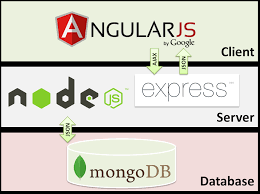
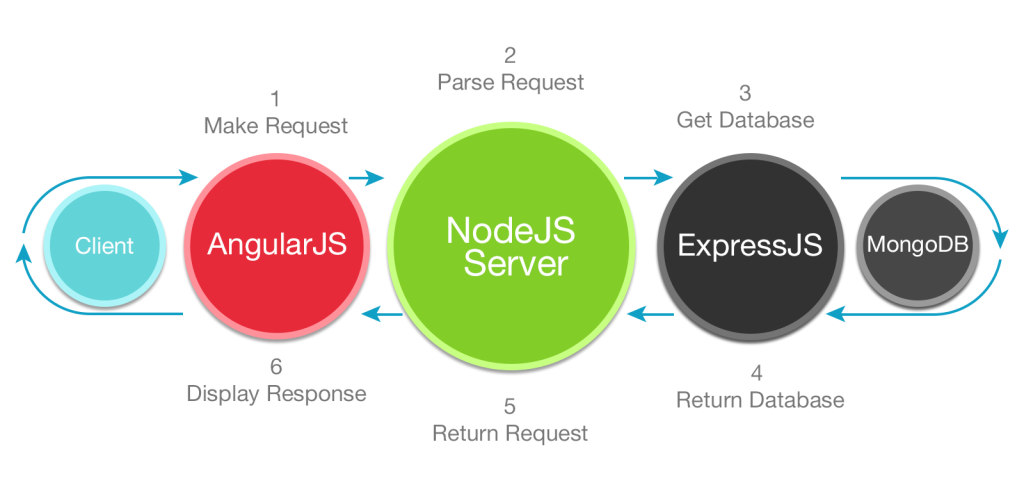
**Day 1 Learning Objectives**

**What is the MEAN Stack? MEAN Stack advantages, Introduction to NodeJS, Benefits of nodejs, NodeJS Installation, Understanding package.json and NPM, Understanding asynch callbacks, Node Modules- BuiltIn, Third Party and Custom modules.**

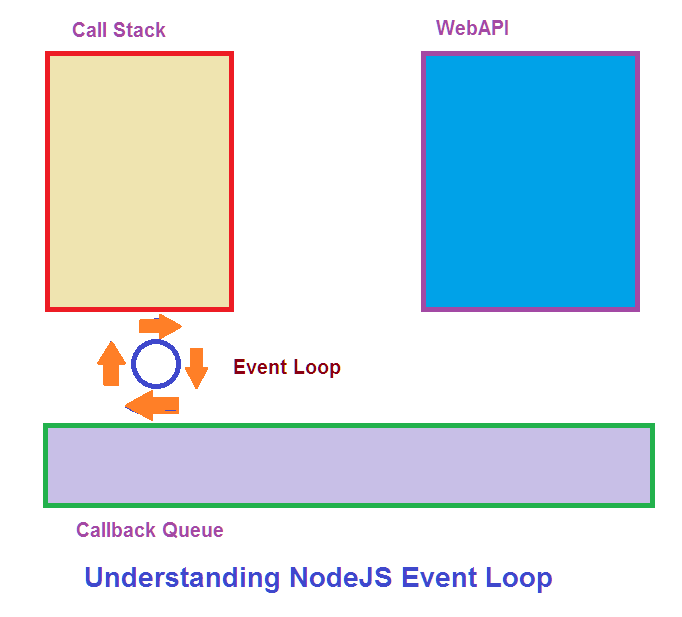






**MEAN stack** is a full-stack JavaScript technology stack that helps you build fast, robust, and maintainable web applications. Combining the powers of MongoDB, Express, AngularJS, and Node.js (hence the acronym), it’s become a popular option for creating dynamic websites.

**NodeJS** is a JavaScript runtime environment built upon Google’s V8 JavaScript engine. The asynch-callback functionality of NodeJS makes it a great choice for server-side applications.



**npm** is –

* a command line program that automatically gets installed when installing NodeJS. It is used to install 3rd Party node modules.
* a repository of 3rd party node modules
* a website

**Some npm commands**:

* npm init
* npm install
* npm install <module-name>
* npm install <modulename> -g
* npm install <modulename> --save

**package.json** is a file in json format that describes any module. Among other things it expresses various types of 3rd-party module dependencies.

Node JS supports **3 types of modules**-

1. Core Modules (like os, http, net, stream, events etc)
2. Third Party Modules (like express, chalk, nodemon, body-parser, express-handlebars etc)
3. Custom Modules- understanding exports

**Day 2 Learning Objectives**

**In-built Node modules- events, stream, fs, net, util, os, http, express, Nodemon and chalk. Committing code to GitHub repository, Deploying on Heroku PaaS cloud**

**Committing code to github**:

1. Prepare to commit code at github
   1. Create a repository at github
   2. In your local project folder create a .gitignore file, which is used to describe the files that we don’t want to push to remote repository. node\_modules/\*
   3. Create a Procfile : Heroku uses Procfile which tells Heroku how to start an app.

web: node app.js

* 1. Create README.md file
  2. Create LICENSE file
  3. var port = process.env.PORT || 3000;

1. Issue the following commands

git init

git add -A

git commit -am "comment"

git push <repository-url> master

**Deploying on Heroku**:

* Create App
* Deploy Method : Select Github
* Connect to Github
* Manual Deploy
  + Master Deploy Branch

**Express**:

* Creating an express application
* Mapping http GET requests
* **Retrieving HTTP GET Request Parameters using** 
  + req.query.nm and
  + req.params.nm

**Http Get Vs Post**

* A GET request can be raised in browser’s address bar.
* The GET method is restricted to send upto 1024 characters only.
* Never use GET method if you have password or other sensitive information to be sent to the server.
* GET can't be used to send binary data, like images or word documents, to the server.
* The data sent by GET method can be accessed using QUERY\_STRING.
* A POST request CANNOT be raised in browser’s address bar.
* The POST method does not have any restriction on data size to be sent.
* The POST method can be used to send ASCII as well as binary data.
* The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.

**Day 3 Learning Objectives**

**Express Template Engine EJS, Express Middleware- static, body-parser and express-session. Bootstrap**

* **Specifying public folder in express app**
  + app.use(express.static(\_\_dirname + "/public"));
* **Understanding body-parser middleware**
  + app.use(bodyparser.json());
  + app.use(bodyparser.urlencoded({extended:false}));
  + req.body.nm
* **Understanding session handling middleware**
  + app.use(session({secret: "secret", resave : true, saveUninitialized : false}));
* **Understanding EJS template engine**
  + No need to ‘require’
  + app.set('view engine', 'ejs');
  + keep template files in views folder. Template files should have .ejs extension.
  + Partials should be kept in the folder views/partials
  + <%include partials/includeHead.ejs %>
  + <%= person%>
  + res.render('landingpage.ejs', {welcomeMessage:person});
* **Bootstrap**
  + **head**

<meta charset="utf-8">

<title>Basic Bootstrap Template</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

* + **Foot**

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">

</script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js">

</script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css">

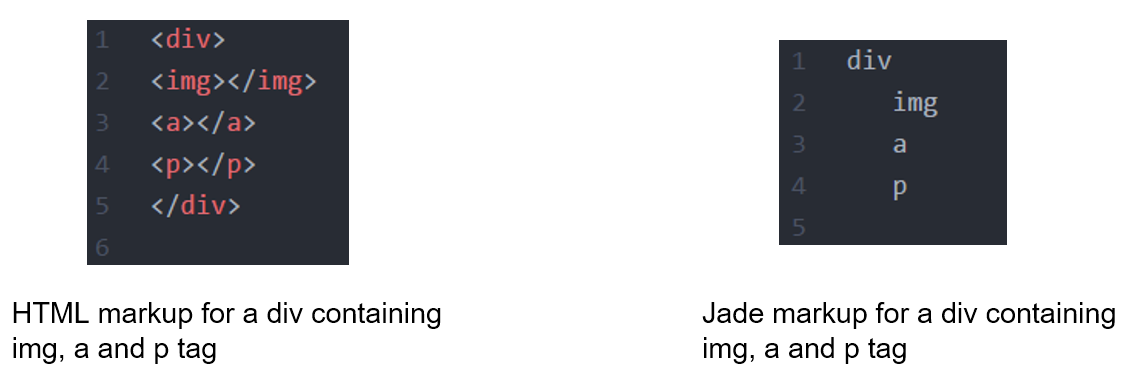
**Day 4 Learning Objectives**

**Template Engines- Jade and Handlebars, Node+Express+Template engine application on heroku, handling HTTP 400 & 500 errors in express applictions, checking heroku logs**

**Day 4 Notes:**

**-Understanding the JADE template engine**

* Unlike EJS and handlebars, JADE is whitespace sensitive and you don’t need to close your HTML tags in JADE. It’s similar to python syntax and it does not use starting and ending curly braces (‘{‘, ‘}’) for code blocks. It uses whitespace indentation, to structure the code blocks.



* In case of div we can even skip the tag name, to specify id attribute we use # symbol and for specifying class attribute we use . symbol



* In JADE you specify the attributes, inside brackets as comma separated values, as shown below





* + In Jade you can specify JavaScript code by using the hyphen (-) symbol before the JavaScript code.
* **HTML to Jade convertor (**[**http://html2jade.org/**](http://html2jade.org/)**)**
* **How to use JADE in code**
  + No need to ‘require’
  + app.set('view engine', 'jade');
  + keep template files in views folder. Template files should have .handlebars extension.
  + Partials should be kept in the folder views/partials
  + Including partials :
    - include partials/includeHead.jade
  + Referring to variables
    - h2= person
    - #{person}
  + res.render('landingpage.jade', {welcomeMessage:person});

**-Understanding Handlebars template engine**

* + var handlebars = require('express-handlebars');
  + app.set('view engine', 'handlebars');
  + app.engine('handlebars', handlebars({}));
  + keep template files in views folder. Template files should have .handlebars extension.
  + Partials should be kept in the folder views/partials
  + {{>includeHead}} – No path, no extension
  + {{cityName}}
  + res.render('landingpage.handlebars', {welcomeMessage:person});
* **Handling HTTP 400 errors (Client Errors) in express applications**

app.use("\*", function(req, res) {

res.status(404);

res.render('404.handlebars', {});

});

* **Handling HTTP 500 errors (Server errors) in express applications**

app.use(function(error, req, res, next) {

console.log(chalk.red('Error : 500::' + error))

res.status(500);

res.render('500.handlebars', {err:error}); // good for knowledge but don't do it

});

* **Checking Heroku logs**-

heroku logs --ps web.1 --app <my-app> -t

**Day 5 Learning Objectives**

**Introduction to MongoDB, NO-SQL, Document Databases, Installation, MongoDB shell, CRUD operations on MongoDB shell, NodeJS programs accessing MongoDB using Native drive**r

**Mongodb and mongo-shell**

In your mongodb installation folder,

* mongod.exe is the mongoDB database system
* mongo.exe is mongoDB shell that is used to manipulate mongodb without writing a program

**Starting and shutting down mongodb**

* Create a data folder for mongodb, say c:\mongodata
* If mongoDB is installed at :\Program Files\MongoDB\Server\3.0\bin then the following commands on command prompt starts your mongodb:

**cd C:\Program Files\MongoDB\Server\3.0\bin**

**mongod -dbpath C:\mongodata**

mongodb shows the following after starting-up:

**waiting for connections on port 27017**

* To shutdown goto command prompt and do this to open mongo-shell:

**cd C:\Program Files\MongoDB\Server\3.0\bin**

**mongo**

* On mongo shell prompt :

**use admin**

**db.shutdownServer();**

[**https://docs.mongodb.com/manual/mongo/**](https://docs.mongodb.com/manual/mongo/)

**CRUD Commands**

db.collection.count() -- Wraps count to return a count of the number of documents in a collection or

matching a query.

db.collection.distinct()-- Returns an array of documents that have distinct values for the specified field.

db.collection.find()-- Performs a query on a collection and returns a cursor object.

db.collection.findOne()-- Performs a query and returns a single document.

db.collection.insert()-- Creates a new document in a collection.

db.collection.remove()-- Deletes documents from a collection.

db.collection.update()Modifies a document in a collection.

**CRUD on mongo shell :**

**use <dbname>**

**show collections**

**show dbs**

**db.createCollection(‘collection-name’);**

**var mycoll = db.getCollection(‘collection-name’);**

**mycoll.find();**

**myColl.insert({vehicle:’car’, model:’maruti’, speed:100});**

**myColl.insert({vehicle:’car’, model:bmw, speed:200});**

**myColl.insert({vehicle:’train’, model:’TGV’, speed:600});**

**myColl.insert({vehicle:’train’, model:’IndianRail’, speed:200});**

**myColl.insert({vehicle:’aircraft’, model:’boeing’, speed:1500});**

**myColl.insert({vehicle:’aircraft’, model:’airbus’, speed:1200});**

**myColl.find();**

**myColl.find({vehicle:’car’});**

**myColl.find({speed:600});**

**myColl.find({speed:{$gt:150}});**

**myColl.update({model:’TGV’}, {$set:{speed:650}}, {multi:false});**

**myColl.remove({model:’airbus’});**

**myColl.count({model:’car’});**

**Day 6 Learning Objectives: NodeJS programs accessing MongoDB using Mongoose driver, MongoLab, Local code accesing MongoLab, NodeJS code on cloud accessing MongoLab DB**

**Manipulating MongoDB data through NodeJS applications:**

* Using mongodb native driver
* Using mongoose driver

**Assignment on day 6:**

Extend no-bcrypt code to have following functionality.

1. Upon successful login the users are shown a list of contents from a collection named "Tech".

"Tech" collection will have two fields in every record, name and description.

2. Provide a mechanism to modify a record in "Tech" collection.

**Help for completing assignment:**

* To show a list of records from Tech collection, use this code hint

<table>

**{{#each recordsArray}}**

<tr>

<td>{{this.tech}}</td>

<td>{{this.description}}</td>

</tr>

**{{/each}**

</table>

**Ref: http://handlebarsjs.com/builtin\_helpers.html**

**Mongolab:**

Monogolab (mlab.com) is cloud service provider for mongodb. Use mongoDB on clous-

* Create an account on mongolab.
* Login to your account and then select “Create New” to create a DB
* Select AWS option for free ‘sandbox’ version of db
* Provide a name to your db
* Create a user for your db: provide user credentials
* To connect using the mongo shell:
* mongo ds015879.mlab.com:15879/edurekadb -u <dbuser> -p <dbpassword>
* To connect using a driver via the standard MongoDB URI
* mongodb://<dbuser>:<dbpassword>@ds015879.mlab.com:15879/edurekadb

**Activities**:

* Connect to mongodb on mongolab from your local mongoDB shell
* Connect to mongodb on mongolab from your local Nodejs application
* Connect to mongodb on mongolab from Nodejs application deployed on heroku
* Deploy the CRUD application on heroku and let it access mongolab’s db

**Day 7 Learning Objectives**

**Summernote, Application using NodeJS, ExpressJS, Handlebars, Bootstrap, MongoDB and its deployment on Heroku**

**Summernote:**

Summernote is rich-text editor that can embedded on any HTML page. It uses JQuery, Bootstrap and font-awesome.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Summarnote Editor Demonstration...</title>

<link href="https://netdna.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.css" rel="stylesheet">

<link href="https://netdna.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.css" rel="stylesheet">

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.js"></script>

<script src="https://netdna.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.js"></script>

<link href="https://cdnjs.cloudflare.com/ajax/libs/summernote/0.7.1/summernote.css" rel="stylesheet">

<script src="https://cdnjs.cloudflare.com/ajax/libs/summernote/0.7.1/summernote.js"></script>

</head>

<body>

<div class="form-group">

<label for="name" class="col-sm-2 control-label">Summarnote</label>

<div class="col-lg-10">

<textarea name="content" id="storyContent" required rows="8" class="form-control"></textarea>

</div>

</div>

<script>

$(document).ready(function() {

$('#storyContent').summernote({

height:100,

toolbar: [

['style', ['bold', 'italic', 'underline', 'clear']],

['font', ['strikethrough', 'superscript', 'subscript']],

['fontsize', ['fontsize']],

['para', ['ul', 'ol', 'paragraph']],

['height', ['height']],

['link', ['linkDialogShow', 'unlink']],

]

});

});

</script>

</body>

</html>

**bcrypt**  is a third party module used to hash a password before storing it in db. Hashing is a one way procedure.

var Bcrypt = require('bcrypt');

var SALT\_WORK\_FACTOR = 10;

var pass = '123456789';

Bcrypt.genSalt(SALT\_WORK\_FACTOR, function(err, salt) {

if(err) {

return console.error(err);

}

Bcrypt.hash(pass, salt, function(err, hash) {

if(err) {

return console.error(err);

}

console.log(hash);

Bcrypt.compare(pass, hash, function(err, isMatch) {

if(err) {

return console.error(err);

}

console.log('do they match?', isMatch);

});

});

});

**Day 8 Learning Objectives**

**HTML5 Server Sockets and Real time application. Using socket.io for server push, bcrypt**

* **Web-Socket** is a protocol providing [full-duplex](https://en.wikipedia.org/wiki/Full-duplex) communication channels over a single TCP connection. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.
* The Web-Socket protocol facilitates **real-time** data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way, a two-way (bi-directional) ongoing conversation can take place between a browser and the server.
* **Socket.IO** is a JavaScript library that enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js.

**Server side:**

Initialisation

var io = socketIO.listen(http-server);

io.sockets.on('connection', function(socket){

});

broadcasting message-

io.sockets.emit('new message',{msg:msg,nick:socket.nickname});

sending message to a single client-

socket.emit('private', {msg:msg,nick:name});

Listeneing to client’s messages

socket.on('ClientMessage', function(data){

});

**Client side:**

Initialisation

<script src="/socket.io/socket.io.js"></script>

var socket = io.connect();

sending message to server

socket.emit('ClientMessage', $('#message').val());

Listening to messages from server:

socket.on('private', function(data){

});

Activities:

* Basic-Chat application
* Breaking news application
* Awesome Chat Application

**Chat-room application:**

http://www.tamas.io/advanced-chat-using-node-js-and-socket-io-episode-1/ - Great

http://www.sitepoint.com/build-node-js-powered-chatroom-web-app-node-mongodb-socket/ --- Good

https://scotch.io/tutorials/a-realtime-room-chat-app-using-node-webkit-socket-io-and-mean

**Day 9 Learning Objectives**

**Grunt, Gulp, Mocha and Chai**

**Grunt** – filesystem based taskrunner

npm install –g grunt-cli

npm install grunt -–save-dev

npm install grunt-contrib-cssmin -–save-dev

npm install grunt-contrib-uglify -–save-dev

npm install grunt-contrib-watch -–save-dev

**Gruntfile.js**

module.exports = function(grunt) {

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

watch: {target: {

files: ['js/\*.js', "css/\*.css"],

tasks: ['uglify', 'cssmin']

}

},

cssmin: {

combine: {

files: {

'css/combined.css': ['css/custom.css', 'css/menu.css']

}

}

},

uglify: {

options: {

mangle: false

},

target: {

files: {

'js/combined.min.js': ['js/dropdown.js','js/util.js']

}

}

}

});

grunt.loadNpmTasks('grunt-contrib-cssmin');

grunt.loadNpmTasks('grunt-contrib-uglify');

grunt.loadNpmTasks('grunt-contrib-watch');

grunt.registerTask('default',['watch']);

}

**Gulp** – a task runner with In-memory processing capabilities

npm install –g gulp

npm install gulp -–save-dev

npm install gulp-cssnano -–save-dev

npm install gulp-concat -–save-dev

npm install gulp-uglify -–save-dev

**gulpfile**

var gulp=require('gulp');

var uglify=require('gulp-uglify');

var concat=require('gulp-concat');

var cssnano=require('gulp-cssnano');

gulp.task('cssmin', function() {

return gulp.src('css/\*.css')

.pipe(concat('combined.min.css'))

.pipe(cssnano())

.pipe(gulp.dest('dist'));

});

gulp.task('uglify', function() {

return gulp.src('js/\*.js')

.pipe(concat('combined.min.js'))

.pipe(uglify({mangle:false}))

.pipe(gulp.dest('dist'));

});

gulp.task('watch', function() {

gulp.watch('js/\*.js',['uglify']);

gulp.watch('css/\*.css',['cssmin']);

});

gulp.task('default', ['watch']);

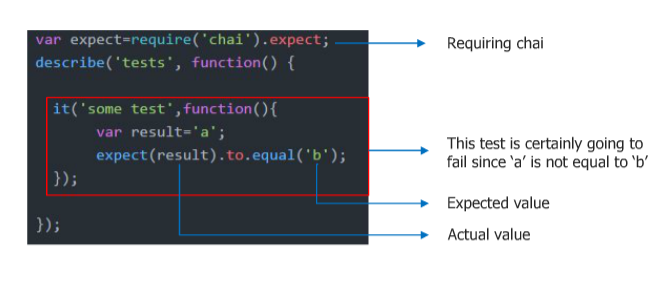
**Mocha** – A flexible test framework for JavaScript. Alternatives are Jasmin and QUnit.

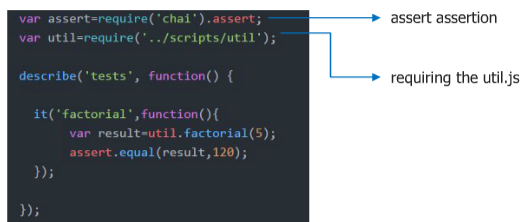
**Chai** – A testcase writing library that allows to write test cases in following styles-

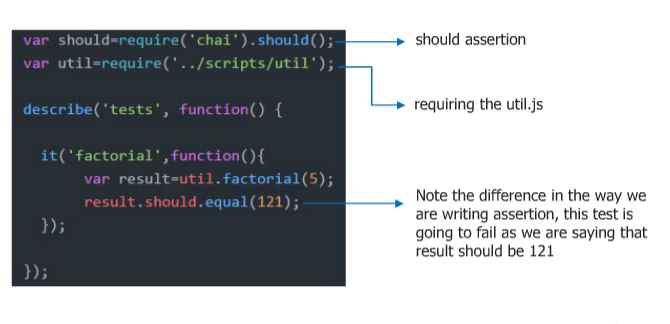
* Should style
* Assert style
* Expect Style

npm install –g mocha

npm install chai –-save-dev







**Day 10 Learning Objectives**

**AngularJS, Directives, Module and Controllers**

Angular.js is a MVW (Model-View-Whatever) open-source JavaScript web framework that facilitates the creation of single-page applications (SPA) and data-driven apps.

**Directives**:

**Directives** are extensions of HTML markups in form of attributes, element names, CSS class and or even HTML comments.

ng-app

ng-model

ng-controller

ng-bind and {{}}

ng-show

ng-hide

ng-init

ng-repeat,

helper variabbles $first, $last, $index, $odd, $even

**Form Related Directives:**

ng-submit

ng-minlength

ng-disabled

ng-messages

ng-messages-include

ng-required and required

**Form States CSS Classes**

$valid ng-valid

$invalid ng-invalid, ng-invalid-required, ng-invalid-minlength

$dirty ng-dirty

$pristine ng-pristine

**Two way data binding**



**Modules and Controllers:**

Angular.js **modules** are a way to encapsulate different parts of your application (directives, controllers, factories, …) and reuse them in other places.

Angular.js **controllers** are code that “control” certain sections of view containing DOM elements in which they are declared. They encapsulate the behavior, callbacks and glue models with views.

To create a module:

angular.module()

To create a controller:

<module>.controller()

**Day 11 Learning Objectives**

**Routing (ngRoute and ng-view), Forms, filters- sorting, converting, Services- $http, $cookies, $locale, $cacheFactory, Custom Services,**

**inbuilt services:** $http, $location, $routeProvider, $stateProvider, $urlRouterProvider, $routeParams

**---Custom Services---**

* **Using module.service() to create a custom service**

//defining a service using module.service. begin with creating an //independent function

function addService(){

var xNum = 0;

this.currNum = function(){return xNum},

this.incrNum = function(){xNum++; return xNum;}

}

//service: add the function created to module.service

rApp.service('addService', [addService]);

//controller

rApp.controller('addController', ['addService', function(addService) {

this.incrementNum = function(){return addService.incrNum();};

this.currentNum = function(){return addService.currNum();};

}]);

* **Using module.factory()to create a custom service**

rApp.factory('addService', [function(){

var xNum = 0;

return {

currNum : function(){return xNum},

incrNum : function(){xNum++; return xNum;}

};

}]);

//controller

rApp.controller('addController', ['addService', function(addService) {

this.incrementNum = function(){return addService.incrNum();};

this.currentNum = function(){return addService.currNum();};

}]);

**Difference between a Controller and a Service**

* All views receive a separate instance of a controller.
* Services a singleton objects. Each service is created only once.

**Writing Single page applications**

* Use ngRoute if you do not need a nested view
* Use ui-router if you need a nested view

**----Using ngRouter----**

$routeProvider

// route for the home page

.when('/', {

templateUrl : 'pages/home.html',

controller : 'mainController'

})

// route for the home page

.when('/home', {

templateUrl : 'pages/home.html',

controller : 'mainController'

})

**----Using ui-router----**

$stateProvider

.state('home', {

url: '/home',

templateUrl: 'pages/home.html'

})

.state('home.list', {

url: '/list', //this can be commented

templateUrl: 'pages/home-list.html',

controller: function($scope) {

$scope.dogs = ['Leo', 'Tiger', 'Dug'];

}

})

.state('home.paragraph', {

url: '/paragraph', //this can be commented

template: 'I could sure use a drink right now.'

})

**Day 12 Learning Objectives**

**Custom Filters, Custom Directives in AngularJS, Application using Full tech stack**

**Custom Filters**

[**https://github.com/Akhil1968/my-custom-filters**](https://github.com/Akhil1968/my-custom-filters)

<html>

<head>

<title>Checkmark Filter</title>

</head>

<body ng-app="filtersApp">

<h4> A custom filter example </h4>

<div ng-controller="phoneController as ctrl">

<ul ng-repeat="phone in ctrl.phones">

<li>{{phone.name}} --- {{phone.amoled | checkmark}}</li>

</ul>

</div>

<script

src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.11/angular.js">

</script>

<script type="text/javascript">

var app = angular.module('filtersApp', []);

app.filter('checkmark', function() {

return function(input) {

return input ? '\u2713' : '\u2718';

};

});

app.controller('phoneController', [function(){

this.phones = [

{name:'MotoX', amoled:true},

{name:'MotoE', amoled:false},

{name:'MotoG', amoled:false}

];

}]);

</script>

</body>

</html>

**Custom Directive**

[**https://github.com/Akhil1968/my-custom-directives**](https://github.com/Akhil1968/my-custom-directives)

<!DOCTYPE html>

<html ng-app="myCustomeDirectiveApp">

<body >

<hello>an Element</hello> Element

<div data-hello>a data Attribute</div> data-Attribute

<div hello> an Attribute</div> Attribute

<!-- directive: hello --> Comment

<p class="hello"></p> Class

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular.min.js">

</script>

<script>

var app = angular.module('myCustomeDirectiveApp', []);

app.directive('hello', [function () {

return {

restrict: 'CEMA', //CEMA C: class, E: element, M: coMments, A: attributes

replace: true, // replaces original content with template

template: '<span><br>Good Morning</span>'

}

}]);

</script>

</body>

</html>

**RESTful API**

REST stands for Representational State Transfer. It is an architecture that allows client-server communication through a uniform interface.

HTTP RESTful API’s are compose of:

HTTP methods, e.g. GET, PUT, DELETE, POST.

POST, GET, PUT, DELETE 🡪 CREATE, READ, UPDATE, DELETE 🡪 CRUD.

**Using ngResource for calling RESTful APIs from Angular code**

var rApp = angular.module('meanRouteApp', ['ngRoute', 'ngResource']);

rApp.factory('TechRestService', function($resource) {

return $resource('/api/tech/:id'); // The full endpoint address

});

**Code within controller**

self.techRecords = TechRestService.query();

**Complete MEAN Stack application: mean-crud**

[**https://github.com/Akhil1968/mean\_crud\_rest**](https://github.com/Akhil1968/mean_crud_rest)

**Bower**

[Bower](http://bower.io/), a package manager makes it easy to manage all your application’s front-end dependencies. Bower manages components that contain HTML, CSS, JavaScript, fonts or even image files.

Packages installed by bower are placed in a bower\_components directory. Command bower init creates a bower.json file.

npm install -g bower

bower install

bower install <package>

bower install <package> --save

bower install <package>#<version>

[bower init](https://bower.io/docs/creating-packages/#bowerjson)

In your HTML you write this to refer to front-end packages installed by bower-

<script src="bower\_components/jquery/dist/jquery.min.js"></script>

**Testing Angular applications**

**URL:** [**https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1**](https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1)

**Test Runner**- Karma, Protractor

**Testing Framework**- Jasmin, Mocha

**Test case writing libraries**- Expect.js, should.js, assert.js, Chai.js

Angular Team recommends using **Karma** and **Jasmin** for testing angular applications.

npm install karma-cli -g

npm install karma --save-dev

npm install karma-jasmine --save-dev

npm install jasmine-core --save-dev

npm install karma-chrome-launcher --save-dev

npm install karma-spec-reporter --save-dev

**File karma.conf.js --- (You can use karma init command to create this file interactively)**

module.exports = function(config) {

config.set({

basePath: '',

frameworks: ['jasmine'],

files: [

'public/bower\_components/angular/angular.js',

'public/bower\_components/angular-route/angular-route.js',

'public/bower\_components/angular-resource/angular-resource.js',

'public/bower\_components/angular-mocks/angular-mocks.js',

'public/script.js',

'public/scriptSpec.js'

],

reporters: ['spec'],

exclude: [],

port: 8080,

// logging level :

// LOG\_DISABLE || LOG\_ERROR || LOG\_WARN || LOG\_INFO || LOG\_DEBUG

logLevel: config.LOG\_INFO,

// enable / disable watching file for re-executing tests

autoWatch: false,

browsers: ['Chrome'],

// Continuous Integration mode

// if true, it capture browsers, run tests and exit

singleRun: true

});

};

**File scriptSpec.js ---**

describe('---Testing controllers in MEAN-CRUD-REST application---', function() {

// Instantiate a new version of my module before each test

beforeEach(module('meanRouteApp'));

var myctrl;

var backendMock;

var anArray = [

{tech: 'Java', description: 'Oracle Sun Language'},

{tech: 'JavaScript', description: 'ECMAScript5'}

];

// Before each unit test, instantiate a new instance

// of the controller

beforeEach(inject(function($controller, $httpBackend) {

backendMock = $httpBackend;

backendMock.expectGET('/api/tech').respond(anArray);

myctrl = $controller('consoleController');

}));

it('---testing readAll through $httpBackend---- ', function() {

expect(myctrl.techRecords.length).toEqual(0);

backendMock.flush();

expect(myctrl.techRecords.length).toEqual(2);

expect(myctrl.techRecords[0].tech).toEqual('Java');

//expect(myctrl.message).toEqual(true);

}); //it

afterEach(function() {

// Ensure that all expects set on the $httpBackend

// were actually called

backendMock.verifyNoOutstandingExpectation();

// Ensure that all requests to the server

// have actually responded (using flush())

backendMock.verifyNoOutstandingRequest();

});

});

**Running Karma ---**

**karma start**

**Yeoman**

Yeoman is a collection of three tools - called Yo, Grunt, and Bower - that allow developers to concentrate on building the functionality of an application, rather than working to build its infrastructure. [Yo](https://www.github.com/yeoman/yo) handles scaffolding of the application, while [Grunt](http://www.gruntjs.com/) takes care of build processes and [Bower](http://www.bower.io/) manages the application's dependencies.

npm install -g yo

npm install -g generator-angular

yo angular myapp

* The generator asks a series of questions to determine how you want the app configured and what packages you'd like to include — would you like to Bootstrap.
* Then, it will generate the scaffolding.
* Along with the app scaffold, the generator will also create JSON manifest files for Bower and npm packages, a gruntfile with build instructions, and test conf files for unit and end-to-end testing with Karma.
* We spend most of our time in the /myapp/app directory.