TEST DOCUMENTATION

PREPARED FOR DAYTRADING INC

PROJECT NAME:	DAY TRADING SOFTWARE
SYNOPSIS:	AN OUTLINE OF THE PLAN USED TO TEST THE PROJECT, THE CONDITIONS BEING TESTED, AND THE RESULTS ACHIEVED.
DATE:	APRIL 7, 2015
PREPARED BY:	BRAIGHTON POLACK (V00763687), EVAN WILLEY (V00703788)

CONTENTS

1. INTRODUCTION

The purpose of this document is to outline the testing plan for the Day Trading Software. The testing was performed on a by component basis to test the functionality of each component, then on the whole system once completed. The general basis for this documentation is to create a separate testing outline for each component of the software, to describe how each was tested. Performance testing is also described briefly in this report, and in more detail in the performance report.

2. TESTING BY COMPONENT

The individual testing of each component is described in the section below. This testing was performed at regular intervals during development to find bugs and errors in each component.

2.1. LOAD BALANCER

The method used to test the functionality of the load balancer throughout the span of the project involved a script that generates sequential quote requests as fast as possible. For example, it starts with stock symbol AAA, then AAB, AAC, etc. Using this script we were able to confirm that each quote request was being returned in the correct order. We could also measure the maximum speed that the load balancer could retrieve quotes. The performance of the load balancer is discussed further in the performance report.

2.2. DATABASE

The database was tested for functionality by performing each type of PostgreSQL command from the command line, and checking the results in the tables. After each test run using a workload file, the tables were all cleared, so the data could be tested for consistency. More testing was done for each particular user command with the system as a whole, as the specific user contents of the database could be more easily tested with the web interface.

In the case of the brief usage of Postgres-XL, the testing also involved shutting off a datanode to determine whether the database could function during a crash.

2.3. LOG SERVER

The log server was tested with the use of the transaction server and workload generator, by running an entire workload file worth of user commands we were able to stress test the log server. Sending individual commands allowed us to test each of the log types below.

- User Command Log
- Quote Server Log
- Transaction Log
- System Event Log
- Error Event Log

The debug type log went unused. Once it was confirmed that each log type was writing properly to the MiniXML linked list and the xml file; as many commands as possible were sent, to determine if there was any point it would fail. Through this stress testing, it was found that our implementation of a single log server thread receiving connections, and adding logs to a queue, caused the log server to fail under high load. This was later rectified by giving each component a log queue, and a single threaded connection sending logs to the log server.

2.4. WEB SERVER

The web servers were tested first by themselves, ensuring that the html inputs, and web interface all functioned correctly. Each command on the web interface was tested, with the inputs printed to a console, to make sure each input was being posted correctly. Testing to make sure each input was filtered correctly was done with every change, and that incorrect values or symbols were removed accordingly. Each command that is constructed by the php on the web server was tested, to make sure the format of the message was correct before sending the commands to the transaction servers. With every change of the messaging protocol, this was retested by printing out the constructed command and checking the format.

The web servers were also tested in tandem with the other components of the system to ensure the responses sent to the user were correct in every possible case. Every combination of possible buys, sells, confirms, cancels, and set triggers were tested. Spamming the transaction servers with these commands in random order was also tested. The most important part of this testing was to ensure the time calculation on pending buy/sells was displaying correctly. A countdown until the quote becomes invalid is displayed for each buy and sell command sent, this countdown timer will stay consistent despite any refreshing, changing pages, or executing other commands. Expiry of the quote is also checked on the transaction server itself, but a goal of this system is to minimize any end user confusion or frustration in what they see on the web interface.

3. SYSTEM TESTING

The system was tested as a whole unit using the workload files provided by the project guidelines, as well as on a per command basis using the web interface. After every major revision, typically before each workload milestone, the commands were tested individually through the web interface. This helped in finding several bugs that would have otherwise gone unnoticed. For example, during some of the final testing there was no data being written to the stocks table. This was noticed immediately when the buy and sell commands were tested on the web server. The transaction server in particular could only be tested with the complete system, as it only exists to serve other components, so much of the testing was done in this way.

Performance and stress testing was done using the workload files provided. If any errors in consistency or crashes occurred, the area affected was tested more closely individually. Testing the system was extremely important at each milestone, as more than half the work put into the project was debugging.

Fault tolerance was tested on the complete system by running a workload file, and disconnecting components one by one from the network. This allowed observation of the other components tolerance to a failure, and how they redistributed load, or paused completely and returned an error.

4. CONCLUSIONS

In conclusion, testing of the system was done regularly and thoroughly. Debugging in C can become easily confusing, so it was beneficial to be able to separate each component out and test most individually. This testing was done mostly prior to each workload milestone, as each larger workload file revealed new issues and stresses on the system.