Assignment No. 7

TITLE	**Develop and deploy web application using AngularJS.**
PROBLEM STATEMENT /DEFINITION	Develop and deploy one of the following web applications: 1. Online pizza order application 2. Student information system for training & placement department 3. Leave management application 4. Blogging platform 5. Meeting room booking application 6. Exam cell automation application
OBJECTIVE	• To understand in depth working of AngularJS • To use AngularJS to develop any web application.
S/W PACKAGES AND HARDWARE APPARATUS USED	Operating System: open source Fedora 20 Networked computer with Internet access Editor : IDE : Netbeans 8.1 Web browser: Mozilla Firefox, Google Chrome
REFERENCES	• https://angular.io/ • https://angular.io/tutorial • https://www.w3schools.com/angular/angular_intro.asp • https://www.tutorialspoint.com/angularjs/index.htm • https://www.javatpoint.com/angularjs-tutorial
STEPS	Refer to steps below
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Statement • Theory • Design part • Troubleshooting (if any) • Conclusion • References

AngularJS

AngularJS (commonly referred to as "Angular.js" or "AngularJS 1.X") is a JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. The JavaScript components complement Apache Cordova, the framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications.

Angular (commonly referred to as "Angular 4" or "Angular 2") is a TypeScript-based open-source front-end web application platform led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

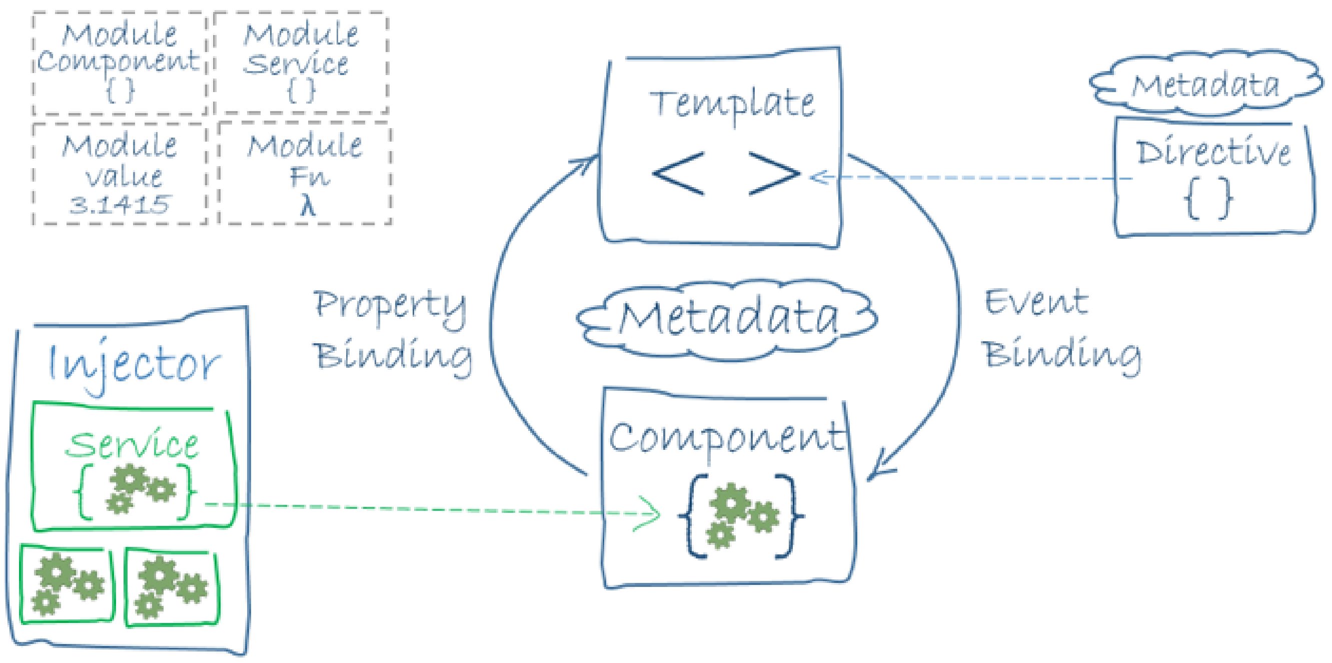
Architecture Overview

Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript.

The framework consists of several libraries, some of them core and some optional.

You write Angular applications by composing HTML *templates* with Angularized markup, writing *component* classes to manage those templates, adding application logic in *services*, and boxing components and services in *modules*.

Then you launch the app by *bootstrapping* the *root module*. Angular takes over, presenting your application content in a browser and responding to user interactions according to the instructions you've provided.



Modules

Angular apps are modular and Angular has its own modularity system called *NgModules*. NgModules are a big deal. Every Angular app has at least one NgModule class, the *root module*, conventionally named `AppModule`. While the *root module* may be the only module in a small application, most apps have many more *feature modules*, each a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule, whether a *root* or *feature*, is a class with an `@NgModule` decorator.

`NgModule` is a decorator function that takes a single metadata object whose properties describe the module. The most important properties are:

- `declarations` - the *view classes* that belong to this module. Angular has three kinds of view classes: components, directives, and pipes.
- `exports` - the subset of declarations that should be visible and usable in the component templates of other modules.
- `imports` - other modules whose exported classes are needed by component templates declared in *this* module.
- `providers` - creators of services that this module contributes to the global collection of services; they become accessible in all parts of the app.

- `bootstrap` - the main application view, called the *root component*, that hosts all other app views. Only the *root module* should set this `bootstrap` property.

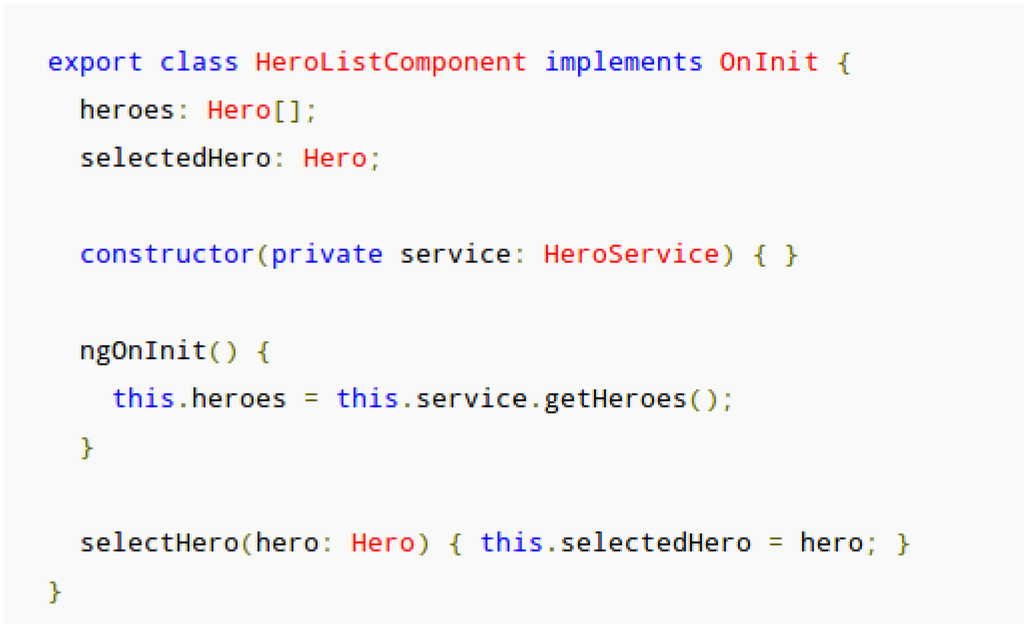
Components

A *component* controls a patch of screen called a *view*.

For example, the following views are controlled by components:

- The app root with the navigation links.
- The list of heroes.
- The hero editor.

You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.



```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

Templates

You define a component's view with its companion **template**. A template is a form of HTML that tells Angular how to render the component.

```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

Although this template uses typical HTML elements like `<h2>` and `<p>`, it also has some differences. Code like `*ngFor`, `{{hero.name}}`, `(click)`, `[hero]`, and `<app-hero-detail>` uses Angular's template syntax.

In the last line of the template, the `<app-hero-detail>` tag is a custom element that represents a new component, `HeroDetailComponent`.

The `HeroDetailComponent` is a *different* component than the `HeroListComponent` you've been reviewing. The `HeroDetailComponent` (code not shown) presents facts about a particular hero, the hero that the user selects from the list presented by the `HeroListComponent`. The `HeroDetailComponent` is a **child** of the `HeroListComponent`.

Metadata

Metadata tells Angular how to process a class. Looking back at the code for `HeroListComponent`, you can see that it's just a class. There is no evidence of a framework, no "Angular" in it at all. In fact, `HeroListComponent` really is *just a class*. It's not a component until you *tell Angular about it*. To tell Angular that `HeroListComponent` is a component, attach **metadata** to the class. In TypeScript, you attach metadata by using a **decorator**. Here's some metadata for `HeroListComponent`

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
/* . . . */
}
```

Here is the `@Component` decorator, which identifies the class immediately below it as a component class. The `@Component` decorator takes a required configuration object with the information Angular needs to create and present the component and its view.

Here are a few of the most useful `@Component` configuration options:

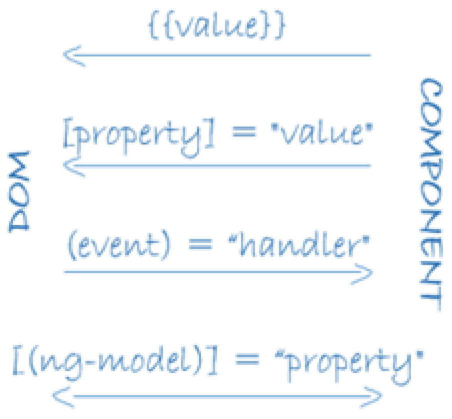
- `selector`: CSS selector that tells Angular to create and insert an instance of this component where it finds a `<app-hero-list>` tag in *parent* HTML. For example, if an app's HTML contains `<app-hero-list></app-hero-list>`, then Angular inserts an instance of the `HeroListComponent` view between those tags.
- `templateUrl`: module-relative address of this component's HTML template, shown above.
- `providers`: array of **dependency injection providers** for services that the component requires. This is one way to tell Angular that the component's constructor requires a `HeroService` so it can get the list of heroes to display.

The metadata in the `@Component` tells Angular where to get the major building blocks you specify for the component. The template, metadata, and component together describe a view. `@Injectable`, `@Input`, and `@Output` are a few of the more popular decorators.

Data binding

Without a framework, you would be responsible for pushing data values into the HTML controls and turning user responses into actions and value updates. Writing such push/pull logic by hand is tedious, error-prone, and a nightmare to read as any experienced jQuery programmer can attest. Angular supports **data binding**, a mechanism for coordinating parts of a template with parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.

As the diagram shows, there are four forms of data binding syntax. Each form has a direction — to the DOM, from the DOM, or in both directions.



Directives

Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by **directives**. A directive is a class with a `@Directive` decorator. A component is a *directive-with-a-template*; a `@Component` decorator is actually a `@Directive` decorator extended with template-oriented features.

Two *other* kinds of directives exist: *structural* and *attribute* directives. They tend to appear within an element tag as attributes do, sometimes by name but more often as the target of an assignment or a binding.

Structural directives alter layout by adding, removing, and replacing elements in DOM.

Services

Service is a broad category encompassing any value, function, or feature that your application needs. Almost anything can be a service. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Examples include:

- logging service
- data service
- message bus
- tax calculator
- application configuration

There is nothing specifically *Angular* about services. Angular has no definition of a service. There is no service base class, and no place to register a service. Yet services are fundamental to any Angular application. Components are big consumers of services.

Dependency injection

Dependency injection is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services. Angular uses dependency injection to provide new components with the services they need.

Angular can tell which services a component needs by looking at the types of its constructor parameters.

Steps to design simple calculator using AngularJS are given below:

Step 1: In Netbeans 8.2 IDE, go to Tools →Plugins and install Angular2 plugin.

Step 2: Download AngularJS 1.6.9 from https://angularjs.org/ and extract the ZIP file.

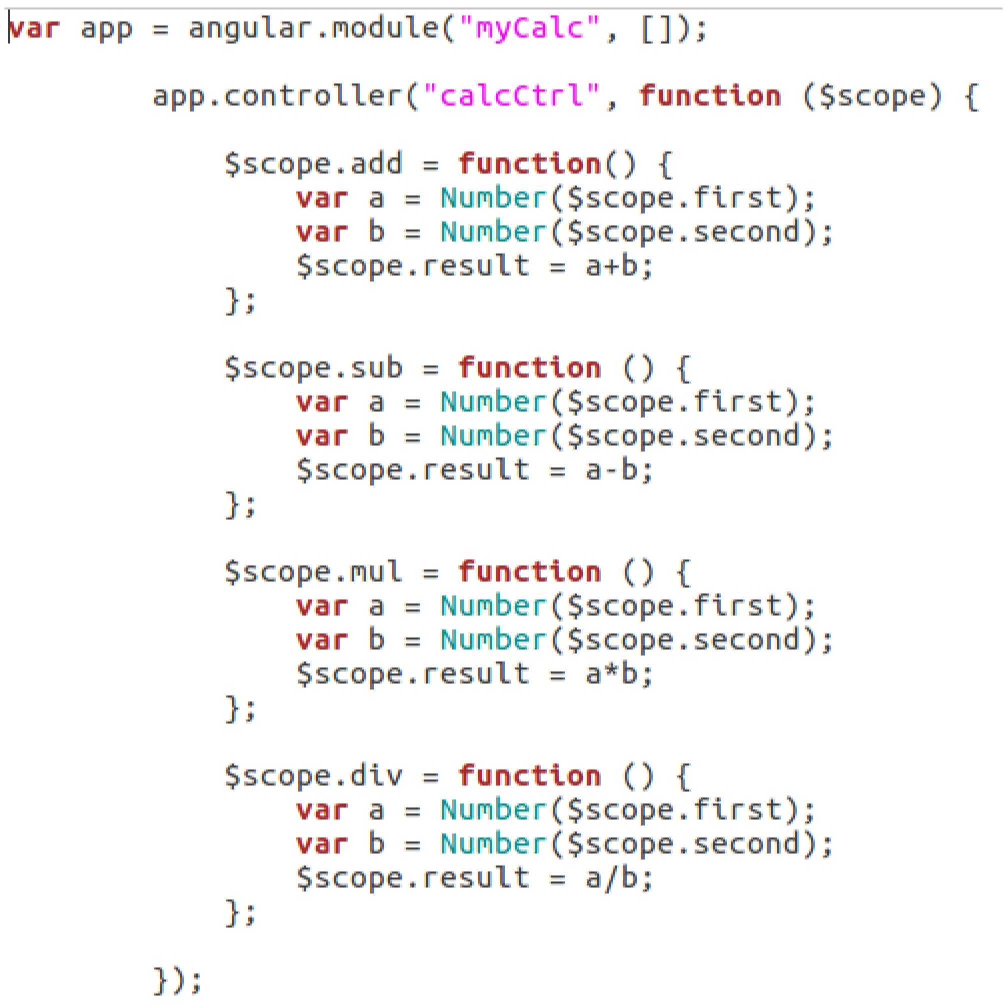
Step 3: Create new HTML5 project and copy angular.js or angular.min.js file from extracted folder to 'Site Root' folder of your project folder.

Step 4: Create a JavaScript file and a CSS file in the 'Site Root' folder of your project folder.

Step 5: Write following HTML code in index.html file. Note that the HTML code contains links to the JavaScript, CSS and angular.js/angular.min.js file.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Calculator</title>
    <script src="angular.min.js"></script>
    <script src="calculator.js" type="text/javascript"></script>
    <link href="newcss.css" rel="stylesheet" type="text/css"/>
</head>

<body ng-app="myCalc"  ng-controller="calcCtrl">

    <div class="calc">

        <p>Calculator</p>
        <div>
            <input type="text"  ng-model="first" placeholder="First Number">
            <input type="text" ng-model="second" placeholder="Second Number">
        </div>
         <input type="text" ng-model="result" placeholder="result">
        <div>
            <input type="button" value="+" class="btn" ng-click="add()">
            <input type="button" value="-" class="btn" ng-click="sub()">
        </div>

        <div>
            <input type="button" value="*" class="btn" ng-click="mul()">
            <input type="button" value="/" class="btn" ng-click="div()">
        </div>

    </div>

</body>
```

Step 6: Write following angularJS code in calculator.js file

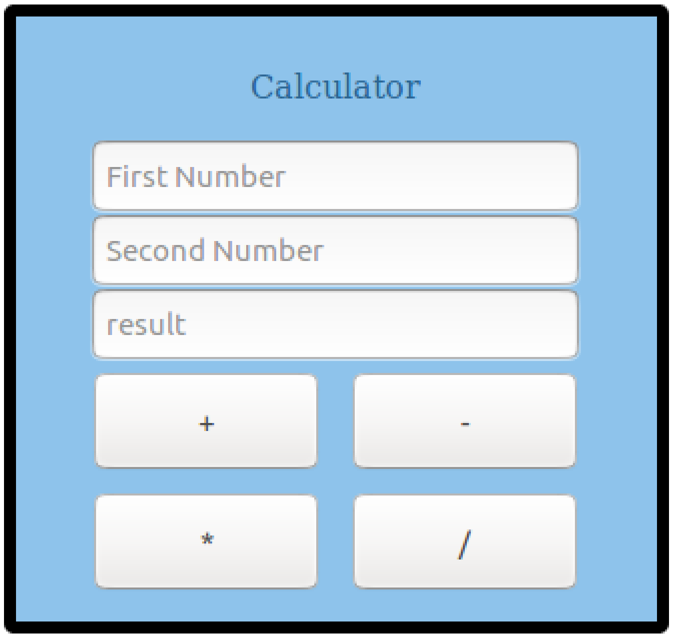


```
var app = angular.module("myCalc", []);

        app.controller("calcCtrl", function ($scope) {

            $scope.add = function() {
                var a = Number($scope.first);
                var b = Number($scope.second);
                $scope.result = a+b;
            };

            $scope.sub = function () {
                var a = Number($scope.first);
                var b = Number($scope.second);
                $scope.result = a-b;
            };

            $scope.mul = function () {
                var a = Number($scope.first);
                var b = Number($scope.second);
                $scope.result = a*b;
            };

            $scope.div = function () {
                var a = Number($scope.first);
                var b = Number($scope.second);
                $scope.result = a/b;
            };

        });
```

Step 7: Write following code in newcss.css file

```
.calc{
    margin: 20px auto;
    padding: 10px;
    border: black solid 6px;
    border-radius:5px;
    text-align: center;
    width: 300px;
    height: auto;
    background: #8ec3eb;
}

.btn {
    margin: 5px;
    width: 114px;
    height: 50px;
}

input[type=text]{
    height:25px;
}

div p{
    color:#2a6592;
}
```

Step 8: Execute the project and you will get the following output.



References:

- https://angular.io/
- https://angular.io/tutorial
- https://www.w3schools.com/angular/angular_intro.asp
- https://www.tutorialspoint.com/angularjs/index.htm
- https://www.javatpoint.com/angularjs-tutorial

Oral Questions:

1. What is data binding in AngularJS?
2. What is scope in AngularJS?
3. What are the controllers in AngularJS?
4. What are the services in AngularJS?
5. What are the filters in AngularJS?
6. Explain directives in AngularJS.
7. Explain templates in AngularJS.
8. What is routing in AngularJS?
9. What is deep linking in AngularJS?
10. What are the advantages of AngularJS?
11. What are the disadvantages of AngularJS?
12. Which are the core directives of AngularJS?
13. Explain AngularJS boot process.
14. What is MVC?
15. Explain ng-app directive.
16. Explain ng-model directive.
17. Explain ng-bind directive.
18. Explain ng-controller directive.
19. How AngularJS integrates with HTML?
20. Explain ng-init directive.
21. Explain ng-repeat directive.
22. What are AngularJS expressions?
23. Explain uppercase filter.
24. Explain lowercase filter.
25. Explain currency filter.
26. Explain filter filter.
27. Explain orderby filter.
28. Explain ng-disabled directive.
29. Explain ng-show directive.
30. Explain ng-hide directive.
31. Explain ng-click directive.
32. How angular.module works?
33. How to validate data in AngularJS?
34. Explain ng-include directive.
35. How to make an ajax call using Angular JS?

36. What is use of $routeProvider in AngularJS?
37. What is $rootScope?
38. What is scope hierarchy in AngularJS?
39. What is a service?
40. What is service method?
41. What is factory method?
42. What are the differences between service and factory methods?
43. Which components can be injected as a dependency in AngularJS?
44. What is provider?
45. What is constant?
46. Is AngularJS extensible?
47. On which types of component can we create a custom directive?
48. What is internationalization? How to implement internationalization in AngularJS?