**Module**

Angular applications are modular in nature. Our application is a collection of many individual modules. Module is a block of code that is created for a specific purpose. Each module exports something as a value a function or a variable or a class that the other modules can import and use.

**Component**

A component is combination of metadata and class. Metadata contains the data used to process the class like metadata contains html elements for the view and the class contains the logic to process the view. Component is kind of directive but it does not live inside other HTML elements.

**Directive**

A directive allows the coder to modify the DoM or extend its behavior.

**Service**

Service is a class that gets data or configuration from the server and after that we inject the service to two components. Any logic that is not related to the view will be residing in the service.

Angular creates only a single instance of a service and provide it to all the components (Singleton Service)

**Routers**

It is responsible for the navigation.

**Dependency Injection**

As a design pattern DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

Defination of dependencies is from inside the constructor to the parameters of the constructor

1. Create the Service = Define a class with @Injectable and import it from modules.
2. Register the Service = Register the service using providers in the proper place like App Component, service like HTTP will be registered in the App Module itself.
3. Declare the dependency = dependecy should be created in the component where the Service is required using constructor(private empSer:EMPSER) and the data is retrieved from the Service using the ngOnInit() lifecycle hook which should be imported and implement the class.

**Synchronous** means the script will send request to the server and after the response comes from the server the script continues with its execution further.

**Asynchronous means** that the script will send a request to the server, and continue it's execution without waiting for the reply. As soon as reply is received a browser event is fired, which in turn allows the script to execute associated actions.

**Observables**

A sequence of item that arrive asynchrously over time. Http calls a single item that is a single response.

1. Make http call from EmpService
2. Receive the Observable and map it
3. Subscribe to the Observable
4. Assign the Emp Data to local variable in view

RxJs-Reactive Extension for Javascript-External library to work with Observables

**HTTP SERVICE**

Import http module in the beginning.

To use http in our service we declare it as a dependency in the constructor

**MAP=** The map operator gets the Response and stores it in a local variable and coverts it into json. The operator is present in RxJS which is to be imported above.

.map((response:Response)=>response.json)

-------------------------------------------------------\

ngOnInit(){

**this.\_servicename.getEmployees().subscribe(resData => this.employees = resData);**

}

We call the getEmployees method from the service which returns a observable, now observable don’t pass information unless subscribed.

**Subscribe** function takes an arrow as argument. It takes the retrieved data in a variable resData and pass it to the view reference variable in the component.

Just like the map operator we have **catch method** that handless error. It takes errorhandler function as argument. The function has

Errorhandler(error:Response)

{

Console.error(error);

Return Observable.throw(error || “Server has error”);// this is thrown to component to show the error in the view

}

In the component class===

**this.\_servicename.getEmployees().subscribe(resData => this.employees = resData, resError => this.errMsg = resError);**

when there is a error, instead of the first argument in subscribe the second argument gets executed

@Injectable

Angular accepts metadata only for classed that has some metadata. And this metadata tells angular what type of instance the injector must provide as a dependency to this class.