UNIT-III
AJAX AND ANGULAR JS

INTRODUCTION TO AJAX:

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

AJAX is based on the following open standards −

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages.

JavaScript:

- Loosely typed scripting language.
- JavaScript function is called when an event occurs in a page.
- Glue for the whole AJAX operation.

DOM:

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents.

CSS:

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

XMLHttpRequest:

- JavaScript object that performs asynchronous interaction with the server.

AJAX Browser Support:

All the available browsers cannot support AJAX. Here is a list of major browsers that support AJAX.

- Mozilla Firefox 1.0 and above.
- Netscape version 7.1 and above.
- Apple Safari 1.2 and above.
- Microsoft Internet Explorer 5 and above.
- Konqueror.
- Opera 7.6 and above.

When you write your next application, do consider the browsers that do not support AJAX.

NOTE − When we say that a browser does not support AJAX, it simply means that the browser does not support the creation of Javascript object – XMLHttpRequest object.

Writing Browser Specific Code:

The simplest way to make your source code compatible with a browser is to use *try…catch* blocks in your JavaScript.

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!–
      //Browser Support Code
      function ajaxFunction() {
        var ajaxRequest;  // The variable that makes Ajax possible!

        Try {
          // Opera 8.0+, Firefox, Safari
          ajaxRequest = new XMLHttpRequest();
        } catch (e) {

          // Internet Explorer Browsers
          try {
            ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
          } catch (e) {

            try {
              ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {

              // Something went wrong
              alert("Your browser broke!");
              return false;
            }
          }
        }
      }
      </script>
```

```
<form name = 'myForm'>
  Name: <input type = 'text' name = 'username' /> <br />
  Time: <input type = 'text' name = 'time' />
</form>

</body>
</html>
```

In the above JavaScript code, we try three times to make our XMLHttpRequest object. Our first attempt –

- ajaxRequest = new XMLHttpRequest();

It is for Opera 8.0+, Firefox, and Safari browsers. If it fails, we try two more times to make the correct object for an Internet Explorer browser with –

- ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
- ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");

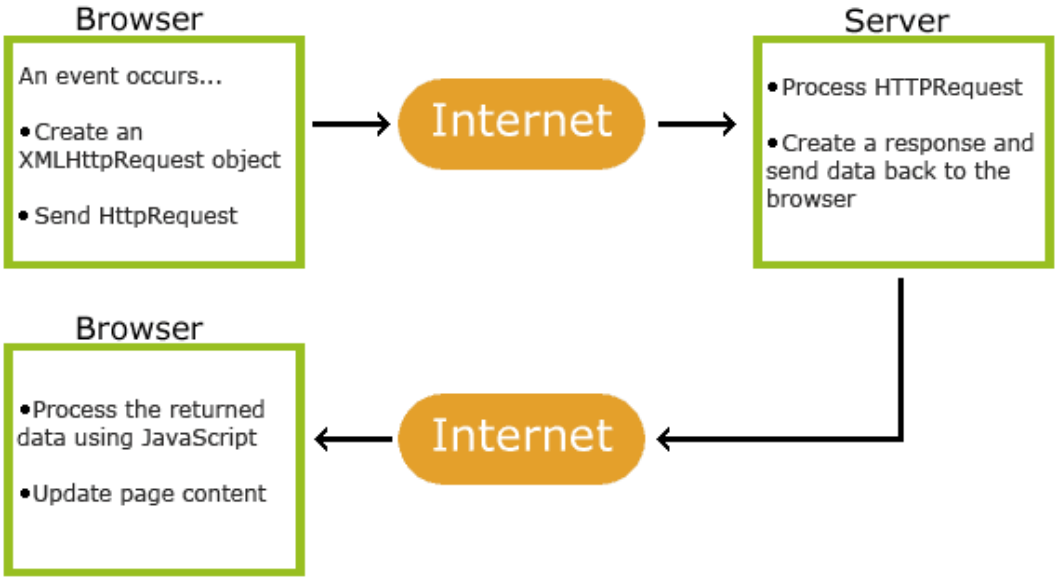
If it doesn't work, then we can use a very outdated browser that doesn't support XMLHttpRequest, which also means it doesn't support AJAX.

Most likely though, our variable ajaxRequest will now be set to whatever *XMLHttpRequest* standard the browser uses and we can start sending data to the server.

AJAX Working Flow:

The following steps are included when you run an AJAX Program:

Step-1: A client event occurs.
Step-2: An XMLHttpRequest object is created.
Step-3: The XMLHttpRequest object is configured.
Step-4: The XMLHttpRequest object makes an asynchronous request to the Webserver.
Step-5: The Webserver returns the result containing XML document.
Step-6: The XMLHttpRequest object calls the callback() function and processes the result.
Step-7: The HTML DOM is updated.



Let us look at these steps one by one.

Step-1: A Client Event Occurs

- A JavaScript function is called as the result of an event.
- Example − *validateUserId()* JavaScript function is mapped as an event handler to an *onkeyup* event on input form field whose id is set to *"userid"*
- <input type = "text" size = "20" id = "userid" name = "id" onkeyup = "validateUserId();">.

Step-2: The XMLHttpRequest Object is Created

```
var ajaxRequest;  // The variable that makes Ajax possible!
function ajaxFunction() {
  try {
    // Opera 8.0+, Firefox, Safari
    ajaxRequest = new XMLHttpRequest();
  } catch (e) {

    // Internet Explorer Browsers
    try {
      ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (e) {
        // Something went wrong
        alert("Your browser broke!");
        return false;
      }
    }
  }
}
```

Configuration of the XMLHttpRequest Object:

In this step, we will write a function that will be triggered by the client event and a callback function processRequest() will be registered.

```
function validateUserId() {
  ajaxFunction();
  // Here processRequest() is the callback function.
  ajaxRequest.onreadystatechange = processRequest;
  if (!target) target = document.getElementById("userid");
  var url = "validate?id=" + escape(target.value);
  ajaxRequest.open("GET", url, true);
  ajaxRequest.send(null);
}
```

Making Asynchronous Request to the Webserver:

Source code is available in the above piece of code. Code written in bold typeface is responsible to make a request to the webserver. This is all being done using the XMLHttpRequest object *ajaxRequest*.

```
function validateUserId() {
  ajaxFunction();
  // Here processRequest() is the callback function.
  ajaxRequest.onreadystatechange = processRequest;
  if (!target) target = document.getElementById("userid");
  var url = "validate?id = " + escape(target.value);
  ajaxRequest.open("GET", url, true);
  ajaxRequest.send(null);
}
```

Assume you enter Zara in the userid box, then in the above request, the URL is set to "validate?id = Zara".

Webserver Returns the Result Containing XML Document:

You can implement your server-side script in any language; however its logic should be as follows.

- Get a request from the client.
- Parse the input from the client.
- Do required processing.
- Send the output to the client.

If we assume that you are going to write a servlet, then here is the piece of code.

```
public void doGet(HttpServletRequest request,
  HttpServletResponse response) throws IOException, ServletException {
  String targetId = request.getParameter("id");
  if ((targetId != null) && !accounts.containsKey(targetId.trim())) {
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("<valid>true</valid>");
  } else {
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("<valid>false</valid>");
  }
}
```

Callback Function processRequest() is Called:

The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the *readyState* of the *XMLHttpRequest* object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

```
 function processRequest() {
  if (req.readyState == 4) {
    if (req.status == 200) {
      var message = ...;
        ...
}
```

Updating the HTML DOM:

This is the final step and in this step, your HTML page will be updated. It happens in the following way −

- JavaScript gets a reference to any element in a page using DOM API.
- The recommended way to gain a reference to an element is to call.

document.getElementById("userIdMessage"),
// where "userIdMessage" is the ID attribute
// of an element appearing in the HTML document

- JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements. Here is an example −

```
<script type = "text/javascript">
  <!--
  function setMessageUsingDOM(message) {
    var userMessageElement = document.getElementById("userIdMessage");
    var messageText;
    if (message == "false") {
      userMessageElement.style.color = "red";
      messageText = "Invalid User Id";
    } else {
      userMessageElement.style.color = "green";
      messageText = "Valid User Id";
    }
    var messageBody = document.createTextNode(messageText);
    // if the messageBody element has been created simple
    // replace it otherwise append the new element
    if (userMessageElement.childNodes[0]) {
      userMessageElement.replaceChild(messageBody, userMessageElement.childNodes[0]);
    } else {
      userMessageElement.appendChild(messageBody);
    }
  }
  -->
</script>
<body>
  <div id = "userIdMessage"><div>
</body>
```

If you have understood the above-mentioned seven steps, then you are almost done with AJAX.

Creating an XMLHttpRequest Object:

The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.

XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.
You already have seen a couple of examples on how to create an XMLHttpRequest object.
Listed below are some of the methods and properties that you have to get familiar with.

XMLHttpRequest Methods

Method Name	Description
abort( )	Cancels the current request.
getAllResponseHeaders()	Returns the complete set of HTTP headers as a string.
getResponseHeader(headerName)	Returns the value of the specified HTTP header.
open(method, URL) open(method, URL, async) open(method, URL, async, userName) open(method, URL, async, userName, password)	Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.
The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

Method Name	Description
send(content)	Sends the request.
setRequestHeader(label, value)	Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties:

Property Name	Description
onreadystatechange	An event handler for an event that fires at every state change.
readyState	The readyState property defines the current state of the XMLHttpRequest object.
responseText	Returns the response as a string.
responseXML	Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
status	Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
statusText	Returns the status as a string (e.g., "Not Found" or "OK").

The following table provides a list of the possible values for the readyState property −

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process.
4	The request is completed.

- **readyState = 0:** After you have created the XMLHttpRequest object, but before you have called the open() method.
- **readyState = 1:** After you have called the open() method, but before you have called send().
- **readyState = 2:** After you have called send().
- **readyState = 3:** After the browser has established a communication with the server, but before the server has completed the response.
- **readyState = 4:** After the request has been completed, and the response data has been completely received from the server.

Handling XML Data with AJAX:

The following example shows how we handle the XML data with the help of the AJAX.

AJAXHtml.HTML:

```
<html>
<head>
<title>Handling XML Data with AJAX</title>
<style>
	table,th,td {
		border : 1px solid black;
		border-collapse: collapse;
	}
	th,td {
		padding: 5px;
	}
</style>
</head>
<body>
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Get my CD collection</button>
<br><br>
<table id="demo">
</table>
<script>
	function loadDoc() {
	  var xhttp = new XMLHttpRequest();
	  xhttp.onreadystatechange = function() {
	    if (this.readyState == 4 && this.status == 200) {
	      myFunction(this);
	    }
```

```
        };
        xhttp.open("GET", "cd_catalog.xml", true);
        xhttp.send();
      }
      function myFunction(xml) {
        var i;
        var xmlDoc = xml.responseXML;
        var table="<tr><th>Artist</th><th>Title</th></tr>";
        var x = xmlDoc.getElementsByTagName("CD");
        for (i = 0; i <x.length; i++) {
          table += "<tr><td>" +
          x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
          "</td><td>" +
          x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
          "</td></tr>";
        }
        document.getElementById("demo").innerHTML = table;
      }
</script>
</body>
</html>
```

cd_catalog.xml:

```
<CATALOG>
    <CD>
        <TITLE>EMPIRE BURLESQUE</TITLE>
        <ARTIST>BOB DYLAN</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>COLUMBIA</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>HIDE YOUR HEART</TITLE>
        <ARTIST>BONNIE TYLER</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS RECORDS</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>GREATEST HITS</TITLE>
        <ARTIST>DOLLY PARTON</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
    <CD>
```

```
        <TITLE>STILL GOT THE BLUES</TITLE>
        <ARTIST>GARY MOORE</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>VIRGIN RECORDS</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1990</YEAR>
    </CD>
</CATALOG>
```

When a user clicks on the "Get CD info" button above, the loadDoc() function is executed. The loadDoc() function creates an XMLHttpRequest object, adds the function to be executed when the server response is ready, and sends the request off to the server. When the server response is ready, an HTML table is built, nodes (elements) are extracted from the XML file, and it finally updates the element "demo" with the HTML table filled with XML data.

Handling JSON with AJAX:

According to the AJAX model, web applications can send and retrieve data from a server asynchronously without interfering with the display and the behavior of the existing page.

Many developers use JSON to pass AJAX updates between the client and the server. Websites updating live sports scores can be considered as an example of AJAX. If these scores have to be updated on the website, then they must be stored on the server so that the webpage can retrieve the score when it is required. This is where we can make use of JSON formatted data.

Any data that is updated using AJAX can be stored using the JSON format on the web server. AJAX is used so that javascript can retrieve these JSON files when necessary, parse them, and perform one of the following operations −

- Store the parsed values in the variables for further processing before displaying them on the webpage.
- It directly assigns the data to the DOM elements in the webpage, so that they are displayed on the website.

Example

The following code shows JSON with AJAX. Save it as **ajax.html** file. Here the loading function loadJSON() is used asynchronously to upload JSON data.

Ajax.html:

```
<html>
  <head>
    <meta content = "text/html; charset = ISO-8859-1" http-equiv = "content-type">
    <script type = "application/javascript">
      function loadJSON(){
        var data_file = "http://www.tutorialspoint.com/json/data.json";
        var http_request = new XMLHttpRequest();
        try{
          // Opera 8.0+, Firefox, Chrome, Safari
          http_request = new XMLHttpRequest();
        }catch (e){
          // Internet Explorer Browsers
          try{
            http_request = new ActiveXObject("Msxml2.XMLHTTP");
```

```
        }catch (e) {
          try{
            http_request = new ActiveXObject("Microsoft.XMLHTTP");
          }catch (e){
            // Something went wrong
            alert("Your browser broke!");
            return false;
          }
        }
        http_request.onreadystatechange = function(){
          if (http_request.readyState == 4  ){
            // Javascript function JSON.parse to parse JSON data
            var jsonObj = JSON.parse(http_request.responseText);
            // jsonObj variable now contains the data structure and can
            // be accessed as jsonObj.name and jsonObj.country.
            document.getElementById("Name").innerHTML = jsonObj.name;
            document.getElementById("Country").innerHTML = jsonObj.country;
          }
        }
        http_request.open("GET", data_file, true);
        http_request.send();
      }
    </script>
    <title>tutorialspoint.com JSON</title>
  </head>
  <body>
    <h1>Cricketer Details</h1>
    <table class = "src">
      <tr>
        <th>Name</th>
        <th>Country</th>
      </tr>
      <tr>
        <td>
         <div id = "Name">Sachin</div>
        </td>
       <td>
        <div id = "Country">India</div>
      </td>
     </tr>
    </table>
    <div class = “central”>
      <button type = “button” onclick = “loadJSON()”>Update Details </button>
    </div>
  </body>
```

</html>

Given below is the input file **data.json**, having data in JSON format which will be uploaded asynchronously when we click the **Update Detail** button.

data.json:

```
{"name": "Brett", "country": "Australia"}
```

Introduction to Angular JS:

Angular JS is an open source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.4.3. Definition of Angular JS as put by its official documentation is as follows −

> Angular JS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology.

- Angular JS is a powerful JavaScript based development framework to create RICH Internet Application (RIA).
- Angular JS provides developers options to write client side application (using JavaScript) in a clean MVC (Model View Controller) way.
- Application written in Angular JS is cross-browser compliant. Angular JS automatically handles JavaScript code suitable for each browser.
- Angular JS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Following are most important core features of Angular JS −

- **Data-binding** − It is the automatic synchronization of data between model and view components.
- **Scope** − These are objects that refer to the model. They act as glue between controller and view.
- **Controller** − These are JavaScript functions that are bound to a particular scope.
- **Services** − Angular JS come with several built-in services for example $https: to make XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Filters** − These select a subset of items from an array and returns a new array.
- **Directives** − Directives are markers on DOM elements (such as elements, attributes, CSS, and more). These can be used to create custom HTML tags that serve as new, custom widgets. Angular JS has built-in directives (ngBind, ngModel...)
- **Templates** − These are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".
- **Routing** − It is concept of switching views.
- **Model View Whatever** − MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer

to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.

- **Deep Linking** − Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
- **Dependency Injection** − AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

Advantages of Angular JS:

- Angular JS provides capability to create Single Page Application in a very clean and maintainable way.
- Angular JS provides data binding capability to HTML thus giving user a rich and responsive experience
- Angular JS code is unit testable.
- Angular JS uses dependency injection and make use of separation of concerns.
- Angular JS provides reusable components.
- With Angular JS, developer can write less code and get more functionality.
- In Angular JS, views are pure html pages, and controllers written in JavaScript do the business processing.

On top of everything, Angular JS applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

Disadvantages of Angular JS:

Though Angular JS comes with lots of plus points but same time we should consider the following points −

- **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable** − If your application user disables JavaScript then user will just see the basic page and nothing more.

The Angular JS Components:

The Angular JS framework can be divided into following three major parts −

- **ng-app** − This directive defines and links an AngularJS application to HTML.
- **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** − This directive binds the AngularJS Application data to HTML tags.

Angular JS Expressions to Bind Data to HTML:

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Angular JS application expressions are pure JavaScript expressions and outputs the data where they are used.

Using numbers:

```
<p>Expense on Books : {{cost * quantity}} Rs</p>
```

Using strings:

```
<p>Hello {{student.firstname + " " + student.lastname}}!</p>
```

Using object:

```
<p>Roll No: {{student.rollno}}</p>
```

Using array

```
<p>Marks(Math): {{marks[3]}}</p>
```

Example:

Following example will showcase all the above mentioned expressions.

testAngularJS.html

```
<html>
   <head>
      <title>AngularJS Expressions</title>
   </head>
   <body>
      <h1>Sample Application</h1>
      <div ng-app = "" ng-init = "quantity = 1;cost = 30; student =
{firstname:'Mahesh',lastname:'Parashar',rollno:101};marks =
[80,90,75,73,60]">
         <p>Hello {{student.firstname + " " + student.lastname}}!</p>
         <p>Expense on Books : {{cost * quantity}} Rs</p>
         <p>Roll No: {{student.rollno}}</p>
         <p>Marks(Math): {{marks[3]}}</p>
      </div>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js
"></script>
   </body>
</html>
```

Angular JS Directives to Bind Data to HTML:

AngularJS directives are used to extend HTML. These are special attributes starting with ng- prefix. We're going to discuss following directives −

- **ng-app** − This directive starts an AngularJS Application.
- **ng-init** − This directive initializes application data.
- **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
- **ng-repeat** − This directive repeats html elements for each item in a collection.

ng-app directive:

ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when web page containing AngularJS Application is loaded. It is also used to load various AngularJS modules in AngularJS Application. In following example, we've defined a default AngularJS application using ng-app attribute of a div element.

```
<div ng-app = "">
   ...
</div>
```

ng-init directive:
ng-init directive initializes an AngularJS Application data. It is used to put values to the variables to be used in the application. In following example, we'll initialize an array of countries. We're using JSON syntax to define array of countries.

```
<div ng-app = "" ng-init = "countries = [{locale:'en-US',name:'United
States'}, {locale:'en-GB',name:'United Kingdom'}, {locale:'en-
FR',name:'France'}]">
   ...
</div>
```

ng-model directive:
This directive binds the values of AngularJS application data to HTML input controls. In following example, we've defined a model named "name".

```
<div ng-app = "">
   ...
   <p>Enter your Name: <input type = "text" ng-model = "name"></p>
</div>
```

ng-repeat directive:
ng-repeat directive repeats html elements for each item in a collection. In following example, we've iterated over array of countries.

```
<div ng-app = "">
   ...
   <p>List of Countries with locale:</p>
   <ol>
      <li ng-repeat = "country in countries">
         {{ 'Country: ' + country.name + ', Locale: ' + country.locale }}
      </li>
   </ol>
</div>
```

Example:
Following example will showcase all the above mentioned directives.

testAngularJS.html:

```
<html>
   <head>
      <title>AngularJS Directives</title>
   </head>
   <body>
      <h1>Sample Application</h1>
      <div ng-app = "" ng-init = "countries = [{locale:'en-US',name:'United
States'}, {locale:'en-GB',name:'United Kingdom'}, {locale:'en-
FR',name:'France'}]">
         <p>Enter your Name: <input type = "text" ng-model = "name"></p>
         <p>Hello <span ng-bind = "name"></span>!</p>
         <p>List of Countries with local: </p>
         <ol>
            <li ng-repeat = "country in countries">
               {{ 'Country: ' + country.name + ', Locale: ' + country.locale
}}
            </li>
         </ol>
```

```
      </div>
      <script src =
      "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"
      ></script>
   </body>
</html>
```

Angular JS Controllers:

AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

```
<div ng-app = "" ng-controller = "studentController">
   ...
</div>
```

Here we've declared a controller **studentController** using ng-controller directive. As a next step we'll define the studentController as follows −

```
<script>
   function studentController($scope) {
      $scope.student = {
         firstName: "Mahesh",
         lastName: "Parashar",
         fullName: function() {
            var studentObject;
            studentObject = $scope.student;
            return studentObject.firstName + " " + studentObject.lastName;
         }
      };
   }
</script>
```

- studentController defined as a JavaScript object with $scope as argument.
- $scope refers to application which is to use the studentController object.
- $scope.student is property of studentController object.
- firstName and lastName are two properties of $scope.student object. We've passed the default values to them.
- fullName is the function of $scope.student object whose task is to return the combined name.
- In fullName function we're getting the student object and then return the combined name.
- As a note, we can also define the controller object in separate JS file and refer that file in the html page.

Now we can use studentController's student property using ng-model or using expressions as follows.

```
Enter first name: <input type = "text" ng-model = "student.firstName"><br>
Enter last name: <input type = "text" ng-model = "student.lastName"><br>
<br>
You are entering: {{student.fullName()}}
```

- We've bounded student.firstName and student.lastname to two input boxes.

- We've bounded student.fullName() to HTML.
- Now whenever you type anything in first name and last name input boxes, you can see the full name getting updated automatically.

Example:
Following example will showcase use of controller.

testAngularJS.html:

```
<html>

   <head>
      <title>Angular JS Controller</title>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></scr
ipt>
   </head>

   <body>
      <h2>AngularJS Sample Application</h2>

      <div ng-app = "mainApp" ng-controller = "studentController">
         Enter first name: <input type = "text" ng-model =
"student.firstName"><br><br>
         Enter last name: <input type = "text" ng-model =
"student.lastName"><br>
         <br>

         You are entering: {{student.fullName()}}
      </div>

      <script>
         var mainApp = angular.module("mainApp", []);

         mainApp.controller('studentController', function($scope) {
            $scope.student = {
               firstName: "Mahesh",
               lastName: "Parashar",

               fullName: function() {
                  var studentObject;
                  studentObject = $scope.student;
                  return studentObject.firstName + " " +
studentObject.lastName;
               }
            };
         });
      </script>

   </body>
</html>
```

Angular JS Forms:
AngularJS enriches form filling and validation. We can use ng-click to handle AngularJS click on button and use $dirty and $invalid flags to do the validations in seemless way. Use novalidate with a form declaration to disable any browser specific validation. Forms controls makes heavy use of Angular events. Let's have a quick look on events first.

Events:
AngularJS provides multiple events which can be associated with the HTML controls. For example ng-click is normally associated with button. Following are supported events in Angular JS.

- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress
- ng-change

ng-click:
Reset data of a form using on-click directive of a button.

```
<input name = "firstname" type = "text" ng-model = "firstName" required>
<input name = "lastname" type = "text" ng-model = "lastName" required>
<input name = "email" type = "email" ng-model = "email" required>
<button ng-click = "reset()">Reset</button>
<script>
   function studentController($scope) {
      $scope.reset = function(){
         $scope.firstName = "Mahesh";
         $scope.lastName = "Parashar";
         $scope.email = "MaheshParashar@tutorialspoint.com";
      }
      $scope.reset();
   }
</script>
```

Validate data:
Following can be used to track error.

- **$dirty** − states that value has been changed.
- **$invalid** − states that value entered is invalid.
- **$error** − states the exact error.

Example:
Following example will showcase all the above mentioned directives.

testAngularJS.html:

```
<html>
   <head>
      <title>Angular JS Forms</title>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></scr
ipt>
      <style>
         table, th , td {
```

```
            border: 1px solid grey;
            border-collapse: collapse;
            padding: 5px;
         }
         table tr:nth-child(odd) {
            background-color: #f2f2f2;
         }
         table tr:nth-child(even) {
            background-color: #ffffff;
         }
      </style>
   </head>
   <body>
      <h2>AngularJS Sample Application</h2>
        <div ng-app = "mainApp" ng-controller = "studentController">
         <form name = "studentForm" novalidate>
            <table border = "0">
               <tr>
                  <td>Enter first name:</td>
                  <td><input name = "firstname" type = "text" ng-model =
                  "firstName" required>
                  <span style = "color:red" ng-show =
"studentForm.firstname.$dirty && studentForm.firstname.$invalid">
                     <span ng-show =
"studentForm.firstname.$error.required">First Name is required.</span>
                  </span>
               </td>
            </tr>
            <tr>
               <td>Enter last name: </td>
               <td><input name = "lastname"  type = "text" ng-model =
"lastName" required>
                  <span style = "color:red" ng-show =
"studentForm.lastname.$dirty && studentForm.lastname.$invalid">
                     <span ng-show =
"studentForm.lastname.$error.required">Last Name is required.</span>
                  </span>
               </td>
            </tr>
            <tr>
               <td>Email: </td><td><input name = "email" type = "email"
ng-model = "email" length = "100" required>
                  <span style = "color:red" ng-show =
"studentForm.email.$dirty && studentForm.email.$invalid">
                     <span ng-show =
"studentForm.email.$error.required">Email is required.</span>
                     <span ng-show =
"studentForm.email.$error.email">Invalid email address.</span>
                  </span>
               </td>
            </tr>

            <tr>
               <td>
                  <button ng-click = "reset()">Reset</button>
               </td>
               <td>
```

```
                    <button ng-disabled = "studentForm.firstname.$dirty &&
                       studentForm.firstname.$invalid ||
studentForm.lastname.$dirty &&
                       studentForm.lastname.$invalid ||
studentForm.email.$dirty &&
                       studentForm.email.$invalid" ng-
click="submit()">Submit</button>
                    </td>
                </tr>

            </table>
         </form>
      </div>

      <script>
         var mainApp = angular.module("mainApp", []);

         mainApp.controller('studentController', function($scope) {
            $scope.reset = function(){
               $scope.firstName = "Mahesh";
               $scope.lastName = "Parashar";
               $scope.email = "MaheshParashar@tutorialspoint.com";
            }

            $scope.reset();
         });
      </script>

   </body>
</html>
```