Code Inspection Report

TextForSale

Client
Arash Fallah

CoffeeBots
Mehreen Awan
John Gordon
Mina Beshai
Daniel Schomisch
Daniel Kelly

11/29/2016

TextForSale

Table of Contents

1. Introduction

1.1 Purpose of This Document

The purpose of this document is to provide a capsulation of the coding practices followed during the development of the TextForSale application. Details discussed include coding and commenting conventions, an overview of defects encountered, our personal impression of the project, and schedule of team meetings.

1.2 References

1. TextForSale System Requirements Document
2. TextForSale System Design Document
3. CodeSchool (2016). The MEAN Stack. Retrieved from "https://www.codeschool.com/mean"
4. hack.hands (2014). How to get started on the MEAN stack. Retrieved from "https://hackhands.com/how-to-get-started-on-the-mean-stack/"
5. thinkster (2016) Learn to Build Modern Web Apps with MEAN. Retrieved from "https://thinkster.io/mean-stack-tutorial#introduction"

1.3 Coding and Commenting Conventions

We implemented our code using the CamelCase naming convention. This means the names of our variables are formed by combining multiple words and capitalizing the first letter of each word, making the code easier to read. The conventions here are the conventions were taught to us by the computer science department at UMBC.

Comment blocks can be found at top of each file describing the contents of the file. Additionally, we wrote in-line comments for any special methods or commands and block comments to describe each function.

Each variable name must be relevant and descriptive of its role. Each class name must be the name of the object it constructs. While simple loop variables are left to individual letters to keep the simplicity of the loop and provide a code that is easy to read. These rules work best such that team members can read and understand the code.

1.4 Defect Checklist

Table 1. Defect Categories

Category	Description
Coding Conventions	Errors in this category are when coding syntax differs from the ‘ideal’. This makes reading the code more difficult, thus complicating the inspection and debugging process.
Logic Errors	Logic Errors result in the code giving a different outcome than what was originally intended.
Security Oversights	Security oversights are weaknesses in code that cause it to be predisposed to external attacks. They make it easier for malicious individuals to hack into your application.
Commenting	Too many, too little, or no comments.

Table 2. Defect Checklist

Category	Defect
Commenting	No function header comments
Security Oversights	Didn’t hash passwords multiple times before storing in database
Security Oversights	Users should only be able to see bare minimum of site before registering - we show them too much.
Logic Errors	If you clear cart, it will remove cart but not the session
Logic Errors	Cart doesn’t immediately update when items are added to it
Logic Errors	Redirected to wrong pages

Coding Convention	Repetitive code
Security Oversights	No backup of user information
Commenting	Missing comments for some variables/methods
Commenting	Missing some file header comments
Commenting	Too many unnecessary comments in certain areas
Logic Errors	Title and Author name being mixed up when searching
Coding Convention	Inconsistent indentation
Coding Convention	Class naming didn't follow CamelCase
Coding Convention	Some variable names didn't follow CamelCase

2. Code Inspection Process

2.1 Description

Code inspection was conducted by all members of the team through github and an occasional meeting in person. However, we plan on reviewing all code and documentation a couple of more times in person before the final product is delivered to the customer.

Our inspection process wasn't entirely ideal because getting the entire team together was a difficult process due to all of our very demanding schedules. As a result, the majority of our inspection process occurred individually through github or via email, and all members ended up inspecting the entire code. Going forward, when we meet in person we will assign each member blocks of code to inspect since it is known that reviewing too much code at once makes it more difficult to debug.

2.2 Impressions of the Process

Our code inspection process was effective. Since each member decided to try to examine the entire code, we all became equally familiar with all parts of it. Although effective, it may not have been the most efficient. Had we taken the “ideal” path, debugging might have been done at a faster pace, but we probably wouldn’t have all thoroughly inspected all parts of all files. Additionally, lack of communication decreased efficiency. Throughout the coding process it was not always clear when someone wanted to make an update to the master branch of our git repository. The lack of communication created wasted effort if one had their new code working on an older version, he/she would have to revert and then implement new code again.

The ‘best’ modular units in our program are anything related to user authentication because any errors are easily visible (users.server.controller.js, users.server.model.js, users.server.routes.js, email-validator.client.directive.js). It was also the area that was most focused on because it was necessary for all other part of the website.
On the other hand, the ‘worst’ unit was definitely updating the shopping cart (cart.server.controller.js, cart.server.routes.js, etc.). simply because it was the last thing created. We have to make sure we don’t damage any other processes while trying to perfect this one and also have to make sure we don’t forget to test it as thoroughly as all of the other units.

2.3 Inspection Meetings

Date	**Time**	**Attendance**
10/14/16	12:00pm-4:00pm	All members
10/28/26	12:00pm-4:00pm	All members
11/11/16	1:00pm-6:00pm	All members
11/23/16	10:00am - 10:45am & 2:00-4:00pm	All members

3. Modules Inspected

#	File/Module	Brief Description
1	controllers/books.server.controller.js	Contains methods for the book controller. Describes how to handle a response.
2	controllers/index.server.controller.js	Creates render controller method
3	controllers/users.server.controller.js	Contains methods for the user controller. Describes how to handle a response.
4	models/books.server.model.js	Contains methods and schema for the 'books' model
5	models/users.server.model.js	Methods and schema for the user model
6	routes/books.server.routes.js	Each route assigns a book controller, or node module, function to an HTTP verb
7	routes/index.server.routes.js	Loads the index controller and defines the routes module
8	routes/users.server.routes.js	Each route assigns a user controller, or node module, function to an HTTP verb
9	views/includes/footer.ejs	Loads the footer of the home, user, and books module
10	views/includes/header.ejs	Loads header, takes care of mobile device view as well.
11	views/includes/navbar.html	Navigation bar
12	config/env/development.js	Sets the development environment settings
13	config/strategies/local.js	Local strategy for npm module "passport-local". Provides standard support for username and password authentication
14	config/config.js	Gets environment and loads appropriate configuration file
15	config/express.js	Contains configuration method for the node module "express". Accepts instance of MongoDB and starts an instance of "express".
16	config/mongoose.js	Connects mongoose to the MongoDB database and implements Bluebird as a Promise library.
17	config/passport.js	Configuration of node module "passport". Serializes and Deserializes user information and

		implements strategy for authentication.
18	assets/main.css	Accommodates the navigation bar. Overrides bootstrap
19	books/config/books.client.routes.js	Contains the route provider definitions for the 'books' portion of client-side Angular app.
20	books/controllers/books.client.controller.js	Front-end controller for books. Handles registering to editing account information.
21	books/services/books.client.service.js	The user service. Main function returns a singleton to be used for making calls to the server's API.
22	books/views/books.create.client.view.html	Creates the view for the client for adding a book.
23	books/views/books.list.client.view.html	Creates the view for a lists of books.
24	books/views/books.read.client.view.html	Creates the view for adding a book to the cart, trading a book, and checking out a book.
25	books/books.client.module.js	Creates the 'books' module.
26	home/config/home.client.routes.js	Directs user to home client view.
27	home/controllers/home.client.controller.js	Implements user authentication.
28	home/views/home.client.view.html	View for the user in the 'home' module.
29	home/home.client.module.js	Creates the 'home' module.
30	users/users.client.module.js	Creates the 'users' module
31	users/config/users.client.routes.js	Contains route provide definitions for the 'users' portion of client-side.
32	users/controllers/users.client.controller.js	Front-end controller for the users.
33	users/directives/email-validator.client.directive.js	Validates the email when a user registers.
34	users/services/authentication.client.service.js	Creates the 'Authentication' service.
35	users/services/users.client.service.js	User service. Main function returns singleton to

		be used for making calls to server's API.
36	users/views/account.client.view.html	Creates view for a user's account.
37	users/views/list.client.view.html	Creates view of a list of users.
38	users/views/register.client.view.html	Creates view for registering a user.
39	public/application.js	Creates the application.
40	Gruntfile.js	Task manager
41	bower.json	Runs bower. Bower must be installed to run the MEAN stack.
42	index.js	Main server file for the TextForSale app. Target for this file is "npm start" command. Loads app dependencies and listens locally on port 31337
43	package.json	Handles all dependencies required for TextForSale.

4. Defects

Defect	**Category**	**Module**	**Fixed?**
Users can access page to add books before they're registered - they shouldn't be able to.	User friendliness	books.client.routes.js	Yes
Passwords were not hashed multiple times before storing	Security	passport.js	Yes.
When the cart is cleared the session is not reset so the cart appears empty until the page changes or is	Logic error	cart.client.controller.js	Yes.

refreshed.			
Searching confuses title and author. Searches for words based on both categories	Logic errors	models/books.server.model.js	Yes
No back ups of books and user information.	Security	passport.js	No

Appendix A – Team Review Sign-off

This document has been collaboratively written by all members the team. Additionally, all team members have reviewed this document and agree on both the content and the format. Any disagreements or concerns are addressed in team comments below.

Team

_____

Print Name

_____ Date _____

Signature

Comments _____

_____

__

_____

Print Name

_____ Date _____

Signature

Comments _____

_____

__

_____

Print Name

_____ Date _____

Signature

Comments _____
_____
__

_____
Print Name
_____ Date _____
Signature

Comments _____
_____
__

_____
Print Name
_____ Date _____
Signature

Comments _____
_____
__

Appendix B – Document Contributions

Mehreen Awan wrote the Purpose, references, coding & commenting conventions, and defect checklist. John Gordon wrote the description, impressions of the process, and modules inspected. Mina Beshai created the tables for the inspection meetings and defects. Daniel Kelly inspected the code and conducted all unit tests used to write this report. Daniel Schomisch drafted appendices A and B.