CS 1530 - SPRINT 2 DELIVERABLE

Coffee-specific Import and Inventory Tracking System

Karnbir Singh – Karnsingh214

Adam Rison – AdamRison

George Totolos – get13

Chris Meier – chrismeiercs

Due Date: June 24, 2015

**Accomplishments**

During this sprint, team communication improved. We decided that stand-ups were best held after each class. We were able to discuss the state of the project, any extra information we needed, and each of our current tasks. We held meetings on Slack to help each other resolve any issues, and had an in-person meeting to map out the project. This meeting in particular gave us a better idea of how to progress with the project, and allowed us to better plan our work.

There were some disagreements on how the system should be designed during our in-person meeting. In the end, we decided on a Pipeline approach, but initially, some of us thought that it might not be the best architecture pattern we could use. During our discussion, each idea was heard and drawn out on the white board. By doing this, we were able to point out any flaws or defects that we might come across with each pattern. Ultimately, everyone was able to see that the Pipeline approach was best and came to an agreement.

Since the last sprint, we decided that communication between ourselves, and with the customer, needed improvement going forward. We held more meetings and more discussions about the project. Moreover, we were able to get any questions we had answered faster, which allowed each of us to get a better understanding of the customer’s needs. Additionally, we roused more interaction with the customer through email and phone call. With more experience came more comfort in contacting the customer with any issues that we had. This led to a better understanding of their needs and how to move forward with designing the system.

Another major focus of this sprint was coordinating and syncing all of the software we are using. All of us are new to SpringMVC, so learning the various quirks and nuances was difficult. Figuring out how each piece of software fit together was particularly frustrating. While some members have worked with similar frameworks, such as Django, Spring felt very bloated. We were able to figure it out by spending a lot of time with tutorials and documentation. Additionally, once we switched over to IntelliJ IDEA from STS, many of the minute, aggravating details were taken care of.

Like much of the software, testing is very new to us as well, which is where we struggled the most. We had trouble determining what tests were appropriate to write and how to find edge cases. The approach we took was writing a test for each method that was not a setter or a getter. Once we figured out how tests are written in the Spring framework, it became easier to figure out what should be tested and in what manner.

As previously mentioned, our project is a web application and our framework enforces this paradigm. As a result, we heavily focus on using the MVC pattern for development. This helps us separate our code into divisions, improving readability and usability. We also discussed how access to our back-end should be implemented. For now, we have settled that the connection and updates should follow the singleton pattern. This will help reduce cluttering by preventing multiple instances of this object scattered around the application. It will also prevent any instances that require the same user be initialized repeatedly throughout the application, as we are using a BaaS (Backend-as-a-Service) called Parse.

During this sprint, we were able to complete some of the “New Shipment” module of the application. Also, the shipping calculator was implemented. During the next sprint, we will begin developing the remainder of the module. The inventory system will also be a major part of the next sprint. Although we currently do not have our back-end access in the master branch, we have created the code necessary to access it and are holding said code in a separate branch. To be safe, we want to be certain that the code is completely without fault, as a minor mistake could lead to storage charges and/or data corruption. During the next sprint, fixing and maintaining the database will hold our highest priority.

**Completed User Stories**

<https://github.com/chrismeiercs/CS1530CoffeeImporter>

1) As a user, I want to keep track of the total shipping cost of an order by using an import calculator so that I know how much coffee to buy to get the best pricing (shipping cost decreases per bag as number of bags goes up).

2) As a vendor/user, I want the web app to calculate the specific shipping cost for importing a product into the country, separate from the rest of the shipping cost, so that I know exactly how much money I am spending out of pocket for the product.

3) As a vendor/user, I want a web app to create a calculated number representing the distributed shipping cost across the weight of the product that I buy so that I can apply it to the product that I sell in order to break even or turn a profit.

**User Story Decisions**

After talking with the customer more, we were able to get a better idea of what she wanted. We determined that parts of the system are to fit together into a pipeline structure. The customer preferred to enter in all of the data by hand and have calculations performed and stored in a database. We decided that is how the development of the application should be approached. We manually went through the process the customer gave us and determined the best way to perform tasks in a step-wise fashion. Our main focus this sprint was the shipment calculator. User stories were selected in a way that fit the pipeline-like process described to us by the customer. The three with the highest priority satisfied this.

Additionally, each of worked on other user stories during this sprint while focusing on the highest priority ones. Some of the other user stories are completed or near completion, but have not been adequately tested or need to be reviewed by the group before they can be merged with the master branch on Github. In other cases, there were issues in software setup that prevented members for working on user stories. Despite the many problems, each of us was able to accomplish some type of goal during this sprint.

Through working with user stories over the course of this sprint, we have discovered that our initial set of fourteen user stories is too vague. Many of the user stories that we have as of right now need to be broken down into smaller user stories and technical items. For instance, work spent on developing back-end integration could fall under the precinct of at least three different user stories, and yet its completion would fail to fully satisfy any of them. Bearing in mind issues like this, the initial part of the next sprint will be spent reprioritizing our user stories and breaking them down into more manageable and concise tasks.

**Testing**

Though test-driven development had a large time cost at the beginning, it did help with being able to structure the code properly to handle different situations. It mainly helped with determining edge cases. For example, in the WeightCalculator class, tests were written when developing the actual calculator portion. Since user needs to input the total weight of the shipment, and the default in the form is zero, a divide by zero error can occur if the user forgets to fill in the input.

System testing also helped with the process. Through system testing, we were able to catch more defects than with unit testing. Going through the same process that the customer will helped point out some issues. When manually testing the cost calculator for the shipments, we found that it is possible to enter that the cost of the products was larger than the total cost of the shipment. After finding this, the method that calculates the shipping cost throws an exception letting the user know that this is not allowed. We then determined that the user should see this message and be able to respond appropriately. When combined, system and unit testing greatly helped.

Before the end to this sprint, some team members looked over the code to find defects. We found that having others review your tests and code helped with finding more defects. Since this was the end of the sprint, they were added to the product backlog. The one issue was form validation. We discovered that the user could enter strings instead of dollar amounts in the calculation forms. When this happens, the program is supposed to catch the misinformation through form validation. However, this was not yet implemented. Instead of alerting the user, Spring returns a 404 error, along with a validation error for the form backing object. We decided to prioritize this issue for the next sprint due to the fact that it was not caught until near the end of this sprint.

Another complicated issue that appeared was in back-end access testing. As previously mentioned, we are using a third-party library to access our Parse database. We are forced to use this third-party library that uses Parse’s REST API because Parse does not support a Java API. Through testing unit testing and running the code, we discovered that for some unclear reason, Parse objects can only be created if the code is ran normally and not as part of a test. When one runs the code, regardless of state or location, as long as internet connection is available, a Parse object should be generated in the database with the assigned values it was given before saving in the background. A similar problem is occurs when attempting to delete an object from the Parse database. This problem, like the previous one with the 404 error, was found towards the end of the sprint. Also, considering the database is sensitive to over-querying, will charge in the event of copious data usage, and can lead to data corruption if not properly used, we decided to leave it out of the master branch to be safe.