To get started using an Angular Material,

of course the first thing that you need to do is to install the Angular Material module.

So, at the prompt type :

**npm install --save @angular/material**

We are installing the 6.4.7 version of the Angular Material module.

In addition, the material module requires the installation of CDK.

So, we'll also install that by doing

**npm install @angular/cdk@6.4.7 --save**,

and install that together with Angular Material.

So, Angular Material makes use of the Angular cdk for its components.

In addition, when you are using Angular Material,

you also need to import the Angular animations module into your Angular application.

So, let's go ahead and install that too.

So, you'll say,

**npm install --save @angular/animations@6.1.7**.

Also, another module that I will install

together is called HammerJS.

The HammerJS module is used by Angular

for supporting some gestures within your Angular application.

So, that's the reason why we install HammerJS also within our application,

and the current version of HammerJS that we are using is 2.0.8.

**npm install hammerjs@2.0.8**

Next, I'm going to move on to install

the Angular Flex-Layout within my Angular application.

So, to do that, I pad the prompt

**npm install --save @angular/ flex-layout**

So, now that we have all the pieces that we need for our Angular application installed,

let's go ahead and then configure our Angular application to make

use of these new modules that we installed into our Angular application.

The first thing that we will do is go to index.HTML file,

and then we will configure the index.HTML file to use the Angular Material icons.

So, to include the Angular Material icons so that they can be

used within the templates of your Angular application, so,

let me go right there in the **head of index.HTML**,

and then create a new link with

**href="https://fonts.googleapis.com/icon?family=Material+Icons"** and this is a style sheet.

So, with this, I can now make use of the Angular Material icons within

my Angular application wherever I choose to do so within the templates.

The next step is to go to the **app module.ts** and

then import the three modules that we have just installed.

So, the first one that I'm going import is called as

the **BrowserAnimationsModule** which I will import **from @angular/platform-browser/animations**. So, now the next one

that I'm going to import is the **MaterialToolbarModule.**

So, this imports the **@angular/material/toolbar** module

that can be used within our Angular applications templates.

So, Angular Material, and also import

the **FlexLayoutModule** from **@angular/flex-layout**.

So, once we have all these modules imported then one additional change that I need to do is- oh let me also import Hammer.JS while I am right there.

So, let me import Hammer.JS into my application.

And once I am at it,

the other change that I need to do is to also include

those modules into the inputs of the NG module decorator here.

So, I'm going to go there and say browser module,

browser animations module there and then after the HTTP module,

I will import the material toolbar module and also the flex

layout module as part of the inputs inside the NG module decorator for my app module.

Summary:-

npm install @angular/material@6.4.7 --save

npm install @angular/cdk@6.4.7 --save

npm install --save @angular/animations@6.1.7

npm install --save hammerjs@2.0.8

===========================================

Next, include the following into the <head> of index.html to make use of Material Design icons:

**<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">**

==================================

Configure your Angular Project to use Flex Layout

* Next, install Angular Flex Layout as follows:

**npm install --save @angular/flex-layout@6.0.0-beta.18**

Updating AppModule

* Then, you need to import the Angular Animations Module, Angular Material Toolbar Module, Flex Layout Module and hammerjs into your root module (src/app/app.module.ts) as follows

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { MatToolbarModule } from '@angular/material/toolbar';

import { FlexLayoutModule } from '@angular/flex-layout';

import 'hammerjs';

@NgModule({

imports: [

BrowserAnimationsModule,

MatToolbarModule,

FlexLayoutModule

],

})

==============================================================

**Component:-**

A component is nothing but a JavaScript class or a Type Script class. So that's why you see here, defining a class saying AppComponent and then you are **exporting** this class. The reason why we use the **export** here, so that this component can be imported into my app module. So you saw that we were importing this into my app module. So whenever you want to **make any component or a module in portable in another module**, then you would always prepend the export to the class here.

So, you will always have a **root component.** So for our application, the app.component.ts file and the corresponding HTML template forms the root component for our application and this root component can then contain

components down in a hierarchy and can include components in the hierarchy.

**Structural Directives:-**

Structural directives are one kind of directives. So directives is a general umbrella under which you have components, then you have structural directives, and then you have attribute directives.

The directives help us is to enable us to give instructions to the Angular on how to render the templates to the DOM. So, they use directives to give instructions to Angular as it is rendering the templates on the screen. So, in general, a directive can be defined directly in Angular using a class with the @Directive decorator. A component is a special kind of directive in Angular, which has its own template associated with it.

There are two other kinds of directives in Angular, structural and attribute directives. Structural directives help us to alter the layout of our content by helping us to add and remove elements from the DOM. So you are literally using the structural directives to manipulate your DOM of your webpage. So, you can use the structural directives by applying them to a host element, typically a div or a list item in the DOM and also to its descendents.

**Learn Mat-grid-list and mat-card components in angular material.**

**Angular Dependency Injection:-**

Dependency injection Is a software design pattern. This is a pattern that is useful for implementing applications where you have one object that is dependent on another object in a way that is efficient. Now the Dependency Injection, as we realize, has two parts. Dependency and injection.

Dependency means that your object is dependent on another object.

So, that is why that is a dependency between your object and another object.

Injection is talking about passing the dependency to a dependent object.

So, if you have an object, if there is a mechanism that allows you to be able to take an object and then make it available to a second object, so that the other object can make use of it without being aware of exactly how the first object is implemented. So, in this case the object that is dependent is unaware, or does not need to worry about how the other object is implemented.It just needs to make use of it, within your object, wherever it is.

If an object is dependent on another object there are three ways it can access the other object.1, It can create the dependent object itself by using the new operatoryou have seen this in the case of classes.So, for example, if you need a new object of a class type,then you create that object using the new within languages, like C++ or Java, and even type script.Then, you can make use of that particular object.The other way is to declare the other object as a global variable, and then you look up the dependency using the global variable.The third way of doing this is to have the dependency passed into you,wherever it is needed.So if you're dependent on something else, then that something else will be injected into you by a system wherever it is required.So, now the third option gives you a lot of flexibility in the way that software is designed.So in this approach, in the third approach, there is no need for hard coding after dependency.

In the first two approaches, the dependency is hard coded in there, because you need to be fully aware of how you create the object that you are dependent upon, within your own object.

Dependency Injection brings about different roles that we need to consider.

First of course is the service that you're going to make use of within your component for example and that needs to be injected in.

Second the client which is dependent upon the service which in this case is your component.

Third, the interface, once injected how do you make use of that service.

And finally the **injector**, the entity that is responsible for injecting that object into your object.

Now, how is this injection taken care of?

This injection is taken care of by the **angular injection subsytem**.

The angular injection subsystem takes care of creation of these services and then inject them into your component wherever you need them. So it also takes care of resolving the dependencies and also,providing those objects to other components that require these objects. So the angular injector subsystem provides all these mechanisms for

us to enable this.