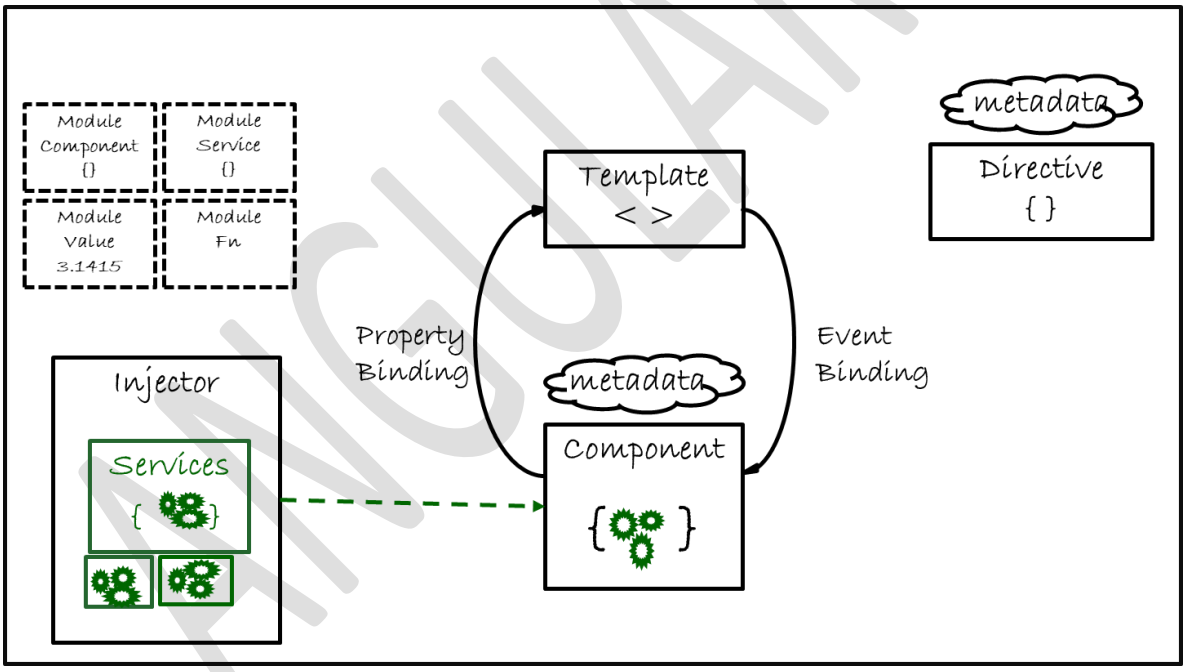
ANGULAR 6

Overview of Angular :

- Angular is a JavaScript framework for developing single page applications
- Angular can be built using ES6 or Typescript
- Typescript uses
  - Class-based object oriented programming.
  - Static Binding

Angular Architecture:



Modules

- Used to break up application into logical pieces of code

Components

- Similar to class

Template

- Define view of angular js

Directive

- To modify DOM element and extend their behaviour.

Services

Set of codes shared by different Components of application.

Install angular using Angular CLI:

Prerequisites

- Node and npm should be installed

Install Angular CLI from command prompt

- Install angular-cli globally **npm install –g @angular/cli**
- Check CLI installation **ng –v**

Create a new Project

- To create a new project **ng new my-project**
- To check angular version, navigate to project folder and type **ng –v**

Run the application

- To run the application **ng serve**
- To run the application and open in browser **ng serve --open**

Building Blocks Of Angular:

- Modules
- Components
- Template
- metadata

- Data binding
- Directives
- Services
- Dependency Injection

Modules:

- Angular apps are modular and provides logical boundary to applications.
- Angular modules are **NgModules**.
- Every Angular app contains at least one Angular module, i.e. the root module as AppModule.
- An application can have one or more root modules
- A module is a class with **@NgModule** decorator.
- ***app.module.ts*** is the root module present in the src folder
- Logical division of the code.

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
 @NgModule ({
     imports: [ BrowserModule ],
     declarations: [ AppComponent ],
     bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Bootstrap Array

- Has the components that should be loaded, so that they can be used in the application.
- Declare the components, so that they can be used across other components in the Angular JS application.

Import array

- Imports components from other angular modules.

Export array

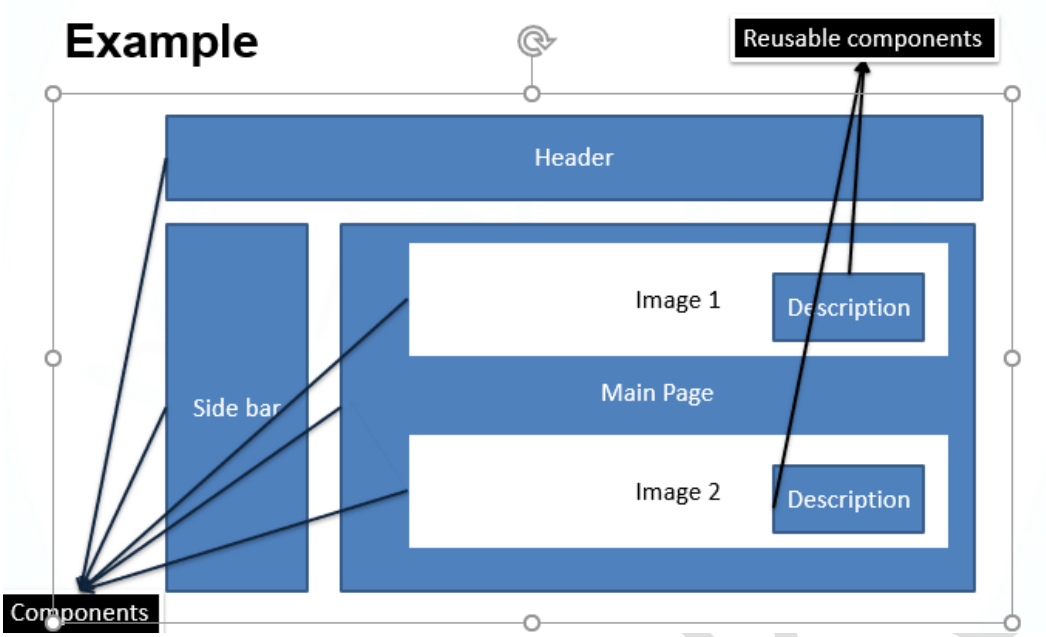
- Exports components, directives and pipes which can be used in other modules.

Decorators:

- Decorators are functions that modify JavaScript classes
- They are used for attaching metadata configuration to the classes
- **@NgModule** is a decorator function which takes metadata object whose properties describe the module
- **@Component** is a decorator function which takes metadata object whose properties describe the component

Components:

- Every angular project has atleast one component called root component.
- Single web page is composed of one or more reusable components
- ***app.component.ts*** is the default component present during creation of angular project.
- Components are classes decorated with **@Component** Decorator
- Each component has a template which can communicate with the code in the component class



App.component.ts :

Eg:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular 4';
}
```

Selector - tag name

templateUrl - the page that will be displayed upon calling the component

styleUrls - the stylesheet for the component

Class name: AppComponent
File name: app.component.ts

Creating a Component:

- To create a component use - **ng g component <component-name>**

  or

  ng g c <component-name>

DATA BINDING:

- Is a technique of passing data from the data source to the view target and viceversa
- Is categorized as
  - One-way(source-to-view or view-to-source)
  - Two-way(view-to-source-view)

One-way(source-to-view)

- Interpolation, Property, Attribute, Class, Style Binding

One-way(view-to-source)

- Event

Two-way

- Using ngModel

Interpolation:

- Retrieve the data in the view using **{{ }}**
- The text between the braces
  - Can be the name of a component property.
  - Can be template expression
- Angular evaluates the expression and then converts expression results to strings

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-sample',
  templateUrl:'./sample.component.html' ,
  styleUrls: ['./sample.component.css']
})
export class SampleComponent implements OnInit {

 message: string = "Have a good day";
 employee = {
   name:'Ram',
   age:20,
   city:'Bangalore'
 }
  constructor() { }
  ngOnInit() {  }

}
```

Sample.component.ts

```
<h1> {{message}}</h1>
<h2>Welcome {{employee.name}}</h2>
<h2>The sum of 2 numbers is {{2+1}}</h2>
```

Sample.component.html

PROPERTY BINDING:

- Is used for passing the data from the source and setting the value of a given element in the view.
- Property Binding can be defined in three ways as
  - Interpolation – (for string value)
  - By wrapping the element property with brackets
  - By prefixing **bind-** before the element property

```
<img src='{{newLogo}}'>
<img [src]='newLogo'>
<img bind-src='newLogo'>
```

```
newLogo = '/assets/images/angular.jpg';
```

```
<button [disabled]='buttonStatus'>Click</button>
<button bind-disabled='!buttonStatus'>Show</button>
```

```
buttonStatus = true;
```

EVENT BINDING:

- Happens when the user interacts with the application
- Event binding sends information from the view to the source

```
<button (click)='changeMessage("Welcome Home")'>Change Message</button>
<h1> {{message}}</h1>
```

```
message = 'Have a good day';
changeMessage = function (message) {
  this.message = message;
};
```

CLASS BINDING:

- Allows to set the CSS class dynamically for a DOM element.
- Controls how the elements appear by adding and removing CSS classes dynamically.
- Can add multiple classes to an element using **ngClass**

```
<h1 [class] = 'myClass'>Class Binding</h1>
<h1 [class.header-style] = 'newClass'>Welcome!</h1>
```

```
myClass = 'header-style';
newClass = true;
```

```
.header-style{
    color: green;
    font-size:20px;
}
```

STYLE BINDING:

- Allows to dynamically change CSS styles for DOM elements
- Can add multiple styles to an element using **ngStyle**

```
<h1 [style.color]='newStyle'>Great Day</h1>
<h1 [style.color]="changeStyle ? 'blue':'brown'">Change Color</h1>
```

```
newStyle = 'orange';
changeStyle = false;
```

TWO WAY BINDING:

- Is used for passing the data from the source to view and from view to source
- Two way binding is achieved with **ngModel** directive

- Import FormsModule to work with ngModel
- Wrap ngModel with parenthesis and square brackets

```
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

import { SampleComponent } from './sample/sample.component';
import { GreetComponent } from './greet/greet.component';

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

```
<input [(ngModel)] = 'username' >{{username}}
```

```
 username = 'Raju';
```

Templating – External and Inline

- Templates helps to define the UI of the app
- It can be inline or external
- An external template is defined by the **templateUrl** property
- The html code can also be added inline and is defined by the **template** property of the component decorator

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-greet',
  template: `
  <h2>
  Hi {{name}}!!  {{message}}
  </h2>
  `,
  styleUrls: ['./greet.component.css']
})
export class GreetComponent implements OnInit {

message :string = 'Have a good day';
name :string;
  constructor() {
    this.name='Raju';
   }
  ngOnInit() { }
}
```

Defining Styles – External and Internal

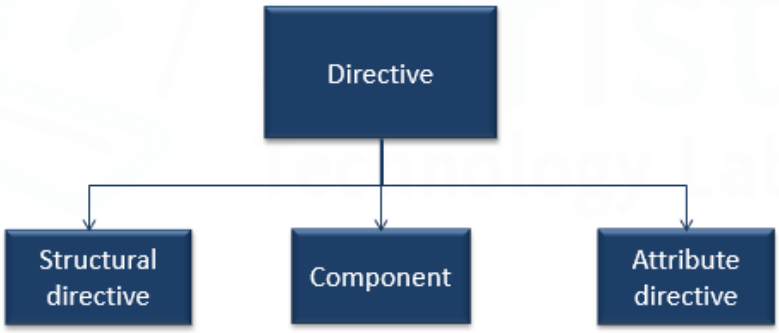
- By default Angular CLI will setup the components to work with external stylesheets.
- An external stylesheet is defined by the **styleUrls** property
- An inline CSS is defined by the **styles** property of the component decorator

```
@Component({
  selector: 'app-greet',
  templateUrl: './greet.component.html',
  styles: [`
   h1{
     background-color:blue;
     color:pink;
     }
    .one{
      letter-spacing:3px;
    }
  `]
})
export class GreetComponent implements OnInit {
  check = true;
```

```
<h1 [class.one]='check'>
  This is  demo
</h1>
```

DIRECTIVE:

- A directive is a custom HTML element that is used to extend the power of HTML



Structural Directive:

- Responsible for html layout.
- Shape and reshape DOM element.
- Identified by * symbol.

Built-in structural directive

- NgIf
- NgFor
- NgSwitch

NgIf:

- It takes a boolean expression
- Makes an entire chunk of the DOM appear or disappear.
- Doesn't hide elements with CSS.
- Adds and removes physically from the DOM.

```
export class GreetComponent implements OnInit {
uname:String;
course:String;
constructor(){
  this.uname='Raju';
  this.course='Angular'
}
  ngOnInit() { }
}
```

```
<div>
  <h2 *ngIf="uname=='Raju' else newtempl">
    Welcome {{uname}}!!!{{message}}
  </h2>
  <ng-template #newtempl><h2>Sorry wrong user</h2>
  </ng-template>

  <p *ngIf="course=='Angular';then templ1 else templ2"></p>
  <ng-template #templ1>
    <h2>You have selected {{course}}</h2>
  </ng-template>
  <ng-template #templ2>
    <h2>course not available</h2>
  </ng-template>
  <div>
```

NgFor:

- Iterate items in template.
    - Run as a loop in collection.
    - Bind data in html template.

- Anything changes in collection , the template recreates and changes the DOM.

```
export class GreetComponent implements OnInit {

message :string = 'Have a good day';
courses:string[] = ['Java','Node','Angular'];
  ngOnInit() { }
}
```

```
<ul >
  <li  *ngFor = "let course of courses">
    {{course}}
  </li>
</ul>
```

Using a Class For Data:

To create a class use **ng g class <class-name>**

```
export class Course {
    //declares a constructor parameter and its type
    //creates a public property with the same name
    //initilazes the property when creating the instance
    constructor(
        public name:String,
        public courseId:number){ }
}
```

NgFor - with array of objects

```
export class SampleComponent implements OnInit {

 message = 'Have a good day';
 courseList:Course[] = [
   new Course("Java",1), new Course("Node",2), new Course("Angular",3),
   new Course("Mongo",4),new Course("Ember",5),new Course("Javascript",6),
   new Course("Node",7)
 ]
constructor() { }
   ngOnInit() {  }
}
```

```
<ul *ngFor="let course of courseList">
  <li>
    {{course.courseId}}.{{course.name}}
  </li>
</ul>
```

Attribute Directive:

- Changes appearance and behaviour of DOM element, component or any other directive.
- Used as attributes of elements.

Inbuilt directives

- **NgClass** - add and remove a set of CSS classes
- **NgStyle** - add and remove a set of HTML styles
- **NgModel** - two-way data binding to a HTML form element

NgClass:

- NgClass directive allows to set the CSS class dynamically for a DOM element.
- It controls how elements appear by adding and removing CSS classes dynamically.
- Set value by passing object literal to the directive
- Can bind CSS classes that are using string array and object to NgClass

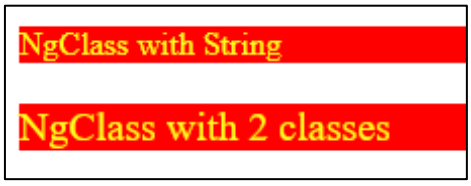
NgClass With String:

- To bind the CSS class to NgClass, create a string and refer to CSS class names enclosed by single quote (').and separated by space

```
<p [ngClass]="'one'">NgClass with String</p>
<p [ngClass]="'one two'">NgClass with 2 classes</p>
```

```
.one{
    background-color: red;
    color: yellow;
}
.two{
    font-size: 20px;
}
.three{
    letter-spacing: 5px;
}
```

OUTPUT:



NgClass with Array:

- To bind array of CSS classes to NgClass with array of CSS classes, create an array with the CSS classes

```
<p [ngClass]="['one','two','three']">NgClass with Array</p>
```

```
.one{
    background-color: red;
    color: yellow;
}
.two{
    font-size: 20px;
}
.three{
    letter-spacing: 5px;
}
```

output

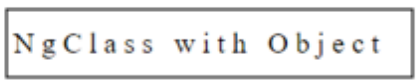
NgClass with Object:

- To bind an Object using { } toNgClass, where the object is the key value pair
- The key is the CSS class name and the value is an expression that returns Boolean value.
- The CSS will be added at run time in HTML element only when if expression will return true.

  If expression returns false then the respective CSS class will not be added

```
<p [ngClass]="{'one':false,'two':true,'three':true}">NgClass with Object</p>
```

```
.one{
    background-color: red;
    color: yellow;
}
.two{
    font-size: 20px;
}
.three{
    letter-spacing: 5px;
}
```



output

NgStyle:

- NgStyle directive is used to set inline styles for the DOM elements.
- Set styles using the NgStyle directive and assign it to an object literal

```
<h2 *ngIf="uname=='Raju' else newtempl"  [ngStyle]="{'background-color':'aqua'}">
  Welcome {{uname}}!!!{{message}}
</h2>
```

- Is more useful when the value is dynamic.
- The values in the object literal can be javascript expressions - they are evaluated and the result is used as the value of the css property

```
<ul >
  <li *ngFor="let course of courseList"
  [ngStyle]="{'color':course.name.includes('o')?'red':'green'}">
    {{course.courseId}}.{{course.name}}
  </li>
</ul>
```

Eg:

```
<div [ngStyle] = 'getStyles()'>Using method as style</div>
```

```
getStyles(){
  let newStyle = {
    'color':false?'red':'orange',
    'background-color':true?'green':'red',
    'font-size.em':true?'3':'4'
  }
  return newStyle;
}
```

Output: Using method as style

SERVICES:

- Reusable code that can be accessed from multiple components
- To create a service **ng g service data**

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

@Injectable()
export class DataService {

  newCourses = ['Java', 'Angular' , 'Node'];
  greetMessage = function() {
    return 'Good day';
  };
  constructor() { }
}
```

MODULES :

Overview of Modules:

- Modules are used to organize an application and extend it with capabilities from external libraries.
- In Angular, libraries are NgModules, like **FormsModule, HttpClientModule, and RouterModule**.
- Third-party libraries like **Material Design, Ionic,** and **AngularFire2** are available as NgModules
- Modules can also add services to the application, where the services can be internal or come from outside sources, like the Angular router and HTTP client.
- Modules can be loaded eagerly when the application starts or loaded lazily asynchronously by the router.

Introduction to NgModule:

- NgModules are a way to organize Angular application

- Every Angular app has at least one module which is the root module (AppModule)
- Bootstrap the root module to launch the application.
- An NgModule is a class marked by the @NgModule decorator.
- NgModules consolidate components, directives, and pipes into cohesive blocks of functionality, each focused on a feature area, application business domain, workflow, or common collection of utilities.

@NgModule:

The metadata has these arrays

declarations: declares the components, directives, pipes that belong to the module.

imports: imports the other modules with the components, directives, and pipes that components in the current module wants

exports: Makes some of those components, directives, and pipes public so that other module's component templates can use them.

providers: Provides services that the other application components can use.

bootstrap: the root component that Angular creates and inserts into the index.html host web page.

Note:

Each bootstrapped component is the base of its own tree of components. Inserting a bootstrapped component usually triggers a cascade of component creations that fill out that tree.

While you can put more than one component tree on a host web page, most applications have only one component tree and bootstrap a single root component.

Example of AppModule :

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';

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

- Import BrowserModule to have browser specific services like DOM rendering
- The root Component is AppComponent in the AppModule

Entry Components :

- An entry component is any component that Angular loads imperatively, (which means you're not referencing it in the template), by type.
- Specify an entry component by bootstrapping it in an NgModule, or by including it in a routing definition.

Two main kinds of entry components:

- The bootstrapped root component.
- A component specified in a route definition.
- **@NgModule** has an entry components array, where the entry components can be specified.

Bootstrapped Entry Components:

- A bootstrapped component is an entry component that Angular loads into the DOM during the bootstrap process (application launch).

```
@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    AppRoutingModule
    ],
  providers: [],
  bootstrap: [AppComponent] // bootstrapped entry component
})
export class AppModule { }
```

Component in a route Definition:

- A route definition refers to a component by its type with

  component: CourseListComponent.

- All router components must be entry components.
- The Compiler can recognize that this is a router definition and automatically add the router component into entry Components array.

```
const routes: Routes = [
  {
    path: 'users',
    component : CoursesListComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Note: Because this would require you to add the component in two places (router and entryComponents.

NgModules in Angular:

NgModule	Import it from	Why you use it
BrowserModule	@angular/platform-browser	To run your app in a browser
CommonModule	@angular/common	To use NgIf, NgFor
FormsModule	@angular/forms	To build template driven forms (includes NgModel)
ReactiveFormsModule	@angular/forms	To build reactive forms

RouterModule	@angular/router	To use RouterLink, .forRoot(), and .forChild()
HttpClientModule	@angular/common/http	To talk to a server

Feature Modules :

- A feature module is used
  - to keep code related to a specific functionality as a separate one
  - for creating a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms
- A feature module collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it shares.
- Create a feature module as

```
ng g module courses
```

Using the feature Module :

Export the component in the feature module to use the template of the component

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

Import the feature module in the root module:

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

Types of feature module :

Feature Module	Usage
Domain Module	• Modules to deliver a user experience dedicated to a particular application eg., editing customer, placing order • Has mostly declarations • Top component alone is exported
Routed Module	• Modules whose top components are the targets of router navigation routes. • All lazy-loaded modules are routed feature modules • Components are not exported and modules are not imported
Routing Module	• Module that provides routing configuration for another module and separates routing concerns from its companion module. • Defines Routes • Does not have declarations • Should be imported only by companion module eg.CourseModule has a routing module as **CourseRoutingModule**

Feature Module	Usage
Service Module	• Modules that provide utility services such as data access and messaging. • Consists entirely of providers and have no declarations. • Angular's HttpClientModule is a good example of a service module. • The root AppModule is the only module that should import service modules.
Widget Module	• Module that makes components, directives, and pipes available to external modules. • Many third-party UI component libraries are widget modules. • Consists entirely of declarations, most of them exported. • Should rarely have providers. • Import widget modules in any module whose component templates need the widgets.

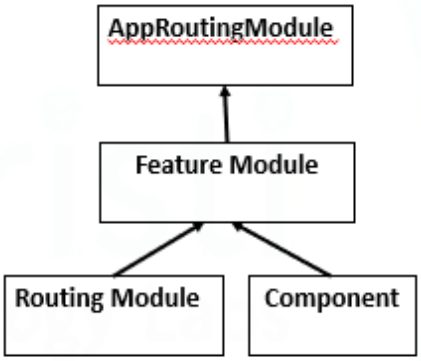
Early Loading of Modules :

- When application starts, eager loading of all the modules happen
- Angular uses the injector to make everything available between modules
- The root application injector makes all of the providers in all of the modules available throughout the app.

Lazy Loading Modules:

- The features module should be wired to **AppRoutingModule**.
- Configure the routes to the feature modules in the **AppRoutingModule**
- The feature module then connects the **AppRoutingModule** to the routing modules

The routing modules then tells the router to load the relevant components.



AppRouting Module:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { CoursesModule } from './courses/courses.module';

const routes: Routes = [
  {
    path: 'users',
    loadChildren: './users/users.module#UsersModule'
  },
  {
    path: 'courses',
    loadChildren: () => CoursesModule
  },
  {
    path: '',
    redirectTo: '',
    pathMatch: 'full'
  }
];
```

forRoot() :

RouterModule.forRoot(routes)

- Added to the AppRoutingModule lets Angular know that this module is a routing module
- ***forRoot()*** specifies that AppRoutingModule is the root routing module.
- It configures all the routes passed to it, gives access to the router directives, and registers the ***RouterService***.
- ***forRoot()*** contains injector configuration which is global like configuring the Router

forChild() :

RouterModule.forChild(routes)

- Added to feature routing modules lets Angular know that the route list is only responsible for providing additional routes
- Is intended for feature modules.
- Can use forChild() in multiple modules.
- ***forChild()*** has no injector configuration, has only directives such as RouterOutlet and RouterLink.

Shared Modules :

- Modules can be shared
- **FormsModule, CommonModule, UtilityComponent, UtilityPipe, UtilityDirective** exported can be used in any other module where the **sharedModule** is imported.
- **FormsModule** is not imported, so cannot be used in this module

- **UtilityComponent, UtilityPipe and UtilityDirective** are imported and so can be used in this module

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { UtilityPipe } from './utility.pipe';
import { UtilityDirective } from './utility.directive';
import { FormsModule } from '@angular/forms';
import { UtilityComponent } from './utility/utility.component';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    UtilityPipe,
    UtilityDirective,
    UtilityComponent],
  exports: [
    CommonModule, FormsModule,
    UtilityDirective, UtilityPipe,
    UtilityComponent]
})
export class SharedModule { }
```

NOTE:

distinction between using another module's component and using a service from another module. Import modules when you want to use directives, pipes, and components. Importing a module with services means that you will have a new instance of that service, which typically is not what you need (typically one wants to reuse an existing service). Use module imports to control service instantiation.

The most common way to get a hold of shared services is through Angular dependency injection, rather than through the module system (importing a module will result in a new service instance, which is not a typical usage).