Market Based Craft Shop Application

Nathan Gibbons

Problem Report submitted
to the College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science in Software Engineering

Katerina Goseva-Popstojanova, Ph.D., Chair
Bojan Cukic, Ph.D.
Dale G. Dzielski, MBA, CMA, PMP

Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2013

Keywords: Market, Java, Single Page Application, Online Retail

ABSTRACT

Market Based Craft Shop Application

Nathan Gibbons

Over the past decade, the opportunity for the average person to make money via online retail has grown tremendously. From the early days of the Internet through the "dot com bubble" years, online retail was the domain of large established companies like Amazon etc.

Online retail has created an environment for small businesses and even the casual hobbyist to become successful entrepreneurs. However, within this small community of sellers, an ever-widening separation has evolved between the individual seller and the more successful of small businesses. The current system of assessing a flat fee for all types of sellers has created an environment that is very difficult for the individual seller to become successful.

This application provides a way to decrease that separation by using a flexible fee schedule that somewhat evens the playing field. It presents a path for individual sellers to become successful and for the successful sellers to thrive while promoting healthy competition. Essentially, fees are assessed according to the current market and will adjust according to the success of the seller and the category of item being sold. This type sliding fee assessment avoids penalizing the individual seller with prohibitively steep fees, but will gradually increase as the seller becomes more successful.

Lastly the application includes additional features that benefit the end user, such as cart timers, waiting lists, and a centralized search engine.

TABLE OF CONTENTS

List of Figures iv

List of Tables iv

Notation iv

CHAPTER 1: INTRODUCTION 1

1.1 KEY ISSUES 2

CHAPTER 2: APPLICATION 4

2.1 MARKET BASED FEE SCHEDULE 5

2.1.1 ITEM CATEGORIES 6

2.1.2 CATEGORY MARKET SATURATION 7

2.1.3 SELLER'S PROPORTION OF SALES 9

2.2 APPLICATION FEATURES 10

2.2.1 CART TIMERS 11

2.2.2 WAITING LISTS 12

2.2.3 UNIVERSAL HAND CRAFTED SEARCH ENGINE 12

2.3 APPLICATION ARCHITECTURE & TECHNOLOGY 13

2.3.1 Database Server 14

2.3.2 APPLICATION PROGRAMMING LANGUAGE 14

2.3.3 TOMCAT APPLICATION CONTAINER 15

2.3.4 JPA/Hibernate 15

2.3.5 SPRING Framework 17

2.3.6 APACHE SOLR 18

2.3.7 REST WEB SERVICES 19

2.3.8 CLIENT INTERFACE 20

CHAPTER 3: CONCLUSION & FUTURE WORK 22

References 24

LIST OF FIGURES

FIGURE 2-1 : FEE REPORTING VIEW 10

FIGURE 2-2 : CART TIMERS 11

FIGURE 2-3 : APPLICATION ARCHITECTURE 13

FIGURE 2-4 : SOLR SEARCH RESULTS 18

LIST OF TABLES

TABLE 2-1 : INCREMENTAL FEE STRUCTURE 8

NOTATION

The following acronyms are used throughout the problem report:

API	:	Application Programming Interface
REST	:	REpresentation State Transfer
ACID	:	Atomicity; Consistency; Isolation; Durability
ORM	:	Object Relational Mapping
MVC	:	Model; View; Controller
SQL	:	Structured Query Language
JSON	:	JavaScript Object Notation
AJAX	:	Asynchronous JavaScript and XML
POJO	:	Plain Old Java Object
HTML	:	HyperText Markup Language
CSS	:	Cascading Style Sheets
URL	:	Uniform Resource Locator
DOM	:	Document Object Model

CHAPTER 1: INTRODUCTION

In the past, sites like Etsy, eBay, and Craigslist have enabled small businesses and hobbyists a chance to peddle their wares and make small to large profits. While these sites offer opportunities, which were previously only available to large retailers, there are still some discrimination even within the small business and hobbyist demographic. This is due in large part to the flat rate fee schedule enforced when listing and selling items on these sites.

The application developed and described herein proposes a solution to this inequity and offers solutions to several other user facing problems that plague current storefront type systems. The introduction of a market-based fee schedule is a distinctly original approach to this problem and is central to the overall solution. This application is designed to cater to producers of handmade crafts, but the market-based pattern described could feasibly be applied to a storefront application of any type of product. The application presented includes both the server and client architecture, which are designed to complement each other, and which are implemented using cutting edge technology and frameworks. The type of agility which is fostered in both the application design and implementation enables continued refinement and extensibility, which are keys to success according to Ashby's law of requisite variety [1].

1.1 KEY ISSUES

The inequity described above inherently stems from the use of a flat rate fee schedule. The site must enforce a fee on the sellers in order to make a profit. In charging this fee, the site must find a happy medium where enough money can be made to be profitable, but not so high that it becomes too expensive for the sellers. Etsy for example charges a fee of 3% of selling price of an item (in addition to a $.20 listing fee per item) [2]. Charging the fees in this way allows the seller to determine the selling price and would appear to be a fair and logical approach. In the world of small crafts and handmade items however, this is not the case, and it actually penalizes the very small producer and hobbyists. Larger producers can buy supplies in bulk and produce more items in a much shorter time than the hobbyist, therefore the 3% fee is not as painful for them because they can afford to lower their prices (due to smaller supplier costs) and the higher volume of sales will offset the cost of the fees. Because the small hobbyist can neither purchase supplies in bulk nor produce in the high volumes as their larger competition, they must charge higher prices for their items and sell fewer of them. This effectually creates an ever-widening divide that prevents the small producers and hobbyists from ever becoming successful. Instead in this problem report, we propose an original, completely data-driven, market-based fee schedule derived from the current selling rate determined by all sellers in the market. This enables smaller producers to be charged smaller fees if they produce fewer items. The sliding fee schedule provides them an avenue to increase sales and the rate for fees will increase as they become more successful.

Another central problem that this application provides a solution for is the checkout process for listing and checkout of certain items. It is not uncommon for certain types of crafts to be made in batches. Dyed Roving [3] and yarn used by knitters & spinners is a prime example of this type of item. Since these items come in batches instead of a steady stream or in bulk, buyers must anxiously wait until the item is posted (usually notified by announcement of some type). Then they must go through the process of adding the item to their shopping cart and working through the checkout process. The problem here is that for most of the current storefront sites, the item is still listed for sale until the checkout process is complete. Therefore, in instances where items are listed in a batch, there is a mad dash through the cart and checkout process. However, the user does not know if the item is still available until the end of the checkout process where they often find the item's stock has been depleted by someone finishing the checkout process before they did. This application uses a system of shopping cart timers and waiting lists, similar to the timers & waiting lists used on sites like Amazon [4] and Ticket Master, to solve that problem and ease process of purchasing in for the batch type of items.

The third and final problem this application aims to solve is the simple problem of discovery. Many sellers tend to shy away from using a 3rd party application to sell their products for many of the reasons listed above, and they often find it easier to sell from their own website. This fragmented system of online retailers represents the current state of ecommerce on the Internet. General search engines like Google, Yahoo, and Bing do well to help users find such individual web sites, but their scope is too broad and users typically get back search results that are

vaguely related, or not retailers at all. If there was a way to filter these general search engines for only retailers of hand crafted items, then the problem would be solved, but this is not the case. This application will implement centralized search engine restricted to only retailers of small handcrafted items, including retailers both within the application and without.

CHAPTER 2: APPLICATION

The application is an entirely web based application that provides an online marketplace for producers of hand crafted items to promote and sell their items for profit. It caters to sellers of handcrafted items of all sizes, but is specifically developed to enable the success of small business and hobbyist crafters. It also aims to become a central, unified search engine for those looking specifically to purchase handcrafted items (whether they are sold on the application or not). This capability is accomplished by providing an external API for listing hand crafted items that can be sold on a crafter's own website. These features are discussed in greater detail in the following sections. It is important to note the distinction that the application is not the "seller" of items. The application exists only as a service provider; providing a platform or marketplace for individual sellers to set up their virtual shops and sell directly to customers. The application charges a fee on each item that is successfully sold within this platform and that fee is determined by several factors discussed below.

2.1 MARKET BASED FEE SCHEDULE

The term "fee schedule" is a reference to the current fee (implemented as a percentage of the selling price) that a seller is charged when they complete the sale of an item by listing it within the application. This fee is determined by several factors, including the type of item sold, the seller's item quantity and sales rate, and the saturation of sales for that type of item. The fee schedule must have minimum and maximum boundaries in order to prevent the fees from becoming too great or too small if it is entirely based on sales rates. These boundaries will most likely need to be evaluated once more data is available, but for the initial purposes of this software, they will be set at 0.5% and 3% respectively.

The determination of the fee schedule is designed to be more lenient toward small businesses and individual hobbyists, yet it will not penalize larger sellers for their success. In order to accomplish this, the fee calculation is based on three specific data points that are pulled from the application's current data stores. These are 1) The type of item sold (or category); 2) the saturation of the market for that particular item category; 3) The seller's proportion of item sales within that category. Stated in very simple terms, if a seller sells more items they will be charged a higher fee. However, based on the nature of handcrafted items, it is unrealistic to expect completely different items to all sell at the same rate. This is why item categories are necessary, so that the application can compare selling rates of the same or very similar types of items. However, this comparison can still be skewed if only a small number of sellers are selling that type of item. This is why the saturation of the item category's market must be taken into consideration.

2.1.1 ITEM CATEGORIES

The categorization of items is necessary in order to establish an accurate market model for the item. For example, simple earrings are not going to take as long to make and will have a higher rate of sale than a complex hand knitted sweater would. It would not be fair or accurate to increase the fee charged to the earring seller based on how many sweaters are sold. By categorizing the types of items the application can more make a more accurate comparison of the sales rates between different sellers and therefore reflect that in the fee assessment.

The item category model introduces a couple of issues that need to be addressed. The first concern is the assumption that every item will fit nicely into a pre-defined category. This is an absurdly optimistic assumption, and very shaky ground upon which to base the fee schedule assessment. Even if it were the case, it's highly plausible that future items will be created that do not fit any of the categories established for all the current items. The second major issue is the definition of the categories themselves and the rules that determine into which category an item is placed. This point could be particularly contentious because the nature of handcrafted items introduces some subjectivity into what category it might fit into, and the fees charged to a seller can significantly change based on this decision. Since the item type can be highly subjective, a seller may disagree that an item is accurately classified. The simplest solution is to let the seller decide into which category the item belongs. However this is not a very stable solution by itself. Sellers could create semantically duplicate categories and the category list would quickly become unwieldy. It might also encourage sellers to create new categories

to try to circumvent the established fee schedule for an existing item category. These problems can be resolved by combining user-defined categories with a sliding fee schedule based on the market saturation.

2.1.2 CATEGORY MARKET SATURATION

It's likely that there will be certain categories that have relatively few sellers. The fee schedule calculation must take this fact into account. For illustration purposes, suppose that a certain category contains only two sellers - seller A and seller B. If seller A sells 100 items and seller B sells 101 items, then without accounting for the market saturation, the application might assess seller B the maximum fee (3%) and seller A the minimum fee (0.5%). This is an extremely inaccurate assessment of fees, and more importantly it is incredibly unfair toward seller B who sold only one more item than seller A.

The other reason that category market saturation must be consulted is because the application will allow the sellers to categorize their own items (including creating new categories). This presents the problem alluded to above, where a seller might create a new category to avoid what might be higher fees if they placed the item in the correct category.

There must be several business rules in place and enforced to address these problems. First, if a new category is created - or if a category only has a single seller - there is effectively no market to consult, and therefore a flat rate must be assigned in this case. In order to discourage the dishonest behavior described above, as well as encouraging honest sellers to carefully consider a pre-defined category before

attempting to create a new (possibly duplicate) category, the flat rate for single sellers will be set to 2.5%. This rate was chosen to be high enough to discourage new categories, and not quite the maximum to try to accommodate legitimate new category creation. This rule will allow the community of sellers to somewhat keep themselves in check and it behooves new legitimate category creators to try to find competition to gain more market saturation in that category.

However, even as the category gains a larger market, it is still unfair to evenly distribute that market across the sellers, as was illustrated in the example with seller A and seller B. For this reason maximum and minimum caps will have a much smaller range initially and increase (up to the previously defined caps) in defined increments as the number of sellers grows. The business rules that govern the incremental fee schedule are described in Table 2-1:

Number of Sellers	***Minimum Fee***	***Maximum Fee***
1	2.5%	2.5%
2	1.5%	2%
10	1.25%	2.25%
50	0.75%	2.75%
>= 100	0.5%	3%

Table 2-1: Incremental Fee Structure

These rules ensure that legitimate categories with a lower saturation will not be penalized and the range can grow as the market becomes more saturated. This is the sliding fee schedule referred to above.

2.1.3 SELLER'S PROPORTION OF SALES

The third factor that an item's fee schedule is calculated from is the proportion of the maximum sales attributed to the seller. These totals are calculated daily. Based on the saturation brackets described above, the seller with the most sales that day will be assessed the maximum fee percentage (since they were attributed the maximum sales).

Taking all three factors under consideration, the evaluation of the fee schedule assessment that is applied to sales for each day is given by

FEE ASSESSMENT = $(s * (f_x - f_n)) + f_n$;

where

f_n = minimum fee;

f_x = maximum fee;

s = (items sold / maximum items sold in that category)

Other sellers will be charged a fee based upon their proportion of the total range. The following example will help to illustrate this calculation. Suppose at the end of the day there were 3 sellers in a particular item category. Seller X sold 2 items; Seller Y sold 3 items; and seller Z sold 5 items. Using the factors described -

Seller Z will be assessed a fee of 2% (as the maximum), seller Y will be assessed a fee of 1.8%(0.6 of the fee range (0.5)) and seller X will be assessed a fee of 1.7%(0.4 of the fee range (0.5)). Each seller may view his or her current fee assessment breakdown at any time within the application. Figure 2-1 illustrates the application's fee reporting view.

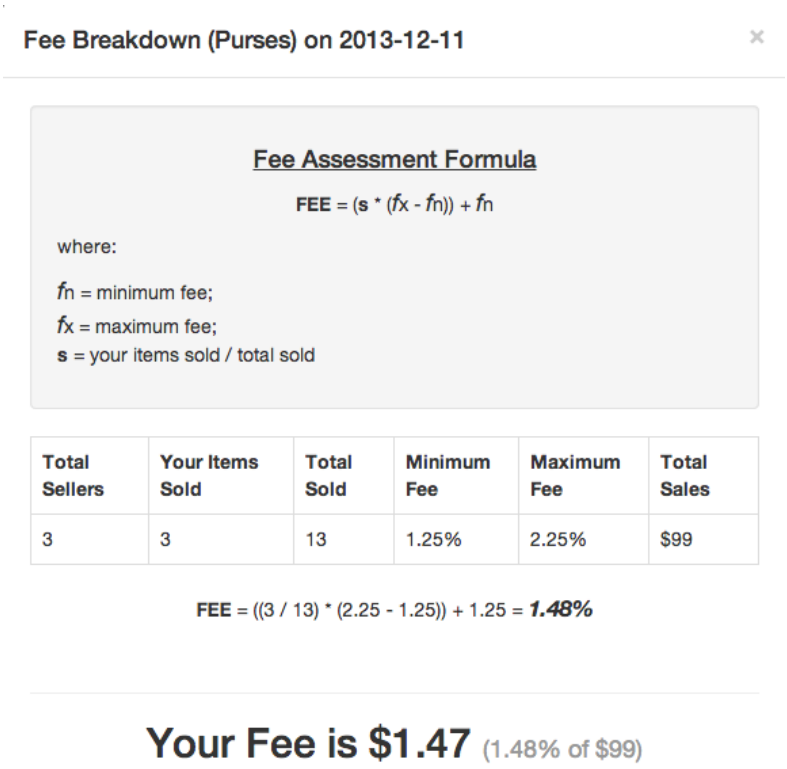

Figure 2-1: Fee Reporting

2.2 APPLICATION FEATURES

The checkout process is another problem area that this application addresses. The main problem, as described earlier, is that a buyer may not discover that an item is out of stock until he/she reaches the end of the checkout processes. This is

often very frustrating for users. In order to help alleviate this problem, the application employs the use of cart timers and waiting lists.

2.2.1 CART TIMERS

The basic concept of a shopping cart timer is that a user is given a specific length of time (in minutes) to complete the purchase and finalize the checkout process. Ticketmaster [5] is a prime example of an online retailer that employs this type of shopping cart timer. However, when user places an item (or items) in the cart, that quantity of items is immediately removed from the available stock and the timer starts for them to complete the checkout process. This way the user is guaranteed that the item is reserved for them as long as they complete the checkout process before the timer runs down. Figure 2-2 shows a shopping cart featuring two items that have cart timers associated.

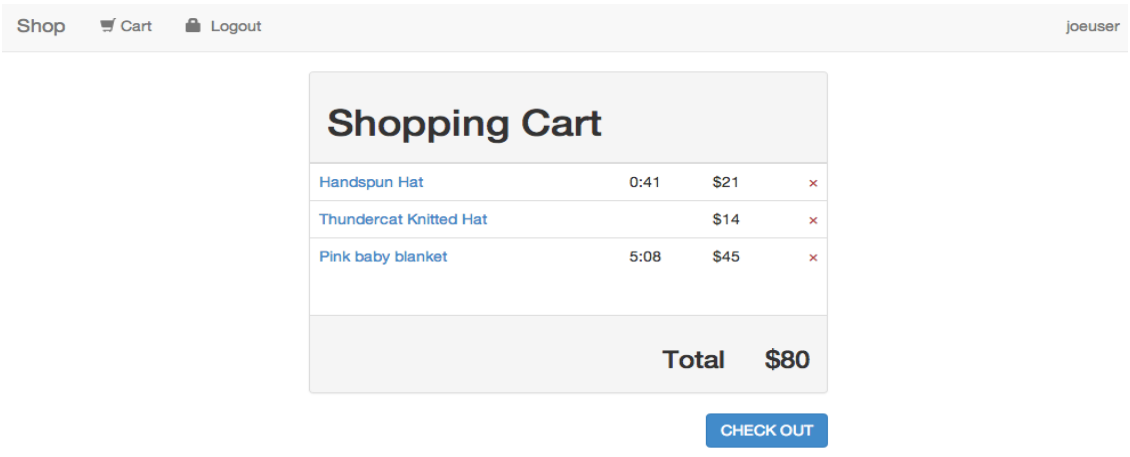

Figure 2-1: Cart Timers

2.2.2 WAITING LISTS

A waiting list is mechanism set in place for when there are no longer any available items in stock, but at least one user has not completed the checkout process. If the user(s) fail to complete the checkout process, then the item(s) in their carts are released and subsequently placed in the cart of the next user on the waiting list and their cart timer begins, etc. These are simple, easily understood constructs that create a much better experience for the end user.

2.2.3 UNIVERSAL HAND CRAFTED SEARCH ENGINE

All items listed within the application are automatically indexed and fully searchable. This has become a standard feature for most websites. However there are many crafters who prefer to sell on their own website and use their own checkout process. In order to find these sellers, customers would need to be directed to the website or find them by their own searching means. Standard search engines like Google and Yahoo are obviously helpful in this regard, but their scope of search results is entirely too large. The application's scope of searching is much narrower, being tailored specifically to hand crafted items only. The application aims to be the single portal for search and discovery of handcrafted items, whether they are listed & sold from within the application or not.

This is accomplished by exposing a REST based indexing API that would allow outside sellers to include their items in the search index and subsequent search results would direct the user to the appropriate site for purchase. The fees would be a simple pay-per-click scheme that is common for online advertising. The

motivation for providing this service is to direct all customers wishing to find hand-crafted items to the application as a one stop shop for finding any and all hand crafted items for sale and therefore encouraging more sellers to use the application itself.

2.3 APPLICATION ARCHITECTURE & TECHNOLOGY

The application is a Java based web application, built using several frameworks and technologies. Figure 2-3 shows a summary of the complete server application stack.

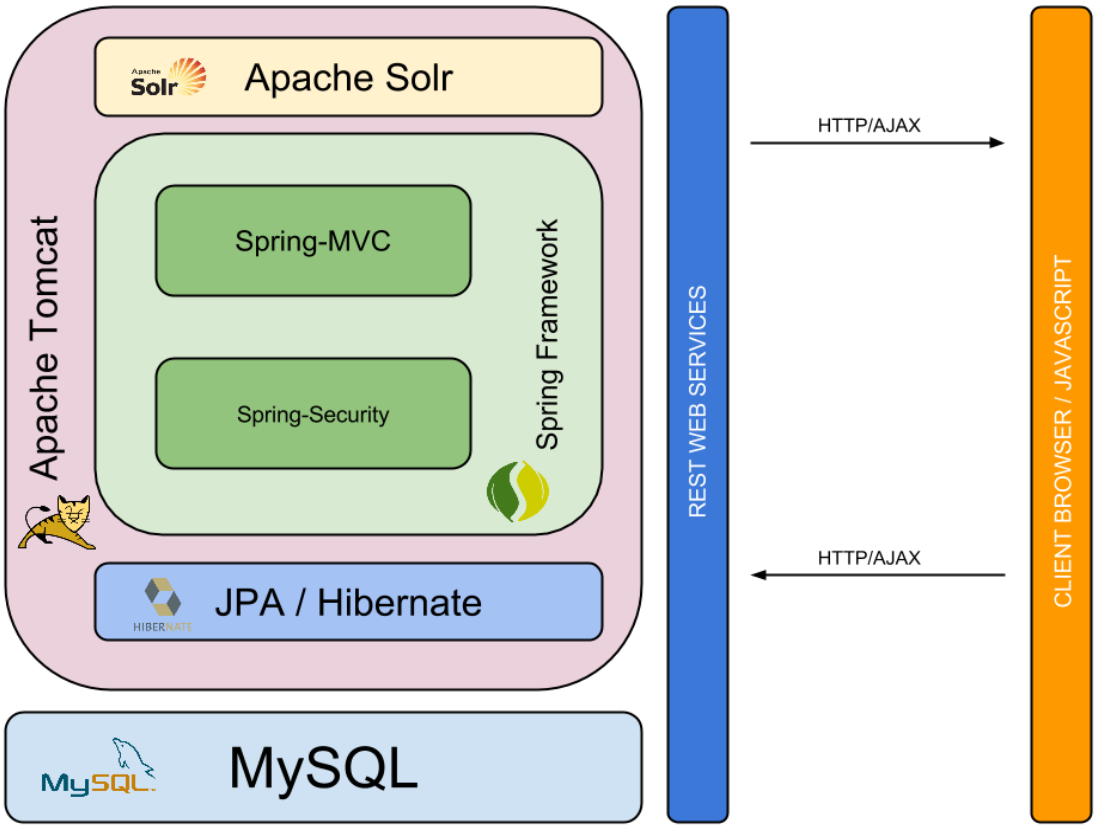

Figure 2-3: Application Architecture

2.3.1 Database Server

There are relatively few requirements for the database technology. In order for the cart timers and waiting lists to function correctly the database must be ACID [6] compliant. Otherwise, the selected database technology could be one of any number database types or vendors. A relational database lends itself nicely to this application, but is not strictly necessary. However there are currently very few non-relational databases that comply with ACID requirements. A relational database also simplifies the application development by enabling use of common ORM structures and technologies. MySQL™ is a well-used [7], ACID compliant [8] database that is free to use under the GNU General Public License.

2.3.2 APPLICATION PROGRAMMING LANGUAGE

Like the database server, the application's programming language could be implemented using any one of several choices. Several factors were considered when selecting the programming language, including ease/speed of the development process, framework availability, scalability and overall performance. Through a series of simple evaluations and also taking into account personal preference and experience, the choices were narrowed to either Ruby on Rails or Java and Spring MVC. Ruby on Rails far out marks Java in the realm of ease and speed of development, however Java exceeds Ruby in terms of scalability and performance [9]. In fact these were the central factors in Twitter's recent migration from Ruby on Rails to Java [10]. The final decision was made to use Java as the

programming language for the application, sacrificing some speed in development for the benefits of future scalability and major performance increases. Although, like the database selection, the language implementation could be any number of choices, altering the language selection later on is a much more difficult task than replacing the database technology. This is due to the modularization and loose coupling that ORM patterns provide. So ultimately, the language decision upfront is fairly important. These considerations presented Java as the better long-term choice for programming language.

2.3.3 TOMCAT APPLICATION CONTAINER

Again, there are several selections available to function as java servlet containers, including Apache's Tomcat™, Oracle's Glassfish™, and RedHat's JBoss™. Since the java servlet specification is well-defined standard, a java servlet application can be altered to use a different application container relatively simply. Tomcat was selected for this application due to its widespread use, abundance of documentation, and open source availability.

2.3.4 JPA/Hibernate

Java Persistence API and the Hibernate ORM Library work hand-in-hand to simplify and standardize communication with the data store (MySQL™) and in this application become the actualization of the "Model" portion of Spring Framework's™ MVC [11] pattern described below. In this application Hibernate™

becomes the implementation of the Java Persistence API, which creates a standard language and programming pattern for data persistence. This application heavily utilizes Java annotations for both JPA and Spring™ components.

Hibernate offers some very high value benefits as well as drawbacks that are somewhat unpredictable or unexpected. One such drawback has to do with the way that Hibernate de-serializes a database record into an object, specifically in regards to collections. Hibernate's implementation of database record relationships is to store them in java collections (usually lists), however Hibernate uses it's own implementation of a `java.util.List [12]` called a "PersistentBag" [13]. This implementation actually violates both Java's hashcode and equals contracts [14]. This particular fact is known by Hibernate [15], yet it remains largely undocumented, and therefore the cause of extreme frustration and many hours of confused debugging throughout the development of this application.

By using Hibernate, the developer also loses most control over the specific SQL [16] that is executed against the database server. This can be either a boon to the developer or a significant problem depending on the particular situation.

Due to these considerations, Hibernate was actually dismissed halfway through development in favor of Spring's own `JdbcTemplate [17]` pattern for data access. However, since the data model changed so frequently during development, requiring updating of the database schema, Java Object Model [18], and SQL statements for each change, this decision was quickly reverted back to using Hibernate. The fact that the persistence layer could be swapped so easily is a

testament to the benefits of Spring Framework's implementation of the Inversion of Control [19] pattern.

2.3.5 SPRING Framework

Spring is a Java based framework that offers several projects that focus in depth on specific functionalities with Inversion of Control [19] support as a central defining feature. Spring helps to somewhat simplify and standardize java application development with the Spring Web MVC [20] project being their web application centric offering. As the name suggests, Spring Web MVC implements web application development using the Model-View-Controller design pattern, which is a common and very useful design pattern for web development and is also implemented to a different degree in the client JavaScript code as will be discussed later. Spring-MVC also simplifies the exposure of REST based web services, which for this application are the central means of communication between the client and server.

Spring-Security [21] is another popular spring project that is utilized in this application. Spring-Security's major features include established patterns and specifications for user authentication and authorization, as well as attack protection and simple integration with Spring Web MVC. This application implements the Spring-Security API to authenticate and authorize users, storing the user credentials in the MySQL data store. Securing application resources is accomplished by specifying URL [22] patterns in a spring specific configuration file. This makes it incredibly simple to alter security settings and secure new URLs without changing

code or needing any recompilation. This pattern (configuration with dependency injection) is common throughout the Spring projects and is one of the largest selling points for the framework.

2.3.6 APACHE SOLR

The Apache Solr™ project is a Java based indexing and search project incorporating Apache Lucene's™ inverted index at its' core. Solr [23] exposes both a REST API as well as a Java client API (SolrJ) [24], used for both adding documents to the index as well as search and retrieval. This application utilizes Solr to implement the universal hand crafted search engine described earlier. As new items are listed within the application, or outside sellers submit an item for inclusion in search, the

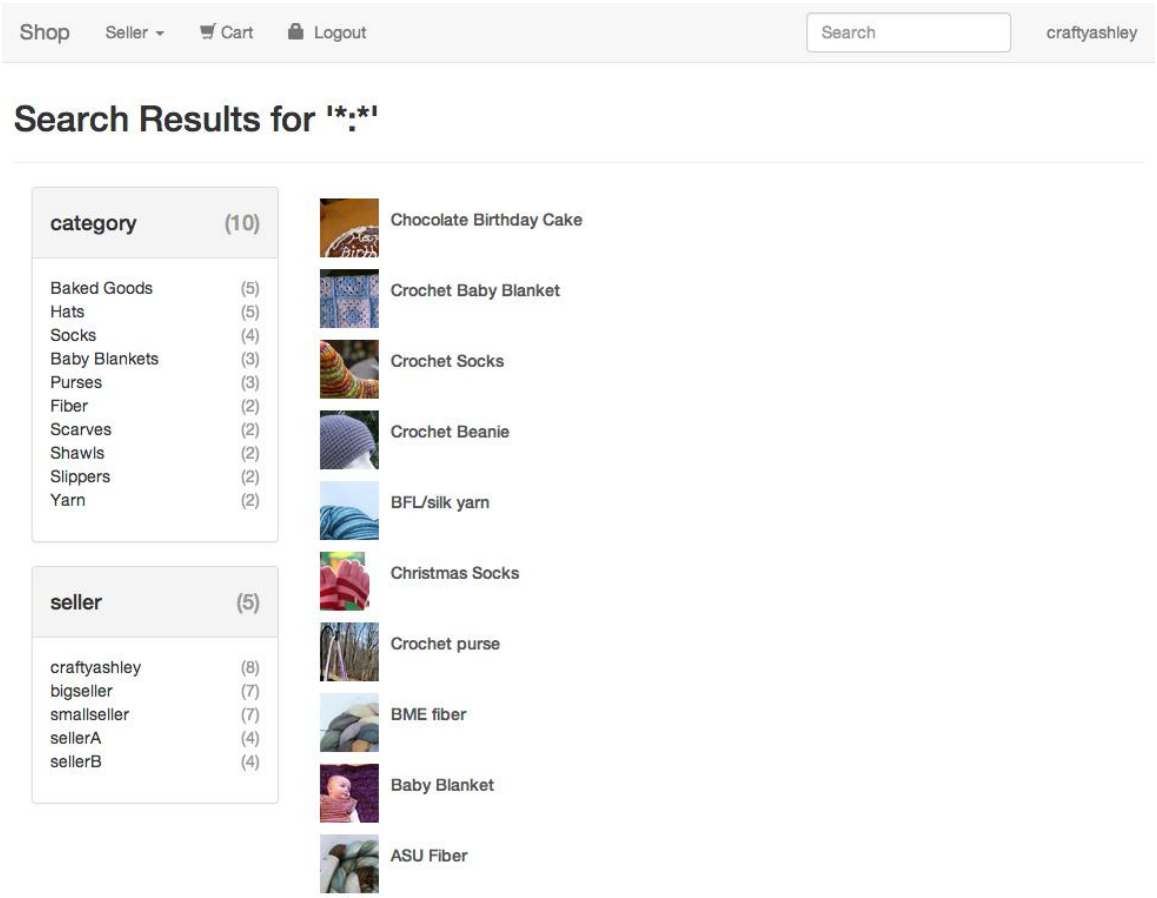

Figure 2-4: Solr Search results

application adds the item to the Solr index using the SolrJ client. When an item is sold, or instructed to be removed by outside sellers, the item is subsequently removed from the index by the same means. Figure 2-4 displays the application search results screen.

Apache Solr makes search & retrieval relatively simple and provides an essential function within the application. Solr's default implementation is served through an application servlet, so it fits nicely within the tomcat application already in use serving the main application.

2.3.7 REST WEB SERVICES

This application is somewhat unique in that it is exposed almost entirely via REST based web services serving data in JSON format. The browser employs a JavaScript framework called EmberJS™ [25] that enables 'single page' applications. This is discussed in greater detail in the next section, but the basic concept is that the entire application is served from a single html page and all data views are interchanged with the server via AJAX calls as opposed to regular http. Spring Web MVC [20] significantly simplifies the creation of REST based services using `@RequestMapping` [26] annotations on methods of `@Controller` [27] annotated objects. Marshaling of the java objects is accomplished using Spring's implementation of Jackson's MappingJackson2JsonView [28]. This marshaller will convert any standard POJO [29] into a JSON [30] representation and attach it to the response object that is sent back to the requester.

By representing almost all interaction with the application via REST web services, the application is primed to expose an external API to allow outside developers to write their own applications to interact with this craft application.

2.3.8 CLIENT INTERFACE

The client facing application is written entirely in JavaScript using HTML 5 and CSS 3 as the presentation layer. It implements a single page application architecture [31], which provides for a much more fluid and responsive user experience since there are no reloads or new page requests. The application uses the browser location hash property to create unique URLs within a single page that can therefore be bookmarked and still utilize the browser's history functionality. This is all accomplished using the EmberJS™ JavaScript framework.

2.3.8.1 EMBERJS™

EmberJS™ is a client-side MVC framework written and implemented in JavaScript. It uses the jquery library to communicate with the server side application via AJAX as well as to perform DOM manipulation and data binding. It uses a powerful templating syntax called handlebars which is an extension of the mustache library.

A common problem with a JavaScript heavy application is code organization. Since JavaScript is an interpreted, loosely typed language, management of a large amount of code can quickly become extremely burdensome to manage an update.

EmberJS imposes order on the JavaScript code, following the MVC pattern. The model is where the local data objects are modeled & stored, the view is the templates embedded within the HTML, and the controller is where most of the event handling takes place. EmberJS requires the application to namespace (by enclosing the entire project within a named object), which ensures there is no conflict with any other JavaScript libraries that might be employed. The model is typically bound to the view templates and any modifications to the data from user interaction is automatically stored in the model which makes it very easy to send the modified object model to and from the server. As stated above, communication with the server is done completely via serialized JSON in AJAX calls.

This AJAX-only type of communication presents a fundamental shift in the way typical website interaction is developed. The UI must be responsive and more informative especially when waiting (asynchronously) for communication with the server to complete. In traditional client/server interaction, the browser itself displays some type of indication of the request status. One area of particular difficulty was sending binary files (images used to showcase the crafted items for sale) to the server for persistence. In traditional web development, this is accomplished using a multi-part form POST request. Doing this would break the single page, AJAX/JSON only paradigm of the entire application. The solution involved the use of relatively new HTML 5 technology. For security purposes, JavaScript is not allowed access to a user's file system. However, in HTML 5 the concept of client side storage was introduced. Essentially, the browser uploaded the image file to its own local storage, thusly allowing JavaScript access to the image for

immediate display even before uploading to the server. JavaScript could therefore access the images, but that still did not provide a way to send the images to the server without using a multi-part request.

Overall this solution presents a much cleaner, more responsive application experience. Almost all model updating happens in place in the browser, no need to submit a form of any type. As soon as a user finishes updating an input field, the update is asynchronously submitted to the server and persisted without the user having to pause for the request to complete.

CHAPTER 3: CONCLUSION & FUTURE WORK

The application described will be a significant tool to enable sellers of handcrafted items to become successful. The market based fee schedule will provide individual sellers and very small businesses an opportunity to become successful and create an environment of healthy competition which will urge both large and small retailers to create better products. Ultimately, this is an environment where the individual seller, larger seller, and end customer all reap the benefits.

This is by no means a comprehensive or fail proof solution. In particular, the min/max boundaries and sales thresholds will most likely need to be adjusted once real data is available to be analyzed. However, the application is specifically designed to be flexible to incorporate these analyses. There is no better teacher

than the data itself. The application business rules are designed to be flexible and largely data driven and completely transparent to the sellers.

References

[1] F. Heylighen and C. Joslyn, "The Law of Requisite Variety," [Online]. Available: http://pespmc1.vub.ac.be/REQVAR.html.

[2] "Fees Policy," [Online]. Available: http://www.etsy.com/help/article/2144.

[3] Battenkill Fibers, "How to Dye Wool Roving," [Online]. Available: http://www.battenkillfibers.com/documents/DyeingRoving_Jan2011.pdf.

[4] Amazon, "Amazon Help: Join a Lightning Deal Waitlist," [Online]. Available: http://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=201134100. [Accessed 5 12 2013].

[5] "Official TicketMaster Site," [Online]. Available: http://www.ticketmaster.com/.

[6] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys,* vol. 15, no. 4, pp. 287-317, 1983.

[7] Oracle, "MySQL Market Share," [Online]. Available: http://www.mysql.com/why-mysql/marketshare.

[8] Oracle, "MySQL and the ACID Model," [Online]. Available: http://dev.mysql.com/doc/refman/5.6/en/mysql-acid.html.

[9] RedMonk, "The RedMonk Programming Language Rankings: June 2013," [Online]. Available: http://redmonk.com/sogrady/2013/07/25/language-rankings-6-13/.

[10] Twitter, "Twitter Search is Now 3x Faster," [Online]. Available: https://blog.twitter.com/2011/twitter-search-now-3x-faster.

[11] Wikipedia, "Model-view-controller," [Online]. Available: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[12] Oracle, "List (Java Platform SE 7)," [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/util/List.html.

[13] RedHat, "PersistentBag (Hibernate API Documentation)," [Online]. Available: https://docs.jboss.org/hibernate/core/3.2/api/org/hibernate/collection/PersistentBag.html.

[14] Oracle, "Class Object," [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#hashCode().

[15] Hibernate, "HHH-3799," [Online]. Available: https://hibernate.atlassian.net/browse/HHH-3799.

[16] Wikipedia, "SQL," [Online]. Available: http://en.wikipedia.org/wiki/SQL.

[17] S. Framework, "JdbcTemplate," [Online]. Available: http://docs.spring.io/spring/docs/2.0.x/api/org/springframework/jdbc/core/JdbcTemplate.html.

[18] Wikipedia, "Object Model," [Online]. Available: http://en.wikipedia.org/wiki/Object_model.

[19] Wikipedia, "Inversion of Control," [Online]. Available: http://en.wikipedia.org/wiki/Inversion_of_control.

[20] Spring, "Web MVC framework," [Online]. Available: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html.

[21] Spring, "Spring Security," [Online]. Available: http://projects.spring.io/spring-security/.

[22] Merriam-Webster, "URL," [Online]. Available: http://www.merriam-webster.com/dictionary/URL.

[23] Apache, "Apache Lucene - Apache Solr," [Online]. Available: http://lucene.apache.org/solr/.

[24] Apache, "Solrj," [Online]. Available: http://wiki.apache.org/solr/Solrj.

[25] [Online]. Available: http://emberjs.com/.

[26] Spring, [Online]. Available: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping.

[27] Spring, [Online]. Available: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-controller.

[28] Spring, "MappingJackson2JsonView," [Online]. Available: http://docs.spring.io/spring/docs/3.1.4.RELEASE/javadoc-api/org/springframework/web/servlet/view/json/MappingJackson2JsonView.html.

[29] Wikipedia, "Plain Old Java Object," [Online]. Available: http://en.wikipedia.org/wiki/Plain_Old_Java_Object.

[30] "JSON," [Online]. Available: http://www.json.org/.

[31] "Single-page application," [Online]. Available: http://en.wikipedia.org/wiki/Single-page_application.