MEAN

MongoDB, Express, AngularJS, Node.js

Required Software: Node.js, MongoDB, Git
Required Global Node Packages: bower, grunt, karma, protractor, http-server (must be installed with npm –g install)
Required Local Node Packages: express, express-session, body-parser, mongojs, passport, passport-local, mongoose
Required Bower components: angular, angular-resource, bootstrap, angular-bootstrap, angular-loading-bar

MEAN from Scratch

MEAN: Node.js

- Create server.js

```
var http = require('http');
var server = http.createServer(function(req, res){
  res.writeHead(200, {'Content-type': 'text/plain'});
  res.end('Hello world\n');
});

server.listen(8000);
console.log('Server is ready!');
```

- Run Node HTTP server.

```
>> node server.js
```

MEAN: Express

- Create project directory with `package.json` in it

```
{
  "name": "mean-scratch",
  "description": "Mean from scratch",
  "dependencies": {
    "express": "3.x"
  }
}
```

- Install node modules by running following commands at project directory

```
>> npm install
```

- Modify server.js to provide Express-based server, run it.

This example does not utilize View Engine of Express. Many of them are available including `Jade`, `EJS`, `Swig`

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

//use 'static' middleware
app.use(express.static(__dirname + '/app'))

app.get('/', function(req, res){
  res.send('Hello world')
})

app.listen(8000)
```

MEAN: MongoDB

- Install & Run MongoDB

```
>> mongod --dbpath=data/
```

- Run MongoDB shell (`/bin/mongo`) to manipulate your database.

```
>> show databases
>> use todo-mean
>> db.todo.insert({title: 'first in list',
        priority: 4})
>> db.todo.find()
>> tmp = db.todo.findOne({title: 'first in list'})
>> tmp.priority = 5
>> db.update({_id: tmp._id}, tmp)
```

- Install MongoDB driver and related libraries

`--save` options will insert a line corresponding to installed components in `package.json`

```
>> npm install --save mongojs
```

* Modify server.js to serve to-do list.

```
var mongojs = require('mongojs')
...
var db = mongojs.connect("mongodb://localhost/todo-
mean", ['todo'])
...
app.get('/todo', function(req, res){
  db.todo.find({}, function(err, docs){
    res.send(docs)
  })
})
```

- Modify server.js to accept new to-do item.

Following code can only accept to-do item in JSON format (request with `Content-Type: application/json`)

```
var bodyParser = require('body-parser')
...
app.use(bodyParser.json())
...
app.post('/todo', function(req, res){
  db.todo.insert(req.body,
      function(err, docs){
         res.send(docs)
      }
  )
})
```

MEAN: AngularJS

MVC

Render to-do list from hard-coded data.

```
<html ng-app="todoApp">
<head>
  <title>Todo List</title>
  <script type="text/javascript"
src="bower_components/angular/angular.js"></script>
</head>
<body ng-controller="todoCtrl">
  <div>
    <input type="text" ng-model="instance.title"/>
    <input type="number"
         ng-model="instance.priority"/>
    <button ng-click="add()">Add</button>
  </div>
  <ul ng-repeat="todo in todos | orderBy:'priority'">
    <li>{{todo.title}}</li>
  </ul>
</body>
<script type="text/javascript">
  var app = angular.module('todoApp', []);
  app.controller('todoCtrl', function($scope){
    $scope.instance = {};
    $scope.todos = [
       {title: 'A', priority: 3},
       {title: 'B', priority: 4},
       {title: 'C', priority: 1}];
    $scope.add = function(){
       $scope.todos.push($scope.instance);
       $scope.instance = {};
    };
  });
</script>
</html>
```

Tips: Most of JavaScript libraries can be downloaded using `bower`. "`.bowerrc`" can be created to control where bower components are stored.

```
{
    "directory": "app/bower_components"
}
```

ngResource

Provide interaction support with RESTful services via the `$resource` service.
To use ngResource, `angular-resource.js` must be included.

```
<script type="text/javascript">
  var app = angular.module('todoApp',
              ['ngResource']);

  app.factory('Todo', function($resource){
      return $resource('todo/:_id', {_id: '@_id'});
  });

  app.controller('todoCtrl', function($scope, Todo){
    $scope.instance = {};
    $scope.todos = Todo.query();
    $scope.add = function(){
      var todo = new Todo($scope.instance)
      todo.$save(function(_todo){
        $scope.todos.push(_todo);
        $scope.instance = {}
      });
    };
  });
</script>
```

Node.js and Express

Modularity

Node.js leverage most of its modularity features on Require.js

- Create a JavaScript source file
- Use require('...') to import another module
- Assign variable or function to exports.* to make them available to others

For example, move database connection logics to `lib/config/db.js`

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

exports.db = mongojs.connect(...);
```

`require('mongojs')` is no longer required in `server.js`, `db` can be assigned directly.

```
var db = require('./lib/config/db').db;
```

Passport

Support many providers such as Facebook, Twitter, Google and Local
Additional `npm` packages must be installed: `express-session, passport, passport-local`. Most of them are in form of express middleware. Order of applying middleware is matter.

```
var cookieParser = require('cookie-parser');
var passport = require('passport');

app.use(cookieParser());
app.use(bodyParser.json());
app.use(express.session({secret: 'SUPER*cat'}));
app.use(passport.initialize());
app.use(passport.session());
```

For modularity purpose, following codes are written in `lib/config/auth.js`

```
var db = require('./db').db,
    passport = require('passport'),
    LocalStrategy =
         require('passport-local').Strategy;
var ObjectId = require('mongojs').ObjectId

// Serialize sessions
passport.serializeUser(function(user, done) {
  done(null, user._id);
});

passport.deserializeUser(function(id, done) {
  console.log('deserializing => ' + id);
  db.user.findOne({ _id: new ObjectID(id) },
      function (err, user) {
        done(err, user);
      });
});

// Use local strategy
passport.use(new LocalStrategy({
    usernameField: 'username',
    passwordField: 'password'
  },
  function(username, password, done) {
    db.user.findOne({ username: username },
      function (err, user) {
      if (err) {
        return done(err);
      }
      if (!user || user.password != password) {
        return done(null, false, {
          'errors': {
            'password': {
               type: 'Password is incorrect.'
            }
          }
        });
      }
      return done(null, user);
    });
  }
));
```

To protect resources, create a method as an express middleware.

```
exports.ensureAuthenticated = function
  (req, res, next) {
  if (req.isAuthenticated()) { return next(); }
  res.send(401);
}
```

Create routes for authentication.

```
var auth = require('lib/config/auth');
...
app.get('/home', auth.ensureAuthenticated,
function(req, res){
  res.send('Hello home');
});

app.post('/login', passport.authenticate('local'),
function(req, res){
    res.json({username: req.user.username})
});

app.get('/logout', function(req, res){
  req.logout();
  res.json({result: 'ok'})
});
```

See more from angular-passport.

```
>> git clone https://github.com/DaftMonk/angular-
passport
```

Mongoose

Mongoose provides a straight-forward, schema-based solution to modeling your application data and includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

var kittySchema = mongoose.Schema({
    name: String
}, {collection: 'Kitten'})
kittySchema.methods.speak = function () {
  var greeting = "Meow name is " + this.name;
  console.log(greeting);
}
mongoose.model('Kitten', kittySchema)
```

In any module that requires mongoose can access collections.

```
var Kitten = mongoose.model('Kitten');
Kitten.find({}, function(err, docs){...});
new Kitten(...).save();  //or Kitten.save(...);
```

Name can be required.

```
name: {type: 'String', unique: true, required: true}
```

AngularJS

Routing & Multiple Views

Template

The `$route` service is usually used in conjunction with the `ngView` directive.

```
<html lang="en" ng-app="phonecatApp">
...
  <script src=".../angular.js"></script>
  <script src=".../angular-route.js"></script>
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
<body>
  <div ng-view></div>
</body>
```

Route Configuration

```
var phonecatApp = angular.module('phonecatApp', [
  'ngRoute']);

phonecatApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/phones', {
        templateUrl: 'partials/phone-list.html',
        controller: 'PhoneListCtrl'
      }).
      when('/phones/:phoneId', {
        templateUrl: 'partials/phone-detail.html',
        controller: 'PhoneDetailCtrl'
      }).
      otherwise({
        redirectTo: '/phones'
      });
  }]);
```

Link to `/phones` can be put at `a href`. In `PhoneDetailCtrl`, `phoneId` can be accessed from injectable `$routeParams`.

```
<a href="#/phones"> List </a>
```

Testing

Unit Testing with Karma

Create karma configuration files by running this command at test directories.

```
>> karma init
```

Angular provides a service, $controller, which will retrieve your controller by name.

```
describe('PhoneListCtrl', function(){

  beforeEach(module('phonecatApp'));

  it('should create "phones" model with 3 phones',
        inject(function($controller) {
    var scope = {},
    ctrl = $controller('PhoneListCtrl',
              $scope:scope});

    expect(scope.phones.length).toBe(3);
  }));
});
```

Run the test.

```
>> karma start karma.conf.js
```

End-to-End Testing with Protractor

- After installing protractor globally. Download selenium web driver.

```
>> webdriver-manager update
```

- Create test/protractor.conf.js

```
exports.config = {
  allScriptsTimeout: 11000,
  specs: [ 'e2e/*.js'],
  capabilities: {'browserName': 'chrome'},
  baseUrl: 'http://localhost:8000/app/',
  framework: 'jasmine',
  jasmineNodeOpts: { defaultTimeoutInterval: 30000 }
};
```

Test case using Protractor API

```
describe('PhoneCat App', function() {
  describe('Phone list view', function() {

    beforeEach(function() {
      browser.get('app/index.html');
    });

    it('should filter the phone list as user types
into the search box', function() {

      var phoneList = element.all(
            by.repeater('phone in phones'));
      var query = element(by.model('query'));

      expect(phoneList.count()).toBe(3);

      query.sendKeys('nexus');
      expect(phoneList.count()).toBe(1);

      query.clear();
      query.sendKeys('motorola');
      expect(phoneList.count()).toBe(2);
    });
  });
});
```

- Run e2e test

```
>> protractor test/protractor.conf.js
```

You may try do this using angular-seed project.

```
>> git clone https://github.com/angular/angular-seed
```

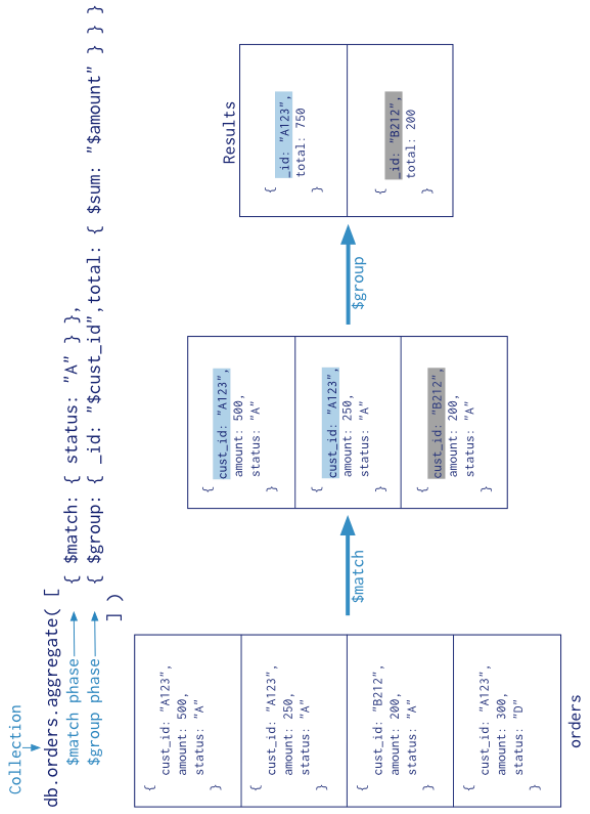
In case, you develop static web-page only. You can run it without node knowledge.

```
>> http-server -a localhost -p 8000
```

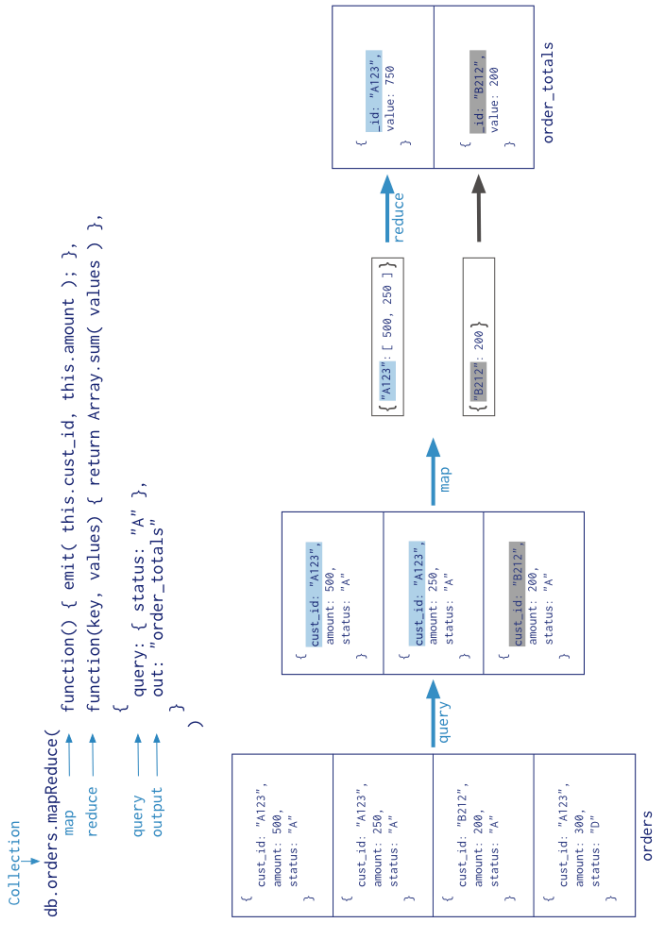
MongoDB

Aggregation Modalities

Aggregation Pipelines (since MongoDB 2.2)



Map Reduce



Data Manipulation Techniques

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier

   db.users.insert (                    ← collection
      {
        name: "sue",                    ← field: value
        age: 26,                        ← field: value
        status: "A"                     ← field: value
      }                                 } document
   )

    db.users.update(                    ← collection
       { age: { $gt: 18 } },            ← update criteria
       { $set: { status: "A" } },       ← update action
       { multi: true }                  ← update option
    )

       db.users.remove(                 ← collection
          { status: "D" }               ← remove criteria
       )
```

Query Selectors

Comparison:	$gt, $gte, $in, $lt, $lte, $ne, $nin
Logical:	$or, $and, $not, $nor

For element, evaluation, Geospatial, Array, Projection Operations, consult API documents.

Helpful queries

```
find( {age: {$gt: 18}}).limit(3).sort({age: 1} )

find( { type: { $in: [ 'food', 'snacks' ] } } )

find( { type: 'food', $or: [ { qty: { $gt: 100 } },
             { price: { $lt: 9.95 } } ] } )

find( {name: /^Fluff/ } )

find( { tags: 'fruit' } ) //match an array element
```

Upsert

If the document contains an `_id` field, then the `save()` method calls the `update()` method with the upsert option and a query on the `_id` field. If a document does not exist with the specified `_id` value, the `save()` method results in an insertion of the document.

MEAN Stack

There are two well-known Full-stack JavaScript Framework for MEA which are created by the same person (`Amos Haviv`). Mean.io is now maintained by `Linnovate` while Mean.js is done by `Amos Haviv`.

Answers in Stack Overflow by `SDude` in Apr 21, 2014.

Scaffolding and Boilerplate Generation

`Mean.io` uses a custom cli tool named 'mean' (very original name)
`Mean.js` uses Yeoman Generators

Modularity

`Mean.io` uses a more self-contained node packages modularity with client and server files inside the modules.
`Mean.js` uses modules just in the front-end (for angular), and connects them with Express. Although they're working on vertical modules as well...

Documentation

`Mean.io` has ok docs / `Mean.js` has AWESOME docs

Community

`Mean.io` is clearly winner and growing faster
`Mean.js` has less momentum cos it is a few months old