Visual Network Monitoring Tool

A Manuscript

Submitted to

the CS 744 Management

Issues in Software Engineering Class

by

Kyle Van Allen, Zeya Kong, and Xianrui Zhu

May, 2018

1. Introduction

This introduction will give detailed information regarding the background information needed to understand the idea behind the project. It will also briefly describe some of the core requirements of the project that needed to be implemented into the final working product.

1.1 Background

A credit card company such as VISA or MasterCard issues thousands of credit cards to its customers. The company generally has a processing center where it keeps the database of all credit cards issued and their current details (balance, card limit, customers personal details and so on). Customers may use these cards anywhere in the world. For the simplicity of this project, the card usage is restricted to a closed region.

When a customer uses the credit card in a location, the transaction details (card details, location IP address, date and time) are sent to one of the relay stations to which the store is directly attached. This relay station then sends (actually 'relays', meaning it does not process/alter the information) the information to the processing center. At the processing center, the transaction details are processed, and a response is sent back to the store indicating the approval or denial of the transaction. Notice that there could be several relay stations between the store and the processing center. A relay station may receive several thousands of these transactions, and so it must queue the transactions first before forwarding to the processing center (or to the next relay station). Further, a relay station needs to find the shortest path to the processing center when it relays the transaction. This path may vary sometimes because another relay station on the shortest path may be inactive/down.

1.2 Requirements

- *Database*

  All data must be stored as a Required format [1]
- *Administrator Login*

  Administrator can login by inputting username, password and answering one security question out of a random three. Only after login can other functions be used.
- *Account Block*

If the user does not answer any of the three security questions correctly, the user's account should be blocked.

- *Overview of the network*

  Including showing the store, processing center, relay station and connection information.
- *Make transactions*

  After inputting correct information the user can create a transaction from a store.
- *Activate or Inactivate a relay station/connection*

  User can activate or inactivate a relay station or connection so that they cannot be used.
- *Observe a relay station*

  An administrator should be able to observe the internal details of a relay station. If selected, this operation should display the information such as the ID of the station, its current status (ACTIVE or INACTIVE), and the queue of transactions at that time.

1.2 Additional Requirements

- *Card/Account management*

  User can add/delete/modify a card or account with standard operation.
- *Shortest path*

  The constraint on "shortest path" in the problem description is now removed. So, any transaction/response can take any path from its origin to destination.
- *Regions*

  The network is now divided into regions. Every store in a region is connected only to the relay stations belonging to that region.

  The processing center does not belong to any region.
- *Gateways*

  Every region has one (and only one) designated relay station called the "Gateway" of that region. All incoming and outgoing transactions from this region must pass through the Gateway of the region.
- *Add Store*

  An administrator must be able to add a new store at any time. Similarly, an administrator must be able to add a new relay station at any time. No store or relay station will be deleted or modified.

- *Limit*

  Every relay station, even if it is a Gateway, has a limit on the number of transactions it can hold in the queue at any one time.

  An administrator must be able to change the limit on the number of transactions of a relay station at any time.

- *Other Changes*

  A store cannot be used as a relay station for moving transactions through it.

2. Implementation Strategies

2.1 Database Design

In the beginning, the primary design of the database focused on preventing redundancy of the data. This meant laying out the tables in such a way that no data needed to be repeated amongst the tables and allowing searches across multiple tables to be possible. For example, the project description states that connections can be formed between a store and relay station, two relay stations, or a relay station and a processing center. Later, gateways were also added to the mix as additional requirements. A connection table was created in the database that stored the IP addresses of each node, along with connection specific information like the weight. The connections table has no context as to what the IP of each node belongs to however, as a store, relay station, gateway, or processing center could have that IP. To resolve this the database contains a table for the store and relay station information separately, with the IP address tying all the tables together. (Note that the gateways and processing center were classified as special relay stations and were found in the relay station table).

Later in the development of the project, the idea of data loss prevention was discussed. The group decided that data should be saved into the program immediately after creation, so an unexpected browser closure or program error would not cause the loss of data users had entered. The team decided this should apply for transactions as well, and so the database was expanded to allow for the saving of in progress transaction details. This added some overhead to the overall speed of the program; however, it gave the program a great tool. Any unexpected termination of the program would have no effect on the data. All created stores, relay stations, and transaction’s positions would be remembered when the program was re-opened. The team

agreed that this was a good tradeoff for the slower animation speed, and this decision ended up heavily influencing the way the database was designed and used.

2.2 Path Algorithm Design

The question, “What exactly should the path finding algorithm return?”, was brought up several times throughout the group’s meetings. The group needed to decide between two answers, return the entire path the transaction should take, or just return the transaction’s next destination. Ultimately, the decision was made that the algorithm should simply return the next destination on the path it should take. The reason for this decision, and the design principle behind it, was adaptability. For the transaction to behave in a realistic manner, the transaction’s path must be adaptable based on changes made to the graph, such as adding, activating, or deactivating graph entities.

The decision to have the algorithm decide the transaction path one step at a time influenced the implementation of the rest of the algorithm. The group decided to implement the transaction animation by splitting it into cycles. In every cycle the algorithm would pick the oldest transaction from each node and get its next destination. Then, the transactions would be updated with their new locations in the database while the animation played for all of them simultaneously. Finally, at the end of the cycle, any transactions that entered the processing center would be decrypted, processed, and encrypted again.

2.3 Graph Structure Design

The first implementation of the graph used the third-party package Graphviz [5], to create and display the graph. Utilizing Graphviz required translating all the relevant graph data in the database into DOT language and having Graphviz translate it. While Graphviz worked well for displaying the graph, the group soon ran into issues with the ability to allow user interaction with the graph. Thus, the decision was made to switch over to a new package, vis.js. This package performed similarly to Graphviz when displaying the graph, even allowing the re-use of the DOT language that the group generated prior. It also allowed for easy click handling events on the entities created in the graph. The group found the big advantage of vis.js later though, in the implementation of datasets. These datasets allowed easy manipulation of the entities on the graph, making implementing the animation very simple. When the requirements

were changed later, and stores and relay stations needed to be added to the graph, the datasets showed their use again. They provided functions for quick and simple insertion of new graph entities, which the group used heavily in the implementation of the project

3. Technology & Tools

The team chose Java as the back-end developing language to develop this project because we are familiar with Java. Specifically, we used the Java Spring Frameworks, which are the developing tools for Java web, to develop the web application. We chose MySQL as the DB for this project because it is easy and free to use. Here are the list of the technologies and tools which we used for this network project.

3.1 Back-end

The back-end was developed using three different technologies.

3.1.1 Java

Java 9 was used to develop this project. Java is a pure object-oriented language and can be easily followed by the whole group based on previous experience with the language.

3.1.2 Spring

The Spring [2] Framework is an application framework and inversion of control container for the Java platform. It has IOC (Inversion of Control) and AOP (Aspect-Oriented Programming) features to make Java easier to use.

3.1.3 Spring MVC

MVC (Model, View and Control) model is very popular in web developing. The aim of MVC model is to solve some large web project development problems. Spring MVC is an MVC model for the Java world.

3.2 Database

The database used was MySQL and the group also used the MyBatis framework.

3.2.1 MySQL

MySQL is a lightweight and easy-to-use database for web-based SQL. For this project, after the team discussion, we decided to use this SQL database instead of a NoSQL database such as MongoDB because MySQL is suitable for the project requirements.

3.2.2 MyBatis

MyBatis is a JDBC framework to help the developer transfer database objects into java objects. We chose this because it can reduce our work and make database interaction easier.

3.3 Front-end

Every web application needs a front-end GUI. We used some typical language for the GUI and the vis.js package for the network and animation display.

3.3.1 JSP/CSS/JavaScript

JSP is the front-end page for the Java back-end instead of html. CSS is the style file to control the front-end page appearance. JavaScript can provide some ajax functions for the data interaction between the front-end and back-end.

3.3.2 Bootstrap

Bootstrap is a powerful front-end framework. It provides some GUI components to help developers build a beautiful web GUI. [4]

3.3.3 Vis.js

Vis.js [4] is a front-end third-party package for drawing an interactive graph. The project requires an interactive network to show the transaction animation, so we decided to use this package instead of drawing it manually.

3.4 Version Control and Management Tools

Because this project is a team work project, we used GitHub to control our project version.

3.4.1 GitHub

Git is a software version control tool. GitHub is the biggest git community to help manage group coding projects. We used GitHub to merge our work and manage the software versions.

3.4.2 Maven

Maven is a software package management tool. We used Maven to manage our developing packages.

4. Challenges

There were many challenges met during development but all of those were solved successfully because all team members payed close attention to the project and we had frequent communication and close cooperation. The group ended up redesigning the whole project and reimplementing it more than three times during the semester. Here is the list of the 3 biggest challenges that the group faced during development.

4.1 Unreliable Third-Party Package

During the third demo sprint, we found the third-party package we used called Graphviz.js [5] could cause some unexcepted errors and make the whole project crash. When the errors first occurred, we didn't want to change the package we used because some front-end and back-end code had some coupling with this package. The group tried to locate all the bugs and fix them. Unfortunately, we couldn't solve it because the package had more than forty thousand lines code and it had been obsoleted. So, we were determined to find another new third-party package and refactor the code. Finally, we found vis.js and redid our work. It spent about a whole sprint refactoring the front-end and back-end code and we finished all the sprint tasks for the third demo by having a heavy work load.

4.2 Work Allocation Challenges

There are often problems with how to separate a team's work to make sure each person has an equivalent workload. Our team met this problem and we dealt with it successfully through communication and mutual understanding, but it was still a big challenge we met. When we

started the project, it was divided into three parts: database, back-end and front-end. Because we were at the beginning of the project, this distribution method was very good. Each team member had a heavy workload. As the project continued, we found this separated method no longer made sense because the DB work became less, and the front-end work became heavier. So, after the second demo we discussed this distribution problem, saw other groups' experiences and we reached an agreement to use a new distribution method to separate the project. This divided the project into various parts according to the different functionalities such as a login part, a network showing part and an animation showing part. We started our work by using this way during the third and fourth demo, but we still ran into trouble. After the fourth presentation, feedback pointed out that each person's work looked very independent because we just focused on our own part and ignored the integration and data interaction. With both previous systems not working, what should the group try next? We set up a meeting to talk about the distribution method immediately after we got the comment from Dr. Kasi and the group thought it was necessary to change the work allocation again. We thought that the previous idea was good and the only thing bad thing about the distribution was because the different functionalities of this project have strong relationships. After the team discussion, we found a more specific way to separate our work, one person would handle the account management part and other two focused on the network functionalities. The network functionalities were then divided into back-end and front-end. We used this way to finish the project and we thought this division method was very good and it was suitable for our team's strengths and weaknesses.

4.3 Data Consistency

The origin of the group's concern with data consistency was from the project requirements we discussed. We thought the system should store the network status when an admin logged out. So, we emphasized the back-end storing procedures. We introduced this design and started our implementation, but we spent most of our time implementing and fixing this functionality. We found that this functionality might cause many bugs which belongs to the database, the back-end and the front-end. We tried to fix it in the back-end and DB by refactoring the back-end code 3 times, but it never quite worked. Finally, we fixed it by redesigning the front-end code.

5. Learning

This section will detail the different lessons that were learned by members throughout development of the project. This section is divided into three smaller sections, one for each member of the group so they can each explain their own lessons learned.

5.1 Kyle Van Allen

Throughout the project I was exposed to many firsts in my programming career. I'd like to describe my thoughts on some new technologies I encountered during this project, and my first team-based programming project.

5.1.1 New Tools Used

This project was my first experience creating and deploying a web-based project onto a server. While I had limited knowledge of html, CSS, and Javascript from previous courses at UWL, I was rusty with regards to those as well. This unfamiliarity with the syntax of the languages proved tricky at first and was one of the leading reasons I elected to focus mostly on the database for the first 2 sprints. After some reading and refreshing of the material however, I found myself able to both write my own code and understand the code of my group-mates much better.

Along with the new front-end technologies, I also had my first experience with the Java Spring framework. This framework was, to be frank, quite frustrating to deal with throughout the project, especially toward the beginning. Zeya was very helpful in giving Xianrui and I pointers on how Spring functions, but even with the help I struggled to understand the workings of Spring until about halfway through the project. This slowed down progress for the group, as I was unable to meaningfully contribute to the back-end development of the code without Zeya's help, for a large amount of the project.

I think the most important take away from this is just how important it is to make sure the tools are decided upon early in the project. I expect that for almost every project there will be at least one person who is unfamiliar with one or many of the tools that is proposed to be used. While learning these tools on the fly to use in the project is possible, it is quite time consuming,

so it is preferable to have sat down and familiarized oneself with the tools before development on the project has begun. This will help that person contribute to the project more consistently throughout development. It will also reduce the amount of time that must be spent refactoring bad code that was created when knowledge of the language was still limited. (Something I personally had to do several times for the Javascript I created throughout the development of the project).

5.1.2 First Group Project

The differences between a project that is done solely by one person and a project that is done by a time, are too numerous for me to list here. It completely changes the dynamic of how a project will be developed, and the creation of software can either be sped up or drastically slowed down based on how effective the team is at working together. The biggest takeaway for me from this experience is the affirmation of what we were learning in class throughout the semester. To be specific, the importance of proper planning and documentation over the lifecycle of a project. My guess is that the biggest failures that software projects run into are caused by a lack of communication amongst the group. Miscommunication will lead to incompatible code, missed functionalities, and buggy software. Communicating through words alone is simply not enough when the survival of the software is on the line, and that's where the documentation fills in the gaps. With the documentation, not only can a team keep track of what they need to do next, but they can also see what they have done before and the results that it leads to.

5.2 Xianrui Zhu

For this project I mainly did the work of GUI design and the part of Card/Account management, along with some initial work on the network.

5.2.1 Technical Learning

I do not have a good understanding of project development, as I lack experience. For this project I was glad to do the card management work because this part is suitable for me. It is not too simple while also not being too hard for me to understand. Work on this section

included coding with the database, front-end, and back-end. Some things I learned from this project were as follows.

- Total understanding of how every part of SSM works.
- I acquired experience with fixing bugs and integrating parts of code together.
- Some database knowledge, like the keyword 'limit', or the key constraint which caused me a lot of trouble.
- I learned how to use AJAX functions effectively.

5.2.2 Teamwork Learning

For our team I primarily took the role of a listener. I think our team worked well together, with each of us playing a key role. Zeya was like a leader, he was positive and was always helping us in coding our parts. Kyle was like a secretary, he planned what to do next and did the document work nicely. I did not want to explain too much in the meeting because I did not want to explain something that everyone already knew, and often my teammates already had better ideas. I would usually sit and listen, filling in the gaps if they had some blind spots when planning the next sprints. Some things I learned about having effective meetings are:

- Listen and think before talking, it will make the meeting more effective.
- People should try to be positive in meetings, to avoid unnecessary conflict.
- Just listening will lead to mistakes in thinking, talking will help clarify details and mistakes during meetings.
- Doing some drafts during meetings can make ideas clearer.
- Schedules can be adjusted to avoid some meaningless meetings when necessary.

- One things I want to mention is in the latter part of project, I did a lot of individual work on my part while Zeya and Kyle worked on theirs. Then, when Zeya and Kyle were talking in meetings, I did not have any good ideas for their transaction work, so I didn't speak up as much. I often visited Zeya's dormitory for integration and talked about some functionality requirements, but this is not recorded in the meeting logs.

5.3 Zeya Kong

After the team discussions, the stand out thing we learned is that Java will make you crazy when you use it to develop web applications. From my personal perspective I learned a lot from this project, not only about the technologies but also about the importance of team cooperation. I learned that communication is an effective way to solve some problems and if each team member tries their best for the team, the project will turn out very nicely, even if different people have different skills. I also learned two more very important things from this project.

5.3.1 Integration Is Not Easy

I did a lot of work with integration during the whole developing processing. I felt it was very difficult to do. For example, during the second demo sprint, I tried to merge the front-end and the back-end code together but when I put those code together, the whole project crashed. I was sure that my back-end code was right. I ended up blaming the team member who had the responsibility in the front-end. He thought his code was OK and it could run on his machine. The only thing I could do was just to debug all the code and try to find the bugs. It was very frustrating to debug because the readability of the front-end code was very poor, and I couldn't understand anything from his code. The same thing happened when I merge the database and the back-end. Both of us were sure that the code could work independently, but they still couldn't work together. Finally, I found that the reason was we used 2 different versions of MySQL. At the end of the sixth sprint, after I merged the account management part into the master branch, all things began to crash, and I spent a long time on debugging. From these experiences, I have realized many reasons why integration can be difficult.

5.3.2 Testing Is Very Important

This one is related to the previous one. After I found out that integration was very difficult, I tried to find the reasons why problems were occurring. After we finished the coding for the project, I think I have found my answer for this question, and its testing. For example, if we did more unit testing before the integration, we could easily find many of the integration bugs. Without testing, the integration becomes harder because the two parts themselves have some

errors. The integration testing is also necessary because if we ignore this part, some potential bugs will get more troublesome as the project becomes bigger and bigger. In addition, the final project testing is also important. In our specific case, we did a lot of debugging overnight before the last presentation, so we didn't have enough time to do the testing when we deployed the project into server. Because we didn't do enough testing before, we spent far too much time on debugging. This meant the project had some obvious errors when we presented our final demo. Overall, this course showed me how important the testing is.

6. References

[1] Kasi Periyasamy, CS 744 - Management Issues in Software Engineering[Online] Available: http://faculty.cs.uwlax.edu/~kasi/cs744/cs744-18/Project%20Description.pdf, 2018. Last Accessed, April 30th, 2018.

[2] Spring Community[Online]. Available: https://spring.io, 2018. Last Accessed, April 30th, 2018.

[3] Bootstrap[Online]. Available: https://getbootstrap.com/, 2018. Last Accessed, April 30th, 2018.

[4] Vis.js[Online]. Available: https://visjs.org, 2018. Last Accessed, April 30th, 2018.

[5] Graphviz[Online]. Available: https://www.graphviz.org, 2018. Last Accessed, April 30th, 2018.