Joint Venture 2k15

This document has the essentials to know while working on the project

A techCompeer solution

Revision Table

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Author** |
| 16 Aug 2015 | Initial | Naveen |
| 22 Aug 2015 | Added Static Data – Walkthrough | Naveen |
| 25 Aug 2015 | Updated Static Data – Walkthrough | Naveen |
| 02-Sep 2015 | Addedd Sass folder to modules | Naveen |
| 03-Sep 2015 | Added MongoDB naming conventions | Naveen |

Table of Contents

[Revision Table 1](#_Toc429043086)

[Categorizing Application Units 3](#_Toc429043087)

[Back End 3](#_Toc429043088)

[Front End 3](#_Toc429043089)

[Routes in the Application 3](#_Toc429043090)

[Project Folder Structure 4](#_Toc429043091)

[Other Backend Files 6](#_Toc429043092)

[Front End Files – under the ‘public’ folder 6](#_Toc429043093)

[Front End – Modules 7](#_Toc429043094)

[Going a level under 7](#_Toc429043095)

[Key files 8](#_Toc429043096)

[js 8](#_Toc429043097)

[less 8](#_Toc429043098)

[sass 8](#_Toc429043099)

[AngularJS Fundamentals 8](#_Toc429043100)

[Directives 8](#_Toc429043101)

[Controllers 9](#_Toc429043102)

[Services 10](#_Toc429043103)

[Hands on Tasks 10](#_Toc429043104)

[Backend 10](#_Toc429043105)

[Front End 13](#_Toc429043106)

[Frequently Asked Questions: 17](#_Toc429043107)

[Database Static Scripts – Creation 18](#_Toc429043108)

[Naming conventions for MongoDB: 19](#_Toc429043109)

# Categorizing Application Units

1. Back End – nodeJS , Express JS, MongoDB,
2. Front End – AngularJS, Bootstrap, etc

# Back End

The application backend is nodeJS with expressJS as middleware and uses MongoDB as Database.

# Front End

The application front end – that is the web site when opened in the browser runs on HTML, CSS3, Bootstrap, jQuery and most importantly ‘AngularJS’.

# **Routes in the Application**

There are two types of routes in the application – backend routes, and FrontEnd routes

Back End Routes

* Handled by ExpressJS, may interact with MongoDB.
* Available in the file, **/app/routes.js**
* On the web, URL looks like this, [http://localhost:8080**/api/checkLogin?username=someone**](http://localhost:8080/api/checkLogin?username=someone)
* They typically fetch data

Front End Routes

* Handled by the UI-router of AngularJS.
* Available in the file, **public\modules\0.0.applicationbase\js\angular\app.js**
* On the web, URL looks like this, [http://localhost:8080/#**/home**](http://localhost:8080/#/home)
* Typically, load different view on the screen, show some popup/modal etc
* Usually, front end routes are after the ‘#’
* We do not have routes containing both Front End and Back End routes.

# Project Folder Structure



A closer look into the files in the root folder, we have a list of files described in the table.

The basic must have files –‘package.json’ and ‘bower.json’



|  |  |
| --- | --- |
| Files and their descriptions | |
| .bowerrc | has the path of the folder where bower install will download the dependencies. Currently it is ‘public/lib’ |
| .gitattributes | has the type of extension and applications associated with it |
| .gitignore | list the files/folders to be ignored by Git |
| 0.firstRun.bat, 1.startMongoDB.bat, 2.sartNodeServer.bat3.closeAll.bat | Batch files to be run in that order to launch the application in the web browser. |
| bower.json | The target file used by the command ‘bower install’. This file has the list of front end dependencies , for example, Bootstrap, AngularJS, Material Design. Etc. |
| Gruntfile.js | Grunt is a Javascript task runner. We define list of tasks that can be triggered from Grunt. For example, everytime we create a new module, we add new folders in the ‘public/modules’ folder, but we then concatenate all the ‘js’ files together and minify it to a location ‘public/assets/js/angular/angApp.js’  This task and many others can be run using Grunt. |
| package.json | The target file used by the command ‘npm install’. This file has the list of dependencies used by nodeJS application. For example, we need expressJS, BodyParser, Grunt, etc. |
| server.js | This file is the ‘curx’ of our application. It stitches various individual files together, sets up an Express JS Server, and responds to the requests received. |
| projectWalkthrough.docx, Readme.md | Are help files about the application. |
| sublime-project.sublime-project | This is a help file used to load sublime text editor with project folder navigation. It has no impact on the application. |

# Other Backend Files



|  |  |
| --- | --- |
| jv.js | This file has the schemas of different collections ( tables ) of joint venture database |
| routes.js | Contains all the possible routes and their actions as handled by the ExpressJS server. For example, when the user visits the location localhost:8080/api/createSampleUser  Will create a sample user in the ‘users’ collection in the db.  Localhost:8080/api/checkLogin – will use the username, password used in the POST and verifies the User in the ‘users’ collection. |
| db.js | A configuration file that says the database is available at which location and port |

# Front End Files – under the ‘public’ folder



|  |  |
| --- | --- |
| Assets | The css, img, and js files used. The ‘css’ folder contains the ‘css’ generated by the Grunt task (LESS compiler ) – We donot change a css file here, we change it in its corresponding LESS file.  Similarly, the ‘js’ folder contains the ‘angApp.js’ which is cumulated and at time minified version of all other ‘angular’ related files in the application. It is generated by the Grunt task. |

|  |  |
| --- | --- |
| Libs | This folder has all the css, js, etc specified in the ‘bower.json’. We donot modify anything here. |
| Modules | This folder is a well organized containing different folders. Organized means, the folders are named in a specific way, as they are appear in the application. |
| templates | This folder has all the ‘.html’ files that contain the ‘div’ content of different angular directives used in the application. |
| Views | This folder has the ‘div’ contents of various views. These are basically ‘.html’ files that are loaded dynamically whenever different links (Urls) are clicked on the application. |
| index.html | The root/base file that will loaded by the ExpressJS for all other ‘requests’ apart from back end. This file is the skeleton of the UI of whole application. |

# Front End – Modules

This folder is well organized. It contains list of modules ( Visual Separation ) present in the application.



As per the naming/numbering convention, 0 contains the absolute basics, and it will loaded before others in the page.

1, is the first visible component / module on the application. In our case, it is the navigation menu.

The navigation menu has, a ‘login’ link, so the login modal (popup) is a sub module of ‘navigation’ and hence the number 1.2

And if you look at the landing page, you can identify the components displayed in that order.

To make it convenient to map the code.

## Going a level under



Consider the section, ‘modalLogin’. It has the following folders

– js

– less.

|  |  |
| --- | --- |
| js/angular | This folder contains angularJS related code, be it controllers, directives or services. Though services folder is not present for this section. |
| Less | Contains the ‘.less’ files for this module (section). |

**Some Observations to be made**: The files under controller folder end with ‘Ctrl’, Directives with ‘Dir’ and Services with ‘Svc’.

The submodule.js initializes the namespace.

## Key files

### js

* public\modules\0.0.applicationbase\js\angular\**app.js**: This file is the crux of the angular application. It defines the ‘name’ of the angular app. Defines the namespace used, and contains the dependencies (usually, all dependencies are injected ) . Also contains the ‘UI-Router’ logic that is - the list of UI routes and their corresponding activities – also known as state management.
* public\modules\0.0.applicationbase\js\angular\**submodules.js**: This file contains the list of submodules that are defined and to be injected to the application.

### less

* public\modules\0.0.applicationbase\less**\applicationbase.less :** This file contains the variables to be used by other less files. That is, this file is imported to every other less file during compilation. You can place the branding specific styling here.

### sass

* public\mocules\0.0.applicationbase\sass\: this folder contains ‘sass’ variables to be used by other ‘.scss’ files.

## AngularJS Fundamentals

### Directives

They are the components of MVC, which modify/extend the usual HTML elements. There are core angular directives, and custom directives.

The core directives are prefixed with ‘ng-‘ example, ng-app=””, ng-bind=””.

The custom directives used in the application are prefixed with ‘app-jv-‘.

Every time you find such tags on any HTML element, know that the element is subject to angular processing and typically, it has a template associated with it, that puts all the HTML ‘content’ of the template into Element.

Consider the following ‘directive’ in the ‘index.html’

<!-- Login Modal -->

<div ng-controller="modalLoginController">

<div **app-jv-modal-login**="">

</div>

</div>

The div with the directive donot have any content inside it ( for obvious reasons), however, when this directive is processed, the login form is embedded into this ‘<div>’

Let us find where the ‘directive’ code is. There are two ways

1. Using Sublime Text : Find in files – Ctrl + Shift + F – Search for appJvModalLogin. It is AngularJS convention that, on the ‘HTML’ the directive has ‘-‘ whereas in the JS file, the ‘-‘ are removed and the word right next to ‘-‘ is capitalized making the whole word ‘Camel Case’.
2. Browse the \public\modules\ : look for name in the modules that is near match to the directive. The folders are numbered in a way that is convenient to look for modules the order, they are present on the screen, and named closer to what the directive name is.

In both ways, the file is : **\public\modules\1.2.modallogin\js\angular\directives\modalLoginDir.js**

 **Key observations**:

**templateUrl**: ‘location of the template file’ that will be embedded into the ‘<div>’ element.

**link**: a function that will be executed when the directive is processed. Generally, you can find all the DOM manipulation logic in the directives.

### Controllers

A controller is a piece of code that holds the ‘Scope’ variables that will be shared to its children – like inheritance, and as the name suggests, controls / responds to user interaction.

Typically, controllers hold the ‘scope’ variables that hold the data from the backend. Say, list of ventures are fetched from the BackEnd and stored in the controller Scope.

This scope variable, are then evaluated by the ‘directives’ under this controller.

### Services

As the name suggest, services are group of functions that perform the work of retrieving or setting data either from the backend or from external source. They are used a means of communication between different controllers.

In our application, we can find services are used to hit the ‘backend’ and retrieve data.

The controllers, inject the ‘Services’ and use the methods to store Data into the scope variables.

# Hands on Tasks

## Backend

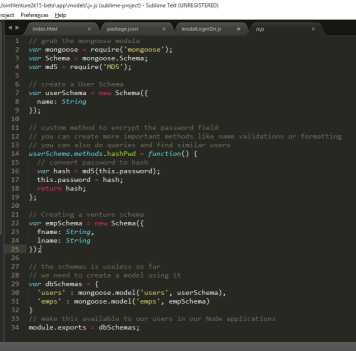
1. Create / Insert / Fetch Data :

So, we want to create a sample ‘Collection’ ( In relational SQL terms, it is Table), and insert a sample data into the table, and retrieve the data.

Things involved,

* + Setting up Schema of the Collection – app\model\jv.js
  + Creating a route in ‘routes.js’, that inserts a sample document ( a row )
  + Creating a route in ‘routes.js’ that retrives documents from the schema.

Refer the file, jv.js



Look at the lines between 22, 25. We have defined a sample ‘empSchema’ which has the fields, fname and lname, both String.

In the line 31, we added this empSchema to a mongoose.model, and added it to dbSchemas

Now, go to ‘\app\routes.js’



Observe the line between 10, 25.

We have created a route, /api/createEmp. That does the following, it creates a new emps schema record with fname – amit, and lname – dwivedi.

The empSchema is saved, and sends a response (‘Emp Created’).

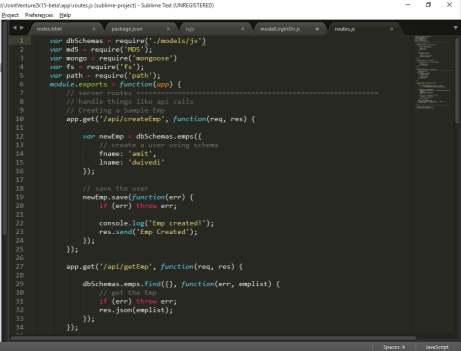
Save the files and restart the node server, and go to <http://localhost:8080/api/createEmp>



Observe that, Emp Created is printed on the browser.

Now, lets create a route that fetches the ‘document’ from the DB.

Go back to routes.js, and do the following



Observe the lines between 27, 34.

We have created a new route ‘/api/getEmp’ and it prints the Emps to the browser screen.

Save the file and restart the node server.

Go to the browser and go to – <http://localhost:8080/api/getEmp>



As you can see, the browser displays the list of documents in the ‘emps’ collection including our sample entry ☺

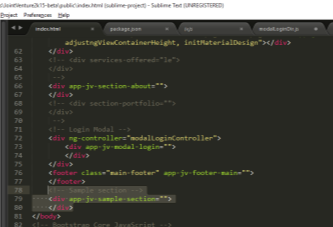
How Cool.

## Front End

1. Insert a sample section on the page, say, ‘sample-section’ and place it below the ‘Footer’, make its background ‘Blue’.

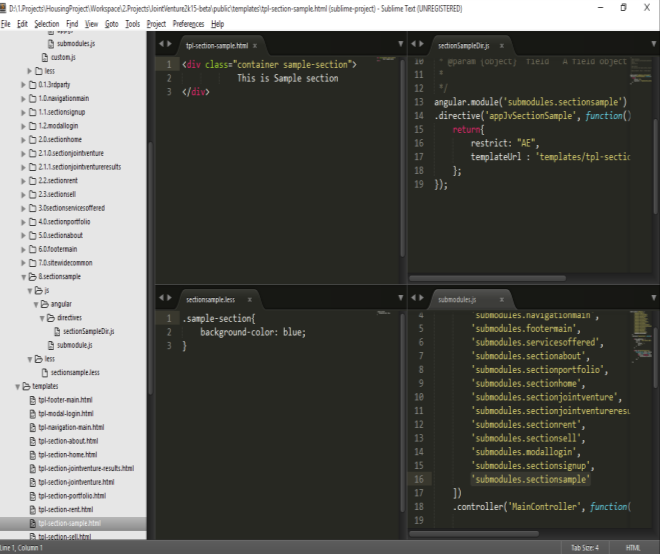
This task will involve the following,

* Modifying the index.html, to include this section. Ideally create a new directive.
* Creating new files under ‘modules’, both js and less
* Creating a new .html file in the \public\templates\ folder.
* Creating a new directive link it to template, a new submodule namespace.
* Inject the submodule, to main submodule - \public\modules\0.0.applicationbase\js\angular\submodules.js
* Adding styles in less file
* Run ‘Grunt’ using ‘build’ in sublime or through command prompt.



Check the line numbers 78 – 80 in index.html. We have added a new directive ‘app-jv-sample-section’

The following screenshot covers the rest of the changes



Created a new file in the ‘templates’ folder with name, ‘tpl-section-sample.html’.

Created a new folder structure under modules

8.sectionsample

js

angular

directives

sectionSampleDir.js

submodule.js

less

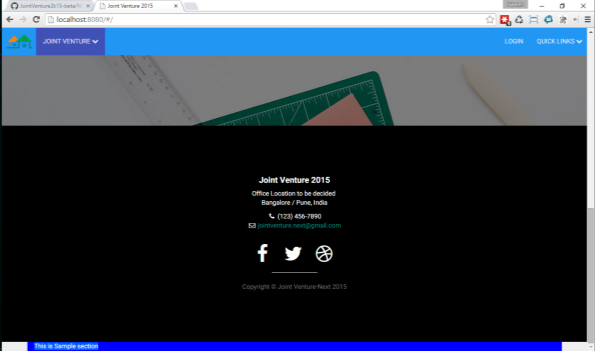
sectionsample.less

The submodules.js is modified. Added the ‘submodules.sectionsample’ to the dependency list (Refer bottom right window )

Run the Grunt file – either in sublime build, or command prompt

and refresh the browser – [http://localhost:8080/#](http://localhost:8080/)

And scroll to the bottom.



Cool isn’t it ?

Congratulations, you are now 70% equipped with coding knowledge required to work on the project! Still have any doubts, questions – feel free to read the FAQ section below or add to it, if not already present.

# Frequently Asked Questions:

Q1: Where is the repository located ?

A: The code repository is at: <https://github.com/naveen-upbeat/JointVenture2k15-beta>

Q2: I need the Workspace folder setup ?

A: Email me (naveen.upbeat@gmail.com) to request access to the workspace folder on google Drive.

Now the workspace folder is available at Box - <https://app.box.com/jointventureWorkspace>

Q3: How do I setup my workspace?

A: From A2 above, download the folder ( probably a .zip) file and extract it to a drive, which has atleast 10 GB of space. Navigate to the ‘3.Softwares’ folder and click on the ‘setupWorkspace.bat’. It will popup a window that displays both the necessary software being installed, and need to be installed.

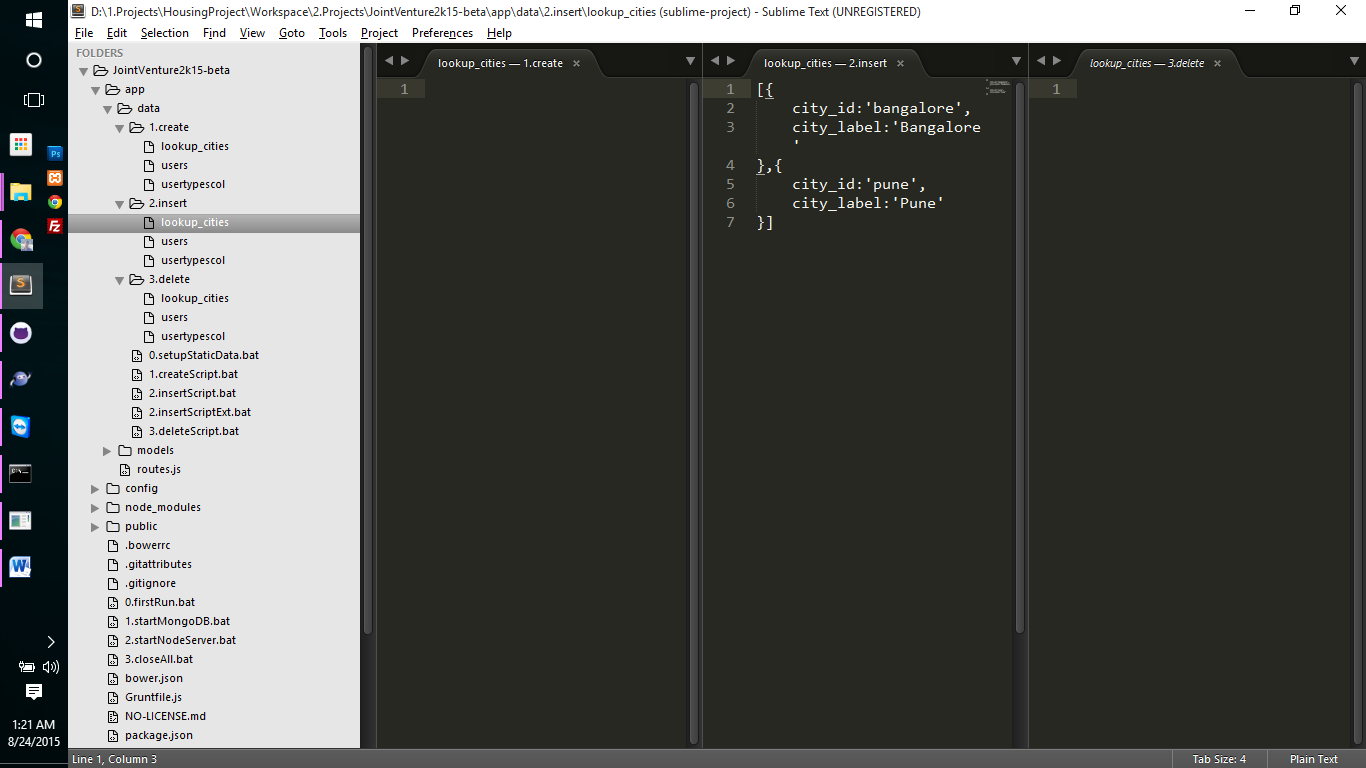
Click on ‘Install All’ that will sequentially install all the softwares required, and you might occasionally see permission popups to continue installation, some popup command prompt windows, and some forms getting filled automatically. Donot try to interfere with the installation, keep all your distractions away for hassle less installation.

Q4: Setup done, need references to learn and understand Javascript, other technologies used ?

A: In the A2, go to 4.Learning folder, it has list of URLs for the technologies used, and pdf’s for getting the basics.

# Database Static Scripts – Creation

Well, the traditional scripts have various create, insert or delete statements. But, now, you don’t have to know all the similar jargon for MongoDB. I assure you there are similar ways, for the sake of static scripts, there is an intuitive way.



Under the \app folder, there is now a \data\ folder that has 3 other folders

1.create, 2.insert and 3.delete

Each folder has files whose name is that of the intended ‘collection’ name.

While the files in the 1.create and 3.delete folders are just empty, the file in 2.insert has the ‘data’ that is expected to be populated into the collection.

Note: If there are multiple records in a collection (obviously) use a new line as line ending, other wise for one document – use only single line JSON.

How it works?

Just create a new file with name of the file that of the intended ‘Collection’, in all the three folders.

In the 2.insert folder, place the content of the intended Collection.

Now, start the MongoDB server (if not started already) and run the “0.setupStaticData.bat” -> That’s it!

# Naming conventions for MongoDB:

We have all worked with RDBMS and might have followed some conventions, however, let us follow some changes that were not followed previously.

1. Collection Name: All collection (Table) names to be CAPITALIZED and singular. Like, table CITY, USER etc.
2. Column names: Column names are small caps and separated by ‘\_’ [underscore] for multiple worded columns. Like, first\_name, last\_name, Use abbreviations if they are easily understood.
3. Id columns: Name the ID column as ‘id’. Saves time and space not prefixing the ‘id’ column.

Debate over Collection name to be singular –

It is unambiguous to have collection name as singular, say, if the collection contains ‘dress’ don’t have to add ‘es’ - ‘DRESSES’ (adding ‘es’), and for collection containing names of cities, the collection can still be called ‘CITY’ ( no ‘ies’ this time ).