of animations of this component
- changeDetection - change detection strategy used by this component
- encapsulation - style encapsulation strategy used by this component
- entryComponents - list of components that are dynamically inserted into the view of this component
- exportAs - name under which the component instance is exported in a template
- host - map of class property to host element bindings for events, properties and attributes
- inputs - list of class property names to data-bind as component inputs
- interpolation - custom interpolation markers used in this component's template
- moduleId - ES/CommonJS module id of the file in which this component is defined
- outputs - list of class property names that expose output events that others can subscribe to
- providers - list of providers available to this component and its children
- queries - configure queries that can be injected into the component
- selector - css selector that identifies this component in a template
- styleUrls - list of urls to stylesheets to be applied to this component's view
- styles - inline-defined styles to be applied to this component's view
- template - inline-defined template for the view
- templateUrl - url to an external file containing a template for the view
- viewProviders - list of providers available to this component and its view children

For Ex: -

```
import { Component } from "@angular/core";
@Component({
   selector: "project-app",
   template: "{{message}}"
})
export class AppComponent {
   message: string;
  constructor() {
   this.message = 'hello';
  }
}
```

@Injectable

A marker metadata that marks a class as available to Injector for creation.

```
@Injectable() class UserService {

}
```

@Directive

Marks a class as an Angular directive and collects directive configuration metadata.

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

@Directive({ selector: 'my-directive', })

export class MyDirective {

 }
```

Metadata Properties:

- exportAs - name under which the component instance is exported in a template
- host - map of class property to host element bindings for events, properties and attributes
- inputs - list of class property names to data-bind as component inputs
- outputs - list of class property names that expose output events that others can subscribe to
- providers - list of providers available to this component and its children
- queries - configure queries that can be injected into the component
- selector - css selector that identifies this component in a template