Development of an Inventory Management System: Agile Software Development

A Senior Honors Thesis

Submitted in Partial Fulfillment of the Requirements for Graduation in the Honors College

By:

Matthew Fritschi

Computer Science Major

The College at Brockport

May 17, 2019

Thesis Director: Dr. Sandeep Mitra, Computing Sciences

Table of Contents

Abstract

For this thesis, I created an inventory management system for The College at Brockport's Kinesiology, Sports Studies and Physical Education department along with a team of three other students. Prior to this, students and faculty would reserve and checkout equipment using pen and paper. In today's highly technology-based world, though, this seemed like an unreliable and out of date way to track the school's inventory. With our system in place, the department is able to keep perfect records of what equipment they have, what has been checked out, by whom, and whether something is in stock at any given time or not. They can receive reports on all previous checkouts that have occurred and keep track of any late fees that may be acquired by a student for returning equipment past a given date. Keeping paper records of all this information is difficult and tough to manage, and so there was a need for a system such as what we intend to develop.

For this project, we took an Agile approach. Agile is a commonly used coding methodology that involved meeting with our thesis director, Sandeep Mitra, once a week to discuss the progress we have made and what should be done within the following week. By doing this we ensure that we are always aware of how much progress has been made by each team member as well as what needs to be accomplished and by when. In Agile software development, there are also opportunities to iterate over each stage of the Software Development Life Cycle multiple times. During any coding endeavor, the requirements that are provided by the customer may change several times as they want new features to be added in, or a current feature to be changed. Because we weren't restricted to going through the SDLC only once, we were able to

incorporate these changing requirements into the finished product. Agile has been proven to be one of the best coding methodologies in place for delivering a product that meets customer needs and delivers the product on time.

In this paper, I will be discussing the different coding methodologies that our team considered using, and the pros and cons of each. I will explain our design process to give further insight on how to properly take a coding project from its conception through to its completion. I will also discuss the different frameworks and coding patterns that we used in the development of the application. Finally, I will give some recommendations for future development on our system and show images of the user interfaces we developed to demonstrate the workflow of the system.

Introduction

The world that we live in today is quickly becoming less reliant on paper and more reliant on using technology to store various records. After all, technology provides us with a level of precision and ease that simply cannot be matched by using paper. Thanks to current day computer applications, human error is far less of a concern when it comes to having accurate and reliable information at our disposal. Because of this, it was very surprising to me that SUNY Brockport's KSSPE (Kinesiology, Sports Studies and Physical Education) department is still using paper forms/binders to record their inventory as well as equipment checkouts. Given the size of the department, I had assumed they would have already bought some sort of record-keeping software to track all of this. Nevertheless, they hadn't. With the assistance of three fellow Computer Science Majors (Liam Allport, Nicholas Barnard, and Lucas Wing), we were able to

develop an inventory management system for the department that will hopefully make the jobs of whoever oversees record keeping much easier.

The features that the KSSPE department had requested for us to add were the ability to add, modify, or delete any workers authorized to use the system, borrowers, equipment categories, or equipment items from a database service. They also requested the ability to have reports that would let them know all previous checkouts, and to provide receipts to any students who would like one. The program should be able to keep track of who has what equipment, when it was taken, when it is due back, and if there are any late fees that should be applied for borrowers who don't return equipment on time.

For this desktop application, we mainly coded using Java, Java FX, and SQL. Java FX was used to create the User Interfaces, Java was used for all of the business logic that needed to be in place, and SQL was used to modify records in a backend MySQL database. As already stated, the development process, we chose to take an Agile approach. There are many coding methodologies that we could have chosen instead, but Agile appeared to be the best for our purposes. We also were able to incorporate many different design patterns within our framework to help create extensible, reliable, and maintainable code. For my thesis, I will discuss the process that needed to be undertaken in order to create a system such as this, as well as compare and contrast various coding methodologies and frameworks that we had considered using along the way.

Methodologies

When taking on a project such as this, it can be very difficult to bring it from its conception to completion without some details falling through the cracks. Since it took so long to create, there are lots of opportunities for miscommunication to occur either between the programmers or with the customer that could lead to a product that no one is happy with. Coding projects are classified as being successful if they are completed on time with every feature functioning properly, and the project has stayed on budget. If the project takes too long, is missing features, or goes over budget, the project will be classified as challenged. Should the project not be completed at all, it is considered a failure. According to the 2009 CHAOS Study done by the Standish Group, only about 35% of all programming projects could be considered successful, 41% are challenged and 24% fail completely. For programmers, it is a constant struggle to make your project a success, but current day programming methodologies can assist us in getting the product to where it needs to be. Three of the most commonly known methodologies are the Waterfall approach, the Spiral approach, and the Agile approach. Each of these provide a guide for how you should go about the stages of the Software Development Life Cycle (SDLC) in order to create a long lasting, well-made program.

Waterfall

Of the three methodologies mentioned above, the Waterfall approach is the least likely to help you achieve success with your project. This method only allows you to go through each stage of the SDLC just one time. The stages are Requirements Analysis, System Design, Implementation/Prototyping, Testing, and Maintenance. In theory, going through each of these stages should result in a fully functional end-result. However, often times the requirements of a programming project will change between the first and

final stages, and the Waterfall approach has nothing in place to handle this. When going down an actual waterfall, once you go downstream there's no going against the current back to the top. Similarly, once you start going through the stages of the SDLC in the waterfall approach, there is no returning to any previous step. When you pass the Requirements Analysis phase, there is no going back to it even if the customer decides that something needs to be changed, which is a big issue. A visual representation of the Waterfall Methodology can be seen below.

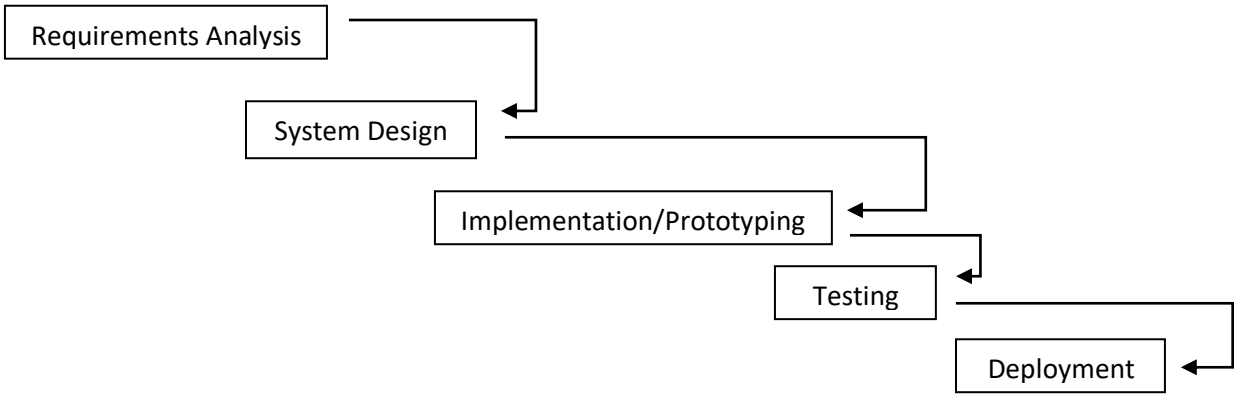


Figure 1: Waterfall Methodology

Because of the issues that accompany the use of this methodology, we did not think that it would be a viable option for our project and considered alternative methods.

Spiral

In order to fix the shortcomings of the Waterfall methodology, the Spiral Methodology was created. In this approach, you still go through each stage of the SDLC in the order they were described above, but you skip over the deployment phase. Instead, you present your tested prototype to the customer to receive feedback on what they would like to be changed, and then repeat each step of the SDLC again. You

continue this cycle until you have a prototype that the customer is happy with and then deploy it. This method is preferred over the Waterfall method because it allows you to make changes to the product, should they be needed. However, going through each stage can be very time consuming and will require lots of documentation. Another issue is that even though you repeat each step, it does not allow you to backtrack to previous steps until you've completed an entire cycle, thus making a potentially small issue take a long period of time to fix.

Considering the scale and timeframe of the project that we undertook, the spiral approach didn't seem to be the best method for our team to pursue. This led us to the conclusion that an Agile approach would best suit our needs.

Agile

Similar to the Spiral methodology, taking an Agile approach fixes the issues of the Waterfall methodology by allowing programmers to iterate through each stage of the SDLC. However, this approach has aspects that are not shared by the other methods that help to make it faster and more efficient. For starters, this approach is heavily reliant on having a very open channel of communication between the customer and the programmers. In doing so, we ensure that if at any point the customer becomes unhappy with what we are creating or would like to change the program's requirements, we no longer have to wait until we have a functioning prototype in order to address these issues. We are able to change our system's design and features as we are developing it in order to ensure the customer is given a product that they are happy

with. While this may still require additional documentation to be created, being able to go back and add it in before completing the entire program saves lots of time.

Another unique feature to the Agile approach is that it required our team to meet in a weekly session called Scrum. In these meetings our team would meet with out thesis advisor to go over any issues that we encountered the previous week, to show our progress on what we've been working on, and to plan which tasks we will accomplish for the next *sprint*. A sprint is an allotted amount of time that a programmer is given to accomplish their given tasks, which for us was a week-long period. Having these sprints is a great way to ensure that the team stayed up to date on each other's progress and made sure that we weren't falling behind schedule for any of our tasks.

Because of its speed and flexibility, the Agile approach was the best development method for our team to follow. Being able to go back and change details about our program as they were given to us proved to be crucial in helping us to meet our deadline for the program's release.

Design Process

One of the most important (and most frequently overlooked) aspects of programming is documenting your work as you proceed. When hearing about creating a new system it's easy to get carried away and just go straight to the coding. After all, that's the most fun part and the quickest way to reach your end product. However, it has been proven that without the proper documentation, a coding project becomes far more likely to fail, and will be more difficult to add on to in the future. With no record of what you've already completed, adding in new features or fixing bugs will be confusing to

someone who hasn't built it from the ground up. Even for experienced programmers, attempting to read someone else's code is a challenge without having something to go off of.

For our project, we created a series of UML artifacts that will hopefully assist future developers in maintaining our system. The documentation that we created were Use Cases, Sequence Diagrams, a State Diagram, and GUI Mockups.

Use Cases

Use Cases are the first step in documenting a new system. Use Cases are meant to clearly define each feature that the customer requested the system to have by stating the feature's name, workflow, and end result along with any preconditions you should have, any possible alternative results, and the entities that will be involved in the process described. These are important because they show us which entities are passing information back and forth and who each entity needs to be in contact with. It is also a very clear, readable way for newcomers to the system to learn and understand what it is capable of doing. An example of our first use case for Registering a Worker whom has no Banner Id in the system can be seen at the end of the paper in the images section.

Sequence Diagrams

Following the creation of our Use Cases we created a set of sequence diagrams, each of which corresponds to a different feature. A sequence diagram is used to show the flow of messages and data between different objects or processes within our system. When we actually began coding the program, these diagrams served as a map

for which classes/objects should be created and showed the exact information that they should have at their disposal to handle certain tasks.

State Diagrams

State Diagrams are meant to show the screen flow throughout the system following a user's interaction. However, these do not contain actual images of any screens, but instead contain small descriptions about what the screen's purpose would be. For example, the first screen that a user sees when opening the program is a login screen. Should they enter proper credentials, the program will then take them to a new screen containing all of the features the system can perform. Our State Diagram provides people with a visual representation to show where all events that could occur within our system will take you. Essentially, it is a map of how to navigate the program and its various features. Use Cases, Sequence Diagrams and State Diagrams are meant to help the programmers determine the architecture of the system that they are creating, and most likely won't mean very much to the customer.

GUI Mockups

Of all the documentation we've created for this project, the GUI Mockup screens are probably the most important from the customer's perspective. GUI stands for Graphic User Interface and are meant to be a template of how the completed screen will look and behave. While these don't have to be perfect, they should at least contain all of the necessary buttons and fields that the actual screen will have on it once it is completed. After creating them, we had to have a meeting with the customer to discuss each page and why we put the fields that we did on it. The customer then gave us any

changes that they would like to see made to the layout of the screen and sent us to fix them. We continued doing this until everyone agreed on a certain layout, and the documentation was completed.

Examples of our Use Cases, Sequence Diagrams, State Diagram and GUI Mockups can be seen at the end of the paper under the Images section. Having now completed all of the documentation, it was time to start coding.

Frameworks

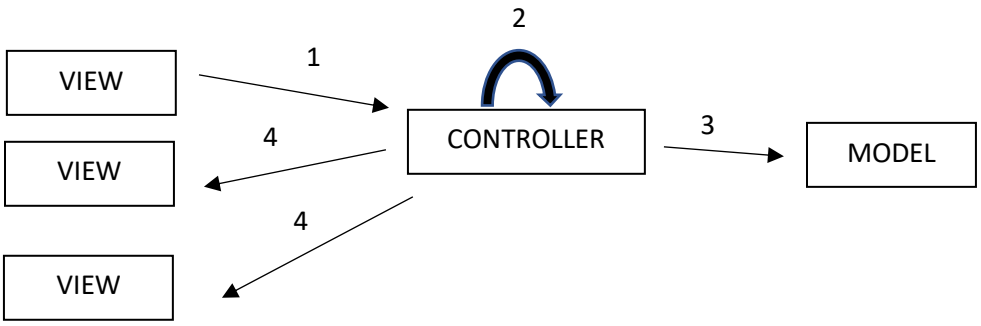
When taking on a program project, there are five standards that must be strived for. If you want your program to be successful for as long a period of time as possible, it must be stable, extensible, maintainable, scalable, and flexible. In addition to each of these, you want to strive for low coupling and high cohesion within your code. Coupling is the degree to which different parts of your system interact and share information with each other, and cohesion is the amount of "sense" that it makes for an interaction between two objects to occur given each object's individual capabilities. So, the only time two parts of the system should interact is if it makes total sense that they need to in order to accomplish some task. The days of having only one programmer working on an entire project are gone, and so the importance of meeting these standards is higher than ever to allow those who have never seen your system before be able to understand and work on it. Luckily, people have invented some frameworks and design patterns that we could follow to assist with this. For this project we utilized the MVC (Model View Controller) framework along with Java's built in Observable framework.

MVC

The issue with aiming for high cohesion and low coupling is that there isn't any universally accepted way to measure it. From looking at the architecture of a system, we can see that one method may be better than another for achieving this, but there isn't a quantitative way to say for certain that one way is definitely the "right way". Luckily, we do know for certain that by utilizing some well-known programming frameworks we can create a system with outstanding architecture that helps to achieve these standards. So, for our project we implemented using the Model-View-Controller (MVC) framework

The MVC framework is essentially a guideline on how you should logically split up the different areas of your code. The model object is used to interact with the backend database, the controller object is charge of handling any business logic that needs to take place, and the view object is the user interfaces that the system contains. When a user interacts with the screen, their action is sent over to the controller. The controller then decides how it will handle the user's action, and what calculations or data manipulation it will have to do. The controller then updates the model, with the new data. Once complete, the model notifies the controller that its update either succeeded or failed, and the controller will then update the screen to inform the user of the status of their request.

Using the MVC framework helps us to achieve the goals stated above by all but eliminating the amount of coupling between the view and the model. Since the view needs to go through the controller, the view has no reason to interact with the model at all which significantly reduces coupling and raises cohesion within the system. A diagram of how the flow of messages occurs within the MVC can be seen below.



1) The view requests an action to be done by the controller.
2) The controller manipulates and processes the data.
3) The controller informs the model of the changes that need to be made.
4) The controller updates the necessary views.

Impresario Vs. Observable

Another way we can reduce coupling is by implementing various design patterns and utilizing frameworks that help realize these patterns. The two frameworks that we had considered using for our project were Impresario, which was developed at SUNY Brockport, and Java's build in Observer framework. Both of these provide various benefits that we needed to take into consideration before choosing one or the other.

Impresario has far less coupling between the view and the controller than Java's Observer framework has due to the fact that the model and controller implement the IModel interface. The interface contains a general purpose method that takes in a key/value pair which will specify the action that needs to be performed (key) and the data that is needed to perform it (value). In addition, the controller has a list of

dependency keys which allows it to only call back views that are associated with the change that has occurred. This is more efficient than the Java Observer framework which would call back all views whether they are needed or not. Despite how much more efficient it is though, that wasn't a necessary feature for us to have in our project. That method is very useful for systems that undergo lots of stress such as multi-user game development projects, but for our application it wouldn't provide too much of a difference.

In the Java Observer framework, instead of implementing an interface, the controller extends the Observable class which does not contain any general purpose method to do the various tasks of the controller. Instead it must call a set of specific methods within the controller class itself, thus resulting in higher coupling. So instead of strictly sticking to either of these two frameworks, we decided to create our own custom framework using the best features from both. By extending the Observable class, we were able to insert our own general purpose method in order to allow for significant reuse of our view code. We still needed to keep our model and controller completely separate, but with the scale of our project, it did not prove to be a very large concern.

Conclusion

After more than an entire semester of work, we were able to deliver a system to the school's KSSPE department that they will be able to use for years to come. Due to the fact that we were careful with documenting our work and followed good programming practices, any future maintenance on our inventory management system, should it need it, will go smoothly. By utilizing tools such as Java's Observable

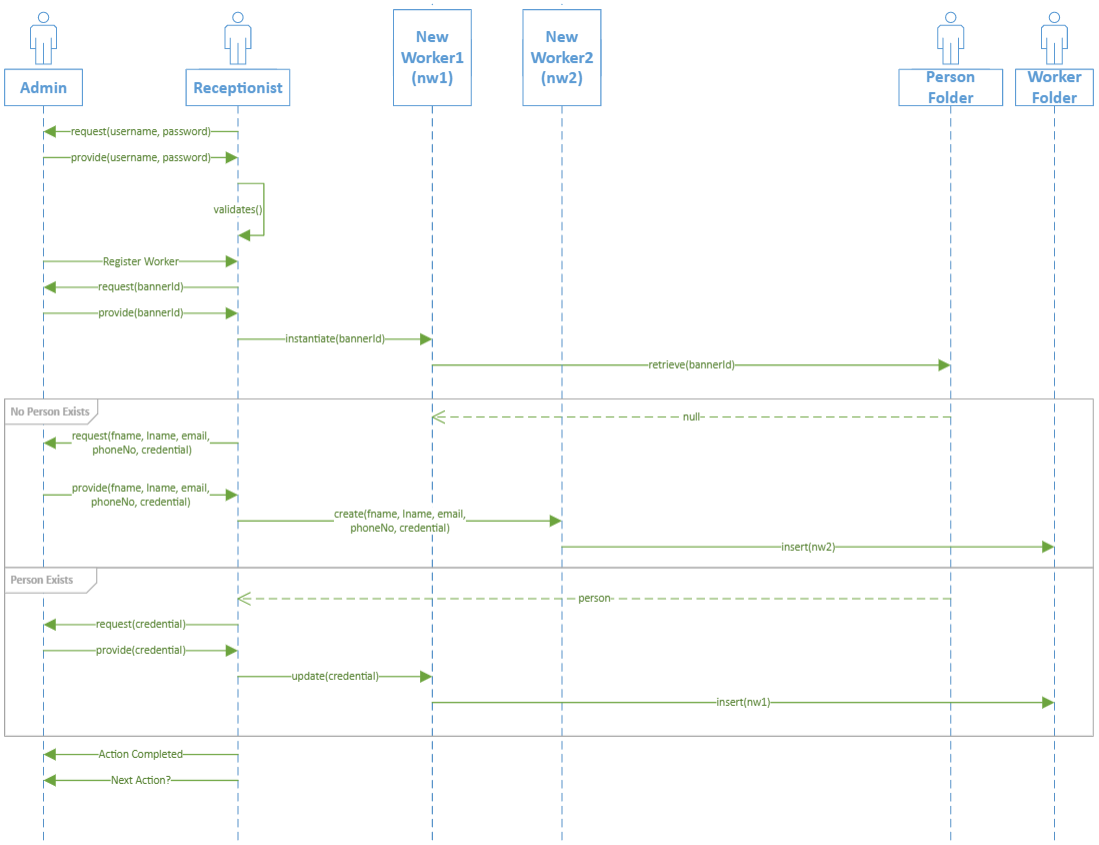
framework and the MVC framework, the application should run smoothly without having any errors for as long as the department decides to use it.

Images

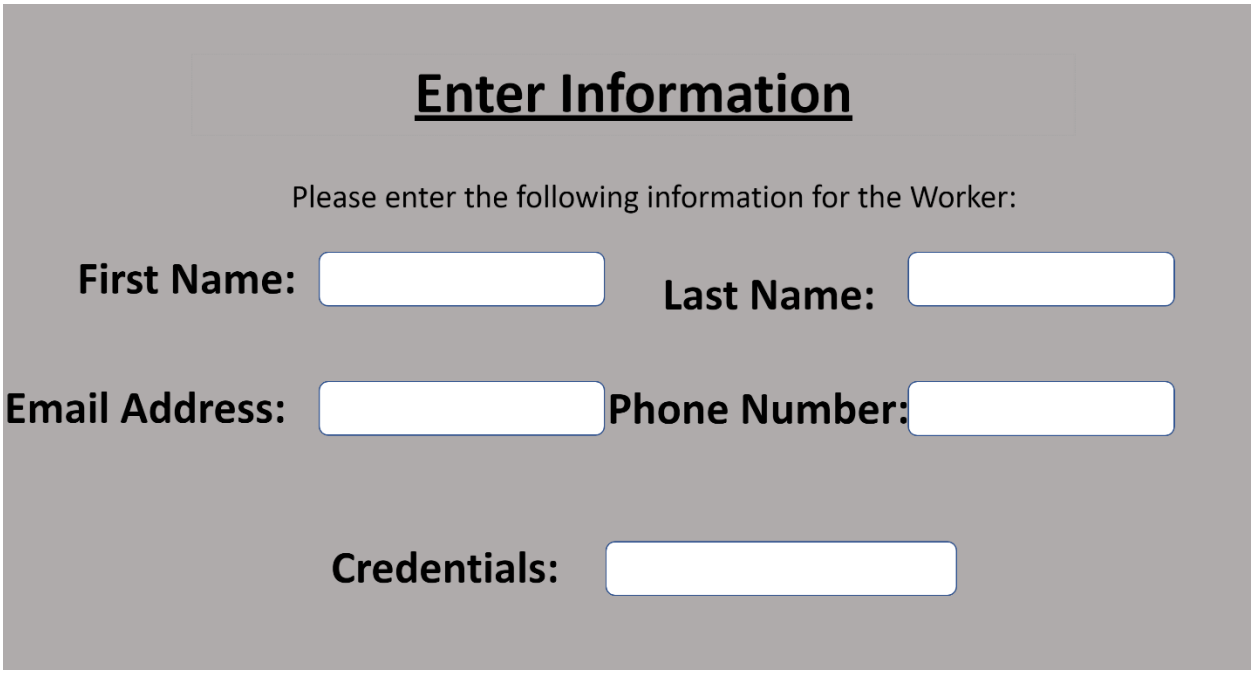
Register Worker Use Case:

Use Case Name:	1. Register a Worker Without an Existing Banner ID
Description:	
A Worker without an existing Banner ID will be registered into the system	
Preconditions:	
Workflow:	
1. Receptionist requests Administrators credentials 2. Administrator provides Receptionist with credentials 3. Receptionist retrieves the Administrator's Worker record from the information provided in step 2. 4. Administrator informs Receptionist they would like to register a new worker. 5. Receptionist verifies that the Worker file from step 3 has 'Administrator' credentials. 6. Receptionist requests the Administrator for the Worker's Banner ID, first name, last name, email address, phone number and credential. 7. Administrator provides the Receptionist with the information requested in step 6. 8. Receptionist fails to find a Person record with the BannerID provided in step 6. 9. Receptionist creates a new Person record with the Banner ID, first name, last name, email address and phone number provided in step 6. 10. Receptionist files the Person record into the Person folder. 11. Receptionist creates a new Worker record with the Banner ID and credential provided in step 6, and sets the status to 'Active' and sets the 'date added' and 'date last updated' fields to the current date. 12. Receptionist files the Worker record into the Worker folder. 13. Receptionist informs the Administrator that the Worker was successfully registered.	
Results:	
A new Person and Worker have been registered.	
Alternates:	
1. The Worker does not have Admin credentials 2. The Worker finds a Person with the provided information	
Entities Involved:	
Worker, Receptionist, "Worker" Folder, "Person" Folder	

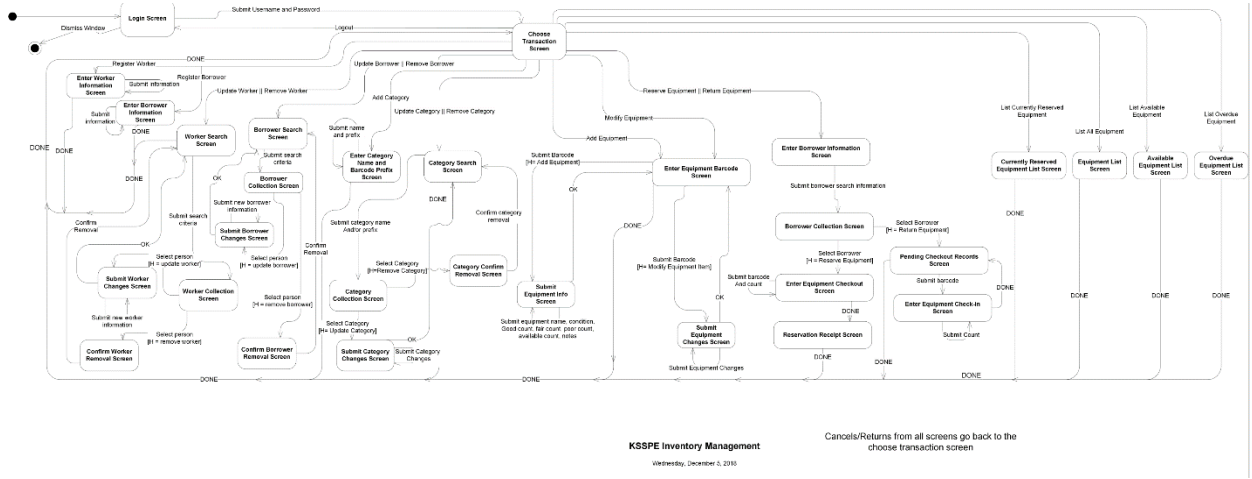
Register Worker Sequence Diagram:



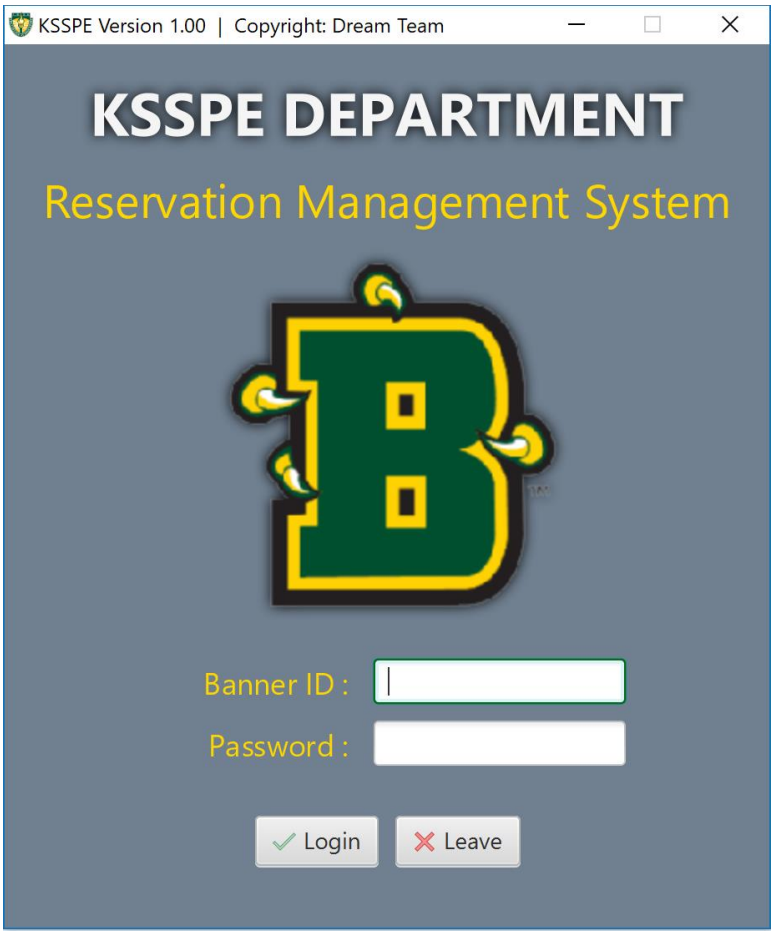
Register Worker GUI:



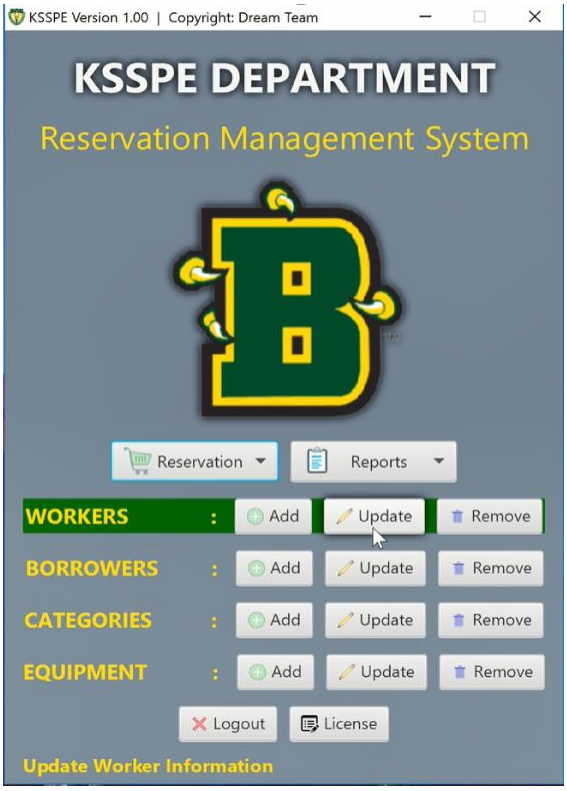
State Diagram:



User Interface, Login Screen:



User Interface, Main Menu:



User Interface, Register Worker: