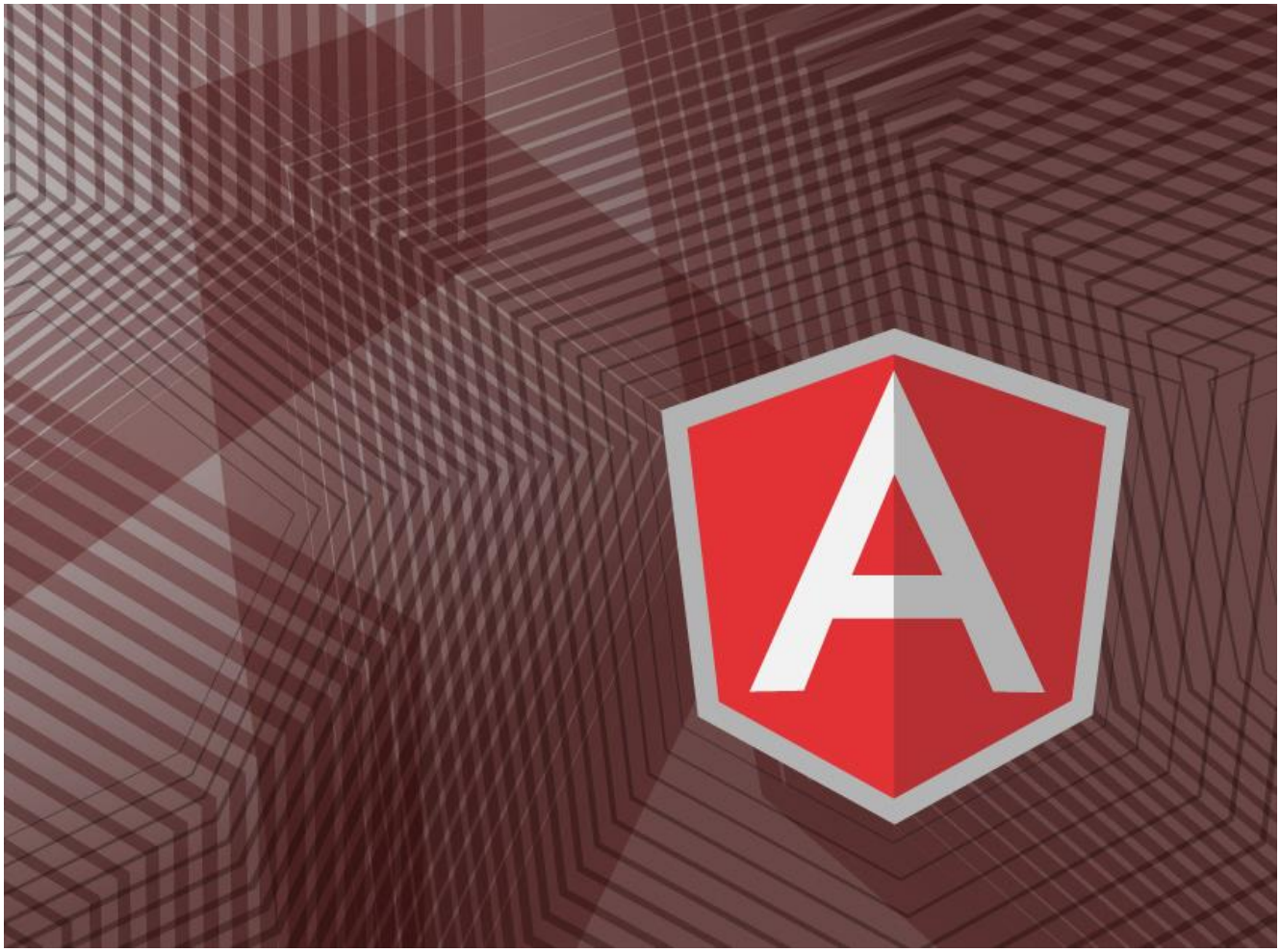


<GO> JS MANUAL

WALCHAND COLLEGE OF ENGINEERING, SANGLI

OVERVIEW OF ANGULARJS

What is AngularJS?

AngularJS 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 AngularJS is as follows −

AngularJS 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 must write. And it all happens within the browser, making it an ideal partner with any server technology.

Features

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

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain.

Core Features

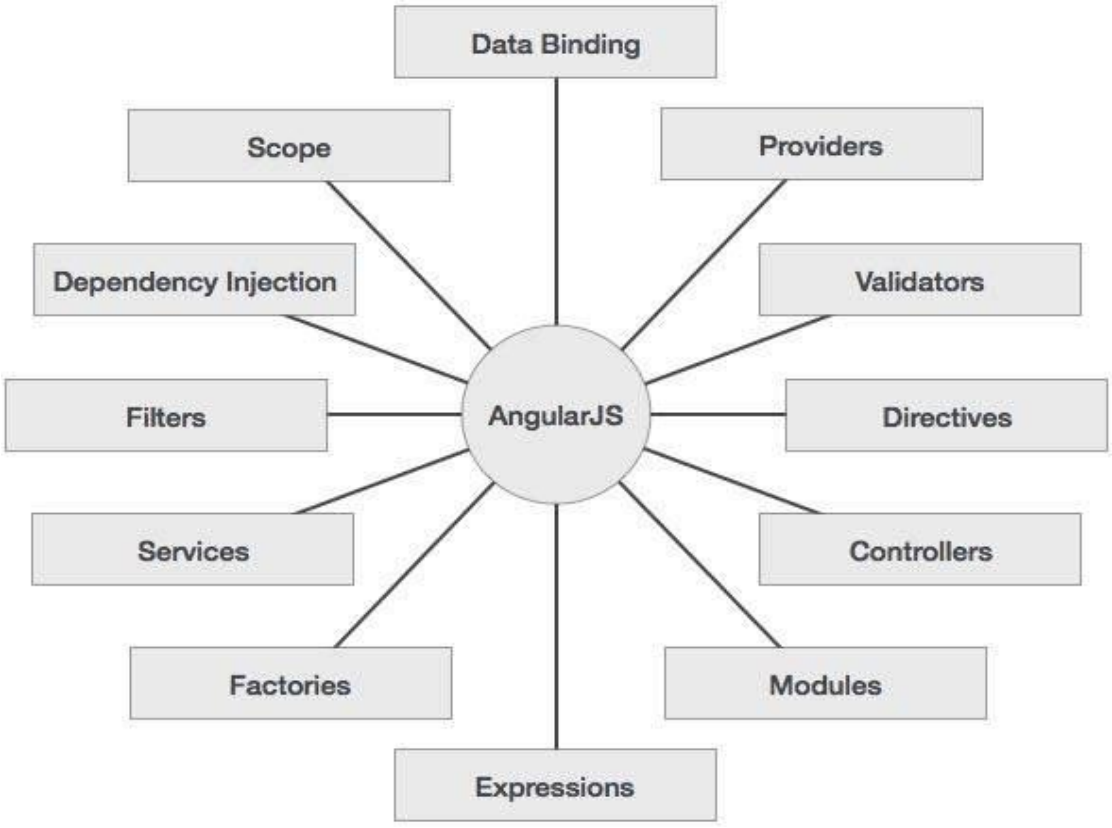
Following are most important core features of AngularJS −

- **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 a glue between controller and view.
- **Controller** − These are JavaScript functions that are bound to a particular scope.
- **Services** − AngularJS come with several built-in services for example $https: to make a 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. AngularJS 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.

Concepts

Following diagram depicts some important parts of AngularJS which we will discuss in detail in the subsequent chapters.



ANGULARJS - ENVIRONMENT SETUP

There are two ways to use AngularJS in your Web App:

1. You can use the online link to include the angularjs library.

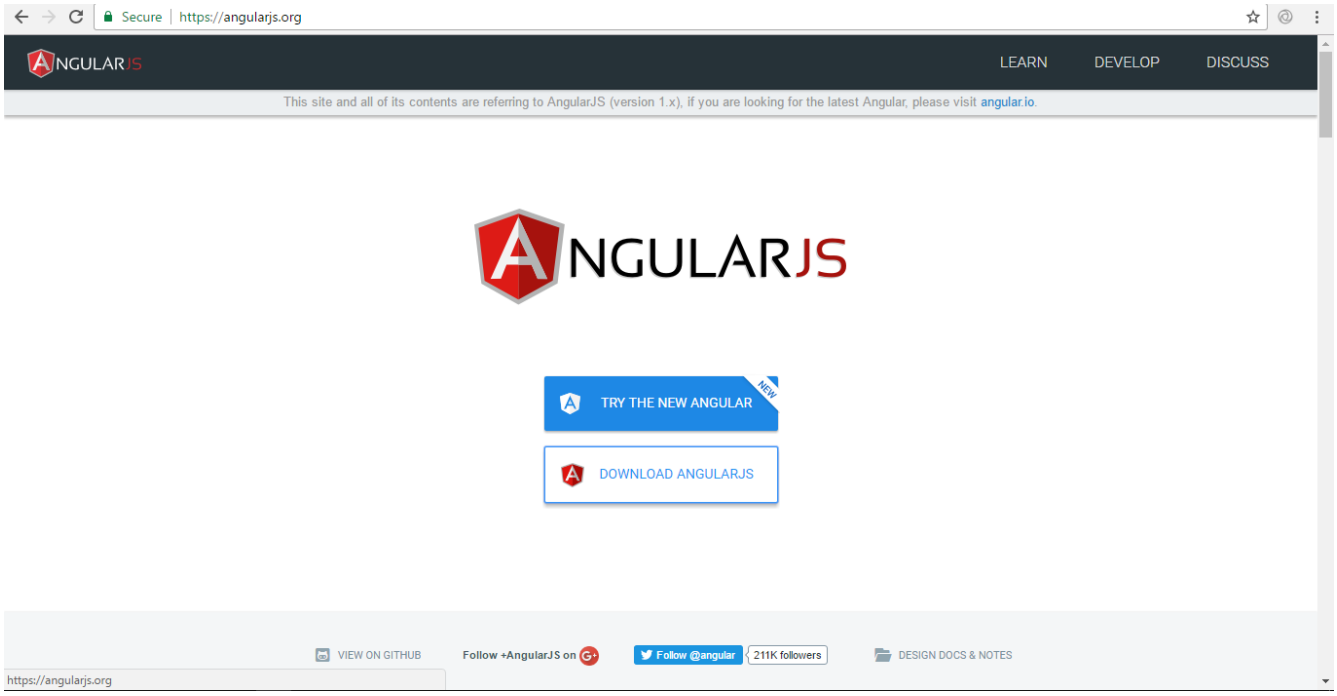
```
<html ng-app>
   <head>
      <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js"></script>
   </head>

   <body>
      <div>
         <label>Name:</label>
         <input type = "text" ng-model = "yourName" placeholder = "Enter a name here">
         <hr />

         <h1>Hello {{yourName}}!</h1>
      </div>

   </body>
</html>
```

2. You can download the file offline and include it into your html file like any other ordinary javascript file.
   To download the file visit www.angularjs.org



Click on Download AngularJS

Download the minified version

And you are ready to use your angular.min.js file in your Web app.

MODEL-VIEW-VIEWMODEL

MODEL
-Represents and holds raw data

Some of this data, in some form, may be displayed in the view

Can also contain logic to retrieve the data from some source

Contains no logic associated with displaying the model

VIEW -User interface

In a web app, it's just the HTML and CSS

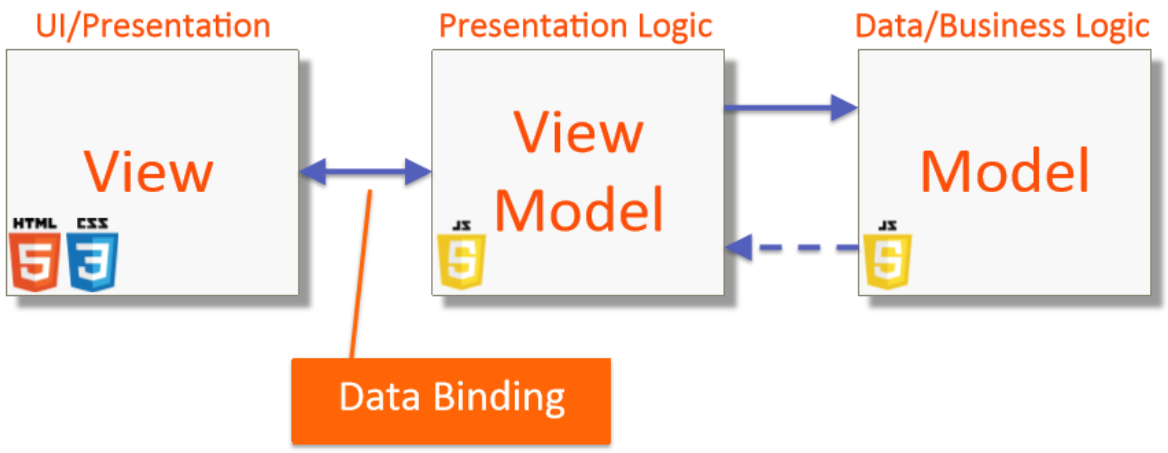
Only displays the data that it is given

Never changes the data

Declaratively broadcasts events, but never handles them

VIEWMODEL Representation of the state of the view ² Holds the data that's displayed in the view ² Responds to view events, aka presentation logic ² Calls other functionality for business logic processing ² Never directly asks the view to display anything

DECLARATIVE BINDER Declaratively binds the model of the ViewModel to the View ² Declaratively means you don't have to write any code • The framework does this "magic" ² Key enabler of the whole MVVM pattern



ANGULARJS - FIRST APPLICATION

STEPS TO CREATE ANGULARJS APPLICATION

Step 1 – Load framework

Being a pure JavaScript framework, It can be added using <Script> tag.

```
<script src = " angular.min.js"></script>
```

Step 2 – Define AngularJS Application using ng-app directive

ng-app − This directive defines and links an AngularJS application to HTML.

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

Step 3 – Define a model name using ng-model directive

ng-model − This directive binds the values of AngularJS application data to HTML input controls.

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

Step 4 – Bind the value of above model defined using ng-bind directive.

ng-bind – This directive binds the AngularJS Application data to HTML tags.

```
<p>Hello <span ng-bind = "name"></span>!</p>
```

STEPS TO RUN ANGULARJS APPLICATION

Use above mentioned three steps in an HTML page.

testAngularJS.htm

```
<html>
   <head>
      <title>AngularJS First Application</title>
   </head>
   <body>
      <h1>Sample Application</h1>
      <div ng-app = "">
         <p>Enter your Name: <input type = "text" ng-model = "name"></p>
         <p>Hello <span ng-bind = "name"></span>!</p>
      </div>
      <script src = "angular.min.js"></script>
   </body>
</html>
```

Open textAngularJS.htm in a web browser. Enter your name and see the result.

ANGULARJS - DIRECTIVES

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 defines the model that is variable to be used in AngularJS.
- **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

ng-model directive defines the model/variable to be used in AngularJS Application. 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>
```

ANGULARJS - EXPRESSIONS

Expressions are used to bind application data to html. Expressions are written inside double braces like {{ expression}}. Expressions behaves in same way as ng-bind directives. AngularJS 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>
```

FILTERS

Filters are used to change modify the data and can be clubbed in expression or directives using pipe character. Following is the list of commonly used filters.

1	uppercase	converts a text to upper case text.
2	lowercase	converts a text to lower case text.
3	currency	formats text in a currency format.
4	filter	filter the array to a subset of it based on provided criteria.
5	orderby	orders the array based on provided criteria.

uppercase filter

Add uppercase filter to an expression using pipe character. Here we've added uppercase filter to print student name in all capital letters.

```
Enter first name:<input type = "text" ng-model = "student.firstName">
Enter last name: <input type = "text" ng-model = "student.lastName">
Name in Upper Case: {{student.fullName() | uppercase}}
```

lowercase filter is used in same fashion

currency filter

Add currency filter to an expression returning number using pipe character. Here we've added currency filter to print fees using currency format.

```
Enter fees: <input type = "text" ng-model = "student.fees">

fees: {{student.fees | currency}}
```

filter filter

To display only required subjects, we've used subjectName as filter.

```
Enter subject: <input type = "text" ng-model = "subjectName">
Subject:
<ul>
  <li ng-repeat = "subject in student.subjects | filter: subjectName">
     {{ subject.name + ', marks:' + subject.marks }}
  </li>
</ul>
```

ANGULARJS - MODULES

AngularJS supports modular approach. Modules are used to separate logics say services, controllers, application etc. and keep the code clean. We define modules in separate js files and name them as per the module.js file. In this example we're going to create two modules.

- **Application Module** − used to initialize an application with controller(s).
- **Controller Module** − used to define the controller.

Application Module

mainApp.js

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

Here we've declared an application **mainApp** module using angular.module function. We've passed an empty array to it. This array generally contains dependent modules.

Controller Module

studentController.js

```
mainApp.controller("studentController", function($scope) {
   $scope.student = {
      firstName: "Mahesh",
      lastName: "Parashar",
      fees:500,
      subjects:[
         {name:'Physics',marks:70},
         {name:'Chemistry',marks:80},
         {name:'Math',marks:65},
         {name:'English',marks:75},
         {name:'Hindi',marks:67}
      ],
      fullName: function() {
         var studentObject;
         studentObject = $scope.student;
         return studentObject.firstName + " " + studentObject.lastName;
      }
   };
});
```

Here we've declared a controller **studentController** module using mainApp.controller function.

Use Modules

```
<div ng-app = "mainApp" ng-controller = "studentController">
   ...
   <script src = "mainApp.js"></script>
   <script src = "studentController.js"></script>

</div>
```

ANGULARJS - 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.

Scope is a special javascript object which plays the role of joining controller with the views. Scope contains the model data. In controllers, model data is accessed via $scope object.

- 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>
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.

ANGULARJS - HTML DOM

Following directives can be used to bind application data to attributes of HTML DOM Elements.

1	ng-disabled	disables a given control.
2	ng-show	shows a given control.
3	ng-hide	hides a given control.
4	ng-click	represents a AngularJS click event.

ng-disabled directive

Add ng-disabled attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

```
<input type = "checkbox" ng-model = "enableDisableButton">Disable Button
<button ng-disabled = "enableDisableButton">Click Me!</button>
```

ng-show directive

Add ng-show attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

```
<input type = "checkbox" ng-model = "showHide1">Show Button
<button ng-show = "showHide1">Click Me!</button>
```

ng-hide directive

Add ng-hide attribute to a HTML button and pass it a model. Bind the model to an checkbox and see the variation.

```
<input type = "checkbox" ng-model = "showHide2">Hide Button
<button ng-hide = "showHide2">Click Me!</button>
```

ng-click directive

Add ng-click attribute to a HTML button and update a model. Bind the model to html and see the variation.

```
<p>Total click: {{ clickCounter }}</p>
<button ng-click = "clickCounter = clickCounter + 1">Click Me!</button>
```

ANGULARJS - VIEWS

AngularJS supports Single Page Application via multiple views on a single page. To do this AngularJS has provided ng-view and ng-template directives and $routeProvider services.

ng-view

ng-view tag simply creates a place holder where a corresponding view (html or ng-template view) can be placed based on the configuration.

Usage

Define a div with ng-view within the main module.

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

ng-template

ng-template directive is used to create an html view using script tag. It contains "id" attribute which is used by $routeProvider to map a view with a controller.

Usage

Define a script block with type as ng-template within the main module.

```
<div ng-app = "mainApp">
   ...
   <script type = "text/ng-template" id = "addStudent.htm">
      <h2> Add Student </h2>
      {{message}}
   </script>
</div>
```

$routeProvider

$routeProvider is the key service which set the configuration of urls, map them with the corresponding html page or ng-template, and attach a controller with the same.

Usage

Define a script block with main module and set the routing configuration.

```
var mainApp = angular.module("mainApp", ['ngRoute']);
mainApp.config(['$routeProvider', function($routeProvider) {
   $routeProvider.
   when('/addStudent', {
      templateUrl: 'addStudent.htm', controller: 'AddStudentController'
   }).
   when('/viewStudents', {
      templateUrl: 'viewStudents.htm', controller: 'ViewStudentsController'
   }).
   otherwise({
      redirectTo: '/addStudent'
   });
}]);
```

Following are the important points to be considered in above example.

- $routeProvider is defined as a function under config of mainApp module using key as '$routeProvider'.
- $routeProvider.when defines a url "/addStudent" which then is mapped to "addStudent.htm". addStudent.htm should be present in the same path as main html page.If htm page is not defined then ng-template to be used with id="addStudent.htm". We've used ng-template.
- "otherwise" is used to set the default view.
- "controller" is used to set the corresponding controller for the view.

Example

testAngularJS.htm

```
<html>
   <head>
      <title>Angular JS Views</title>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
      <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-
route.min.js"></script>
   </head>
   <body>
      <h2>AngularJS Sample Application</h2>
      <div ng-app = "mainApp">
         <p><a href = "#addStudent">Add Student</a></p>
         <p><a href = "#viewStudents">View Students</a></p>
         <div ng-view></div>
         <script type = "text/ng-template" id = "addStudent.htm">
            <h2> Add Student </h2>
            {{message}}
         </script>
         <script type = "text/ng-template" id = "viewStudents.htm">
            <h2> View Students </h2>
            {{message}}   </script>
      </div>
      <script>
         var mainApp = angular.module("mainApp", ['ngRoute']);
         mainApp.config(['$routeProvider', function($routeProvider) {
            $routeProvider.

            when('/addStudent', {
               templateUrl: 'addStudent.htm',
               controller: 'AddStudentController'
            }).

            when('/viewStudents', {
               templateUrl: 'viewStudents.htm',
               controller: 'ViewStudentsController'
            }).

            otherwise({
               redirectTo: '/addStudent'
            });
         }]);

         mainApp.controller('AddStudentController', function($scope) {
            $scope.message = "This page will be used to display add student form";
         });
         mainApp.controller('ViewStudentsController', function($scope) {
            $scope.message = "This page will be used to display all the students";
         });
      </script>
   </body>
</html>
```

ANGULARJS - SERVICES

AngularJS supports the concepts of "Separation of Concerns" using services architecture. Services are javascript functions and are responsible to do a specific tasks only. This makes them an individual entity which is maintainable and testable. Controllers, filters can call them as on requirement basis. Services are normally injected using dependency injection mechanism of AngularJS. AngularJS provides many inbuilt services for example, $https:, $route, $window, $location, $filter etc. Each service is responsible for a specific task for example, $https: is used to make ajax call to get the server data. $route is used to define the routing information and so on. Inbuilt services are always prefixed with $ symbol.

There are two ways to create a service.

- factory
- service

Using factory method

Using factory method, we first define a factory and then assign method to it.

```
var mainApp = angular.module("mainApp", []);
mainApp.factory('MathService', function() {
   var factory = {};
   factory.multiply = function(a, b) {
      return a * b
   }
   return factory;
});
```

Using service method

Using service method, we define a service and then assign method to it. We've also injected an already available service to it.

```
mainApp.service('CalcService', function(MathService){
   this.square = function(a) {
      return MathService.multiply(a,a);
   }
});
```

ANGULARJS - DEPENDENCY INJECTION

Dependency Injection is a software design pattern in which components are given their dependencies instead of hard coding them within the component. This relieves a component from locating the dependency and makes dependencies configurable. This helps in making components reusable, maintainable and testable.

factory

factory is a function which is used to return value. It creates value on demand whenever a service or controller requires. It normally uses a factory function to calculate and return the value.

```
var mainApp = angular.module("mainApp", []);  //define a module

//create a factory "MathService" which provides a method multiply to return
multiplication of two numbers
mainApp.factory('MathService', function() {
   var factory = {};
   factory.multiply = function(a, b) {
      return a * b
   }
   return factory;
});

//inject the factory "MathService" in a service to utilize the multiply method of
factory.
mainApp.service('CalcService', function(MathService){
   this.square = function(a) {
      return MathService.multiply(a,a);
   }
});
…
```

service

service is a singleton javascript object containing a set of functions to perform certain tasks. Services are defined using service() functions and then injected into controllers.

```
//define a module
var mainApp = angular.module("mainApp", []);
...
//create a service which defines a method square to return square of a number.
mainApp.service('CalcService', function(MathService){
   this.square = function(a) {
      return MathService.multiply(a,a);
   }
});
//inject the service "CalcService" into the controller
mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
   $scope.number = defaultInput;
   $scope.result = CalcService.square($scope.number);
   $scope.square = function() {
   $scope.result = CalcService.square($scope.number);
   }
});
```

ANGULARJS - INCLUDES

HTML does not support embedding html pages within html page. To achieve this functionality following ways are used −

- **Using Ajax** − Make a server call to get the corresponding html page and set it in innerHTML of html control.
- **Using Server Side Includes** − JSP, PHP and other web side server technologies can include html pages within a dynamic page.

  Using AngularJS, we can embed HTML pages within a HTML page using ng-include directive.

```
<div ng-app = "" ng-controller = "studentController">
   <div ng-include = "'main.htm'"></div>
   <div ng-include = "'subjects.htm'"></div>
</div>
```

ANGULARJS - AJAX

AngularJS provides $https: control which works as a service to read data from the server. The server makes a database call to get the desired records. AngularJS needs data in JSON format. Once the data is ready, $https: can be used to get the data from server in the following manner −

```
function studentController($scope,$https:) {
var url = "data.txt";
   $https:.get(url).success( function(response) {
      $scope.students = response;
   });
}
```

Here, the file data.txt contains student records. $https: service makes an ajax call and sets response to its property students. *students* model can be used to draw tables in HTML.

Examples

data.txt

```
[
   {
      "Name" : "Mahesh Parashar",
      "RollNo" : 101,
      "Percentage" : "80%"
   },

   {
      "Name" : "Dinkar Kad",
      "RollNo" : 201,
      "Percentage" : "70%"
   },

   {
      "Name" : "Robert",
      "RollNo" : 191,
      "Percentage" : "75%"
   },

   {
```

```
      "Name" : "Julian Joe",
      "RollNo" : 111,
      "Percentage" : "77%"
   }
]
```

testAngularJS.htm

```
<html>
   <head>
      <title>Angular JS Includes</title>
      <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 = "" ng-controller = "studentController">
         <table>
            <tr>
               <th>Name</th>
               <th>Roll No</th>
               <th>Percentage</th>
            </tr>
            <tr ng-repeat = "student in students">
               <td>{{ student.Name }}</td>
               <td>{{ student.RollNo }}</td>
               <td>{{ student.Percentage }}</td>
            </tr>
         </table>
      </div>
      <script>
         function studentController($scope,$https:) {
            var url = "data.txt";

            $https:.get(url).success( function(response) {
               $scope.students = response;
            });
         }
      </script>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"></script>

   </body>
</html>
```

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

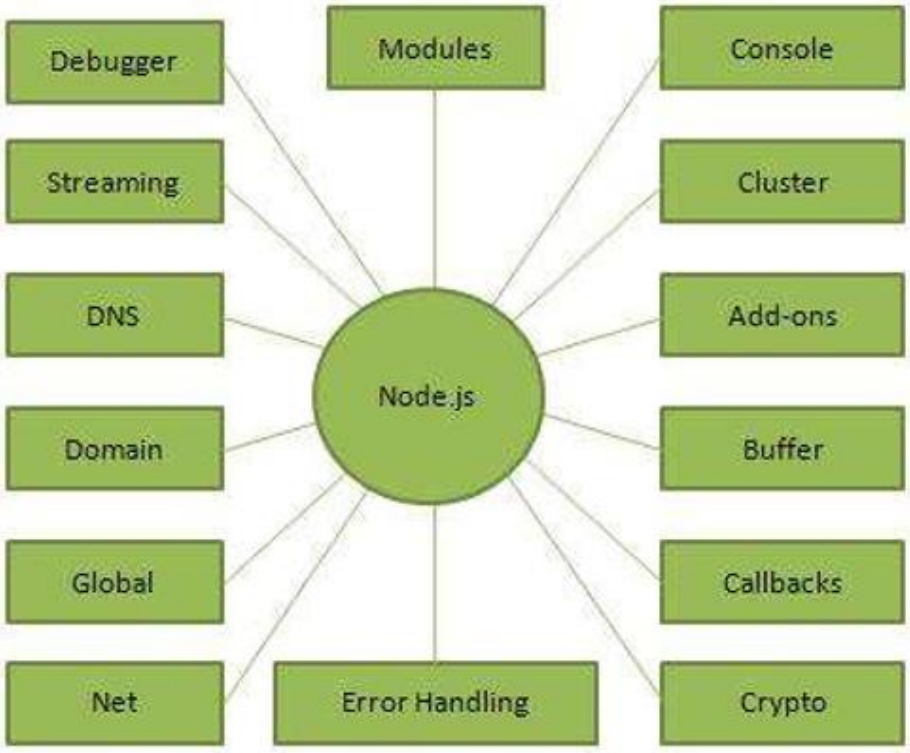
Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent

Node.js = Runtime Environment + JavaScript Library

Features of Node.js Following are some of the important features that make Node.js the first choice of software architects.

- Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- Very Fast − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- Single Threaded but Highly Scalable − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.
- License − Node.js is released under the MIT license.

Concepts



Where to Use Node.js?

Following are the areas where Node.js is proving itself as a perfect technology partner.

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

DOWNLOAD NODE.JS ARCHIVE

Download the latest version of Node.js installable archive file from Node.js Downloads. At the time of writing this tutorial, following are the versions available on different OS.

Windows node-v6.3.1-x64.msi
Linux node-v6.3.1-linux-x86.tar.gz
Mac node-v6.3.1-darwin-x86.tar.gz

INSTALLATION ON WINDOWS

Use the MSI file and follow the prompts to install Node.js. By default, the installer uses the Node.js distribution in C:\Program Files\nodejs. The installer should set the C:\Program Files\nodejs\bin directory in Window's PATH environment variable. Restart any open command prompts for the change to take effect.

Verify Installation:

Executing a File Create a js file named main.js on your machine (Windows or Linux) having the following code.

```
/* Hello, World! program in node.js */

console.log("Hello, World!")
```

Now execute main.js using Node.js interpreter to see the result:

```
$ node main.js
```

If everything is fine with your installation, it should produce the following result:

```
Hello, World!
```

A NODE.JS APPLICATION CONSISTS OF THE FOLLOWING THREE IMPORTANT COMPONENTS:

1. Import required modules: We use the require directive to load Node.js modules.

2. Create server: A server which will listen to client's requests similar to Apache HTTP Server.

Read request and return response: The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Step 1 - Import Required Module We use the require directive to load the http module and store the returned HTTP instance into an http variable as follows:

```
var http = require("http");
```

Step 2 - Create Server We use the created http instance and call http.createServer() method to create a server instance and then we bind it at port 8081 using the listen method associated with the server instance. Pass it a function with parameters request and response. Write the sample implementation to always return "Hello World".

```
http.createServer(function (request, response) {

  // Send the HTTP header
  // HTTP Status: 200 : OK
  // Content Type: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});
```

```
   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);
// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');
```

The above code is enough to create an HTTP server which listens, i.e., waits for a request over 8081 port on the local machine.

Step 3 - Testing Request & Response Let's put step 1 and 2 together in a file called main.js and start our HTTP server as shown below:

```
var http = require("http");
http.createServer(function (request, response) {

   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);
// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');
```

Now execute the main.js to start the server as follows:

```
$ node main.js
```

Verify the Output. Server has started.

```
Server running at http://127.0.0.1:8081/
```

Make a Request to the Node.js Server Open http://127.0.0.1:8081/ in any browser and observe the following result.

Congratulations, you have your first HTTP server up and running which is responding to all the HTTP requests at port 8081.

REPL TEMINAL

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks:

- Read - Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- Eval - Takes and evaluates the data structure.
- Print - Prints the result.
- Loop - Loops the above command until the user presses ctrl-c twice.

The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

Starting REPL REPL can be started by simply running node on shell/console without any arguments as follows.

```
$ node
```

You will see the REPL Command prompt > where you can type any Node.js command:

```
$ node
>
```

Simple Expression Let's try a simple mathematics at the Node.js REPL command prompt:

```
$ node
> 1 + 3
4
```

```
> 1 + ( 2 * 3 ) - 4
3
```

Use Variables You can make use variables to store values and print later like any conventional script. If var keyword is not used, then the value is stored in the variable and printed. Whereas if var keyword is used, then the value is stored but not printed. You can print variables using console.log().

```
$ node
> x = 10
10
> var y = 10
undefined
> x + y
20
> console.log("Hello World")
Hello Workd
undefined
```

REPL supports multiline expression similar to JavaScript. Let's check the following do-while loop in action:

```
$ node
> var x = 0
undefined
> do {
... x++;
... console.log("x: " + x);
... } while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

... comes automatically when you press Enter after the opening bracket. Node automatically checks the continuity of expressions.

NPM

Node Package Manager (NPM) provides two main functionalities:

- Online repositories for node.js packages/modules which are searchable on search.nodejs.org
- Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

NPM comes bundled with Node.js installables after v0.6.3 version. To verify the same, open console and type the following command and see the result:

```
$ npm --version

2.7.1
```

If you are running an old version of NPM, then it is quite easy to update it to the latest version. Just use the following command from

Installing Modules using NPM There is a simple syntax to install any Node.js module:

```
$ npm install <Module Name>
```

For example, following is the command to install a famous Node.js web framework module called express:

```
$ npm install express
```

Now you can use this module in your js file as following:

```
var express = require('express');
```

UNINSTALLING A MODULE

Use the following command to uninstall a Node.js module.

```
$ npm uninstall express
```

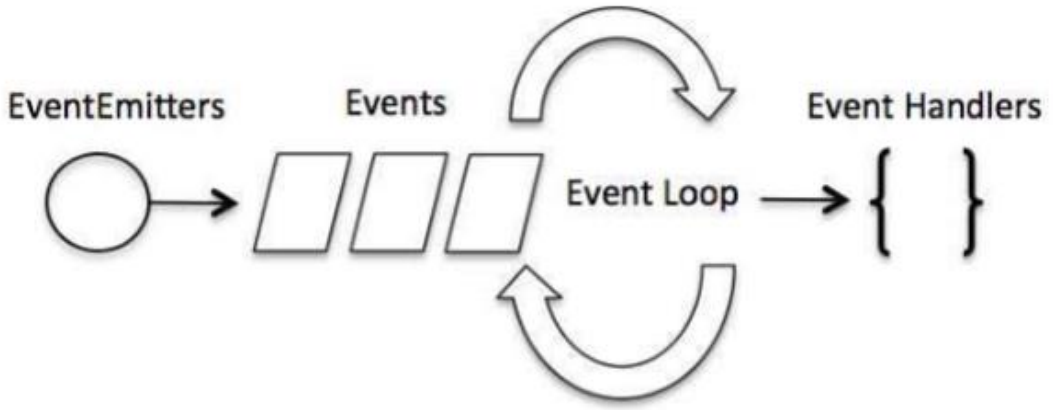
EVENT LOOPS AND EMITTERS

Node.js is a single-threaded application, but it can support concurrency via the concept of event and callbacks. Every API of Node.js is asynchronous and being single-threaded, they use async function calls to maintain concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

Event-Driven Programming

Node.js uses events heavily and it is also one of the reasons why Node.js is pretty fast compared to other similar technologies. As soon as Node starts its server, it simply initiates its variables, declares functions, and then simply waits for the event to occur.

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.



Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as Observers. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners as follows:

```
// Import events module

var events = require('events');

// Create an eventEmitter object

var eventEmitter = new events.EventEmitter();
```

Following is the syntax to bind an event handler with an event:

```
// Bind event and even handler as follows

eventEmitter.on('eventName', eventHandler);
```

We can fire an event programmatically as follows:

```
// Fire an event

eventEmitter.emit('eventName');
```

NET MODULE

Net Module Node.js net module is used to create both servers and clients. This module provides an asynchronous network wrapper and it can be imported using the following syntax.

```
var net = require("net")
```

Example Create a js file named server.js with the following code:

File: server.js

```
var net = require('net');
var server = net.createServer(function(connection) {
   console.log('client connected');
   connection.on('end', function() {
      console.log('client disconnected');
   });
   connection.write('Hello World!\r\n');
   connection.pipe(connection);
});
server.listen(8080, function() {
  console.log('server is listening');
});
```

Now run the server.js to see the result:

```
$ node server.js
```

Verify the Output.

```
server is listening
```

Create a js file named client.js with the following code:

File: client.js

```
var net = require('net');
var client = net.connect({port: 8080}, function() {
   console.log('connected to server!');
});
client.on('data', function(data) {
   console.log(data.toString());
   client.end();
});
client.on('end', function() {
   console.log('disconnected from server');
});
```

Now run the client.js from another terminal to see the result:

```
$ node client.js
```

Verify the Output.

WEB MODULE

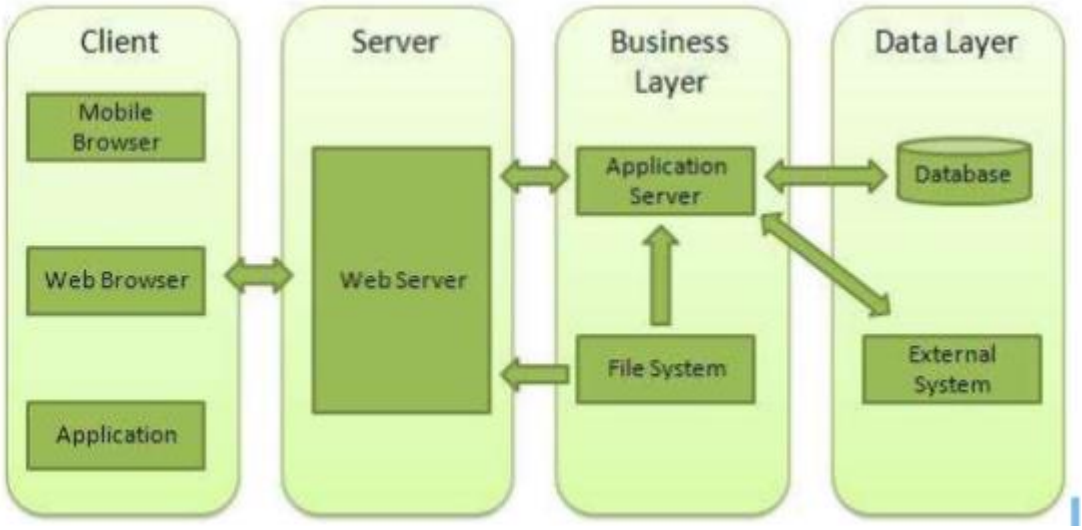
What is a Web Server? A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients. Web servers usually deliver html documents along with images, style sheets, and scripts.

Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.

Apache web server is one of the most commonly used web servers. It is an open source project.

Web application architecture

A Web application is usually divided into four layers:



Client - This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.

Creating a Web Server using Node Node.js provides an http module which can be used to create an HTTP client of a server. Following is the bare minimum structure of the HTTP server which listens at 8081 port.

Create a js file named server.js:

File: server.js

```
var http = require('http');
var fs = require('fs');
var url = require('url');
// Create a server
http.createServer( function (request, response) {
   // Parse the request containing file name
   var pathname = url.parse(request.url).pathname;
```

```
  // Print the name of the file for which request is made.
  console.log("Request for " + pathname + " received.");

  // Read the requested file content from file system
  fs.readFile(pathname.substr(1), function (err, data) {
    if (err) {
      console.log(err);
      // HTTP Status: 404 : NOT FOUND
      // Content Type: text/plain
      response.writeHead(404, {'Content-Type': 'text/html'});
    }else{
      //Page found
      // HTTP Status: 200 : OK
      // Content Type: text/plain
      response.writeHead(200, {'Content-Type': 'text/html'});


      // Write the content of the file to response body
      response.write(data.toString());
    }
    // Send the response body
    response.end();
  });
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

Next let's create the following html file named index.htm in the same directory where you created server.js

File: index.htm

```
<html>
<head>
<title>Sample Page</title>
</head>
<body>
Hello World!
</body>
</html>
```

Now let us run the server.js to see the result:

```
$ node server.js
```

Verify the Output.

```
Server running at http://127.0.0.1:8081/
```

Server - This layer has the Web server which can intercept the requests made by the clients and pass them the response.

Business - This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.

Data - This layer contains the databases or any other source of data.

Make a request to Node.js server Open http://127.0.0.1:8081/index.htm in any browser to see the following result.

Verify the Output at the server end.

```
Server running at http://127.0.0.1:8081/

Request for /index.htm received.
```

Creating a Web client using Node A web client can be created using http module. Let's check the following example.

Create a js file named client.js:

File: client.js

```
var http = require('http');

// Options to be used by request
var options = {
   host: 'localhost',
   port: '8081',
   path: '/index.htm'
};
// Callback function is used to deal with response
var callback = function(response){
   // Continuously update stream with data
   var body = '';
   response.on('data', function(data) {
      body += data;
   });
   response.on('end', function() {
      // Data received completely.
      console.log(body);
   });
}
// Make a request to the server
var req = http.request(options, callback);
req.end();
```

Now run the client.js from a different command terminal other than server.js to see the result:

```
$ node client.js
```

Verify the Output.

```
<html>
<head>
<title>Sample Page</title>
</head>
<body>
Hello World!
</body>
</html>
```

Verify the Output at the server end.

```
Server running at http://127.0.0.1:8081/

Request for /index.htm received.
```

References:

1. tutorialspoint.com
2. coursera.com
3. Wikipedia.org

Hosts

Harshal Khairnar

Minal Parchand

Komal Kotyal

Abhishek Jadhav