Jason Rice
5/03/2016

# Agile Web Development with MEAN Stack

## Abstract

To learn more about the frameworks and technologies available through the employment of MEAN: Mongo NoSQL database, Express JavaScript (JS), AngularJS, and NodeJS. Apply MEAN towards the agile development of rapidly prototyped web application modules that can be applied towards a central web application. Enforce proper design pattern usage throughout the development. Display knowledge of MEAN and advantages through the development of a Mongo NoSQL database containing JSON documents with backend services running off of a NodeJS server instance and utilizing ExpressJS to enforce proper object (collection) mappings and other services such as security and auto-dependency injection (automatically loaded libraries). This robust backend will support a frontend built around the AngularJS Model-View-Controller (MVC) framework alongside standard HTML5/CSS3 and other CSS derivatives such as LESS and Twitter Bootstrap. A majority of the controllers and services in the MVC will be tested against a PhantomJS text-based browser running off of a GulpJS and KarmaJS based framework with Jasmine as the test engine. Code coverage and other testing paradigms, such as Test Driven Development (TDD), shall be enforced with tools such as IstanbulJS. Each week in the 14-week semester code branches will be committed to a Github repository and the final web application will be a result of the combined branches into the master branch.

## Deliverable

A full MEAN based web application deployed through Github with the code made available for the public to view on Github. The web app will showcase the various aspects of MEAN through a set of detailed tutorials that will include but will not be limited to the following: screenshots, printable code, database diagrams, data flow diagrams, design pattern diagrams, and UI features. The web app will contain an admin portal for allowing administrators to upload and modify existing course content. The web app will be protected from malware, spyware, and hackers through the employment of a secure encrypted key. For demonstration purposes the HTTPS keys will be self-signed. Users must register for the tutorials and once registered must enter in their username and password to login. Only admins with a secure username and password may log in to the admin portal. Admins may also view usage metrics describing who is logging into the web app, when, and from what locations. The metrics may also point out what tutorials are the most popular and vice versa. The landing page for the MEAN web app will take the user to a table of contents. The table of contents shall be based on the tutorials.

Outline

1. Introduction to JavaScript and NodeJS
    a. Setup a git repository and link it to a unique directory on file system.
    b. Install Node Package Manager (NPM) on my host end system then install NodeJS required packages under the unique git directory. Future required packages and code would be installed in the git directory and checked in. This way at any given time a user may fork the project on github.com and run the application with NodeJS.
    c. JavaScript syntax refresher with NodeJS
        i. Go through basic JavaScript concepts and run each JavaScript file against NodeJS (node) on the command line to discover the inner workings of JavaScript. Try to setup some basic AJAX calls to outbound servers or even setup a local host server (e.g. Tomcat) and connect to that. Test the AJAX calls within NodeJS by running the application. Over a mechanism for running the application with auto-restart when any code changes are detected (i.e. nodemon).

Weekly Deliverable:

Design and develop a driver application to verify installment of MEAN dependencies. Develop a series of scripts and configuration files that a user can use to setup all MEAN dependencies on Linux (CentOS/RHEL), Mac OS X, and Windows 7. Develop and maintain a package.json that npm will use to load required dependencies. For dependencies that our outside of the scope of npm (Node) provide download and setup instruction. For example this includes packages such as Python, the GNU g++ compiler, and at least one IDE. The target IDE will be Netbeans. The script will have a test feature to run a series of basic tests against NodeJS to ensure it is setup properly. The tests will run NodeJS against it's master configuration file (app.js) to make outbound calls to ensure AJAX is working properly. If the setup is correct the user should be able to navigate to the port displayed by NodeJS on the localhost IPv4 (127.0.0.1) and on the command line NodeJS will provide feedback that it is fetching requested pages.

2. NodeJS and Project Structure
    a. Setup Project Structure
        i. Include MVC based structure for server side.
        ii. Include MVC based structure for client side.
        iii. Setup project structure for CSS and vendor JS packages.
    b. Vendor support for NodeJS
        i. Establish configurations and test run additional NPM packages including Gulp, Karma, ElectrolyteJS, AngularJS, Angular Bootstrap, ExpressJS, etc.
        ii. Add any additional configuration files and add an ElectrolyteJS rule for connecting to the Mongo database.
    c. Test vendor setup
        i. Run NodeJS against the app.js file and ensure that it uses ElectrolyteJS to connect to the Mongo database and establishes an HTTP connection listening on port 80.

Weekly Deliverables:

Extend bash script to setup MVC server and client side directories along with vendor (third party) CSS and JS API's as mentioned in section 3.b.i Add additional configurations for ExpressJS, ElectrolyteJS, and Mongoose to the master app.js and setup the configuration directory with a developer.js file. Setup tests in the app.js to ensure ExpressJS is able to synch up with the Mongo database. This is prerequisite for Mongoose to work properly. Create a simple AngularJS HTML file and load it from the app.js file. This will ensure that the dependencies for AngularJS will be injected properly. The Angular based web page shall display a message saying describing the purpose of the web page and the fact that the web app is in the early stages of development. A few demonstrations of Angular may be included just to test that the AngularJS libraries for AngularJS and AngularJS UI (bootstrap) were added properly.

3. ExpressJS and Mongoose
    a. Setup ExpressJS database configuration
        i. Add a configuration file for Mongoose to connect to the default database and export the proper Mongoose and ExpressJS dependencies.
    b. Setup ExpressJS routing configuration
        i. Explore what it means to setup an ExpressJS Singletons.

ii. Learn how to create configuration rules with ExpressJS that route to various configurations.

c. Develop Mongoose models that can tie into the backend Mongo database and represent the entities (collections). This will be used later throughout the UI code.

Weekly Deliverable:

Extend the original NodeJS bash script and app.js by adding the additional MVC directories. Add the ElectrolyteJS and Mongoose configurations to the app.js file. Create a test to prove that Mongoose is able to connect to the local Mongo database instance. This includes setting up the Mongoose API directories (/api/<collection name>). Create a development.js configuration file for basic Mongo (Mongoose) configuration settings during development mode. Generate the routes directory with ElectrolyteJS route configuration files. Submit a write up describing how ElectrolyteJS employs the Inversion of Control.

4. Mongo NoSQL
   a. Database Design and Conceptualization
      i. Draw a Collection Design Document (CDD)
      ii. Compare and contrast traditional RDBMS (SQL) vs NoSQL schemas and design tools.
      iii. Design a set of database collections (entities).
   b. Install Mongo, Mongoose, and Seed Database
      i. Install Mongo and Mongoose
      ii. Seed Database collections with JavaScript functions that load the database collections with test data

Weekly Deliverable:

Submit a collection design document to a documents directory. Anytime a collection is added or modified this document must be updated. Submit a MongoDB seed file that will be a JavaScript (js) file that when ran against NodeJS will seed the database with mock (test) data and then run a few tests to ensure the data is mocked correctly. The test data will be a set of customers that belong to a bank and shall include the following: balance, user id, first name, last name, occupation, and state of residence. In the future this database will be loaded with tutorial notes so a tutorial collection will need to be generated along with tutorial images but those will be added as needed and this is just the setup stage so no tutorial data will be stored yet. Other collections may be added at this time or in the future. This is the brainstorming phase for the backend database.

5. Hands on with HTML5 and Twitter Bootstrap LESS (CSS) with SPA
   a. AngularJS works well with HTML5 and Twitter Bootstrap.

i. Install Twitter Bootstrap with NPM.
ii. Read up on AngularJS at the official web site (angularjs.org) to discover how AngularJS is applying a concept known as SPA (Single Page Application). Think about how to incorporate SPA into the web design.
iii. Design the web application mockups and storyboard with a free wireframe tool (e.g. Ninja Mock). Try to imagine how the various components can be integrated into a SPA. This mockup doesn't have to be perfect and can be adjusted throughout the lifecycle of the development. Not all components will conform to SPA and that is fine. For example a login page can be added that will redirect the web app from a page (html) running against HTTP to a page running against HTTPS.
iv. Develop a set of basic views that will act as a lander for the web page and other pages to demonstrate a basic life cycle for the web application (login, browse table of contents, logout). In order for the user to log in to the web app they must first register an online account that is free by setting up a username based on an email address and password. Apply Twitter Bootstrap to each page along with HTML5. This doesn't have to be perfect and though AngularJS may be included the main emphasis is just to get the basic UI implemented. The goal here is to develop a set of templates (components) that will later be applied by AngularJS. These templates are just fillers and are static html pages.

Weekly Deliverables:

Submit a style guide and Software Design Document (SDD) to the documents directory. This will contain the wireframes. The mockups will not describe every page in detail but rather provide a guideline of what the final web components should look like and what design they should adhere to. Implement the UI of the main (lander) page, registration page, and login page with the table of contents. Other pages may be added as needed. Note that in the future some of the HTML5 may be replaced in favor of Angular elements. These web pages do not need to be linked together in a session but the lander should provide links to all of them to make grading trivial.

6. AngularJS Introduction
   a. AngularJS MVC First Take
      i. Take a first look at controllers and services and learn about Dependency Injection (DI) and the usage of $scope and other AngularJS API utilities.
      ii. Research directives and how they can be incorporated with templates, controllers, and services.

        iii. Apply Dependency Injection (DI) throughout the front-end html (web page) through the use of ng-bind.
        iv. Learn more about ng-include and templating.
        v. Learn about other ng API utilities available through AngularJS.
    b. AngularJS Setup
        i. Apply the html templates from the previous week to create a very basic web application to demonstrate the MVC design pattern in AngularJS. Use services, controllers, and directives. Setup the web app login page, registration page, Table of Contents page, and other pages (e.g. Contact Us).
        ii. Tie this new Single Paged Application (SPA) back to NodeJS for default display when NodeJS starts up.
        iii. Setup the key pairs for secure login and added User collection to the database. This will store the username and password for each user along with info for admins such as the users last login date/time and from which IPv4 or IPv6 the user logged in from.

Weekly Deliverables:

Generate a directive and controller or service for each of the templates. Apply ng-bind to the template (html) pages to include dynamic content (e.g. preferences, username, etc.). Include a mock (default) username of 'user' for test purposes. The password will be 'password'. When a user logs in switch from HTTP to HTTPS. Develop and admins page that will allow and admin to query for who logged in, when, and from where the user logged in from.

7. AngularJS Scope and Testing
    a. Continue to learn more about AngularJS by utilizing the $scope operator and other features.
    b. Apply Test Driven Development (TDD) by setting up KarmaJS through NodeJS on the backend with GulpJS. GulpJS will manage the test goal and other goals such as JavaScript minification for production deployments.
    c. Download PhantomJS and setup in Karma config file. Finish setting up Karma configuration file.
    d. Download IstanbulJS and setup in GulpJS config file.
    e. Ensure a code coverage of previous AngularJS code of at least 80%.
    f. Future AngularJS additions must adhere to the 80% code coverage rule.

Weekly Deliverables:

Setup a Karma and Gulp config file. Add goals to Gulp such as Minification, logging, and tests (integration and unit) that will all run automatically when NodeJS starts up. Create Karma Jasmine spy based unit tests. Each test describes a feature of the web UI and IstanbulJS ensures that code coverage enforced meaning each code module in the MVC is being accounted for. Comment out some tests. If the code coverage is below 80% a warning is issued from the command line and the tests are marked as having failed.

8. AngularJS and MEAN Advanced Topics
    a. Continue to develop web application and apply AngularJS routing.
    b. Pull data files from remote servers through AJAX based AngularJS calls with the $http API.
    c. Perform an AngularJS $http GET to a remote server.
    d. Perform an AngularJS $http POST to a remote server.
    e. Learn about AngularJS form validation.
    f. Learn more about logging options available through NodeJS
    g. Format results of HTTP transactions back to Mongoose for storing in the Mongo database.
    h. Perform error checking and handling with Mongoose to ensure no invalid values are inserted into the database.

Weekly Deliverables:

Apply MorganJS logging. All logs should be recorded (teed off) to a **log** directory. The log types should be based on their meaning such as the following: INFO, WARNING, ERROR, SEVERE. Apply form validation so all forms account for faulty inputs. Implement tests to prove form validations. Develop the first of the tutorials and link these tutorials to the Table of Contents. The tutorial data shall be contained with the Mongo database. Add the data to the Mongo database and employ Mongoose to retrieve the datasets. Develop Karma (Jasmine) tests to prove the data resides in Mongo.

9-14 Agile Web Application Development with MEAN Final Project
- a. Continue to apply MEAN to develop your application.
- b. Allow six weeks to continue research and development of MEAN and to apply it towards your web application.
- c. Integrate all prior discussed advanced topics.
- d. By the end of development you should be able to make new discoveries and gain new insights towards the development of MEAN based web applications. For example you should be able to use Mongoose to add additional object models.
- e. Try to branch out from MEAN and use npm to install additional JavaScript libraries such as MorganJS for logging. Use any available JS tool to enhance the web development and web application. The goal here is to discover through trial and error what JavaScript libraries available through npm will enhance the MEAN based web app.

6 Week Deliverable:

At the end of each week append the README.md or README.txt file to include what tutorials where added and where to locate them under the Table of Contents. Each tutorial must showcase (demonstrate) through AngularJS and

Twitter CSS (LESS) the various aspects of MEAN. Showcase at least 4 additional JS third party (vendor) API's and incorporate these into online tutorials. Throughout each week the code coverage must be maintained at 90%. Update the Admin portal to allow admins to update tutorial content. This will allow course the MEAN instructors to update content on the fly. Updating content will either enforce a new write or update on the Mongo database. Add tests to notify the tester when any new content is added to the Mongo database.