COMP 489

TME 3 Test Plan

Jason Bell 3078931

February 28, 2016

This implementation of a P2P web service does not use JBoss, but unfortunately due to family issues I’m submitting this at the 11th hour (just about literally) and didn’t have time to ask for permission. Instead, it uses Apache Tomcat 7. I hope this is acceptable, considering it’s extremely widely used and robust.

The reasons that I did not use JBoss are:

1. It has become a commercial product. I was going to have to use an open source alternative such as Wildfly anyways. However, Wildfly has terrible documentation, and the server would not start: it just gave me ModuleNotFoundExceptions for javax:main, even though I followed every tutorial I could find to the letter. I could not find any solutions to this problem. So, Tomcat. It was extremely painless to implement, and is basically ubiquitous, so I hope that’s ok!
2. The article that you are sent to study to learn how to use JBoss is completely useless. Not only is it obsolete, but the contents of the web.xml file that it tells you to use are totally garbled. Some sort of encoding errors. The double quotes that are double quotes are in some Unicode that eclipse doesn’t recognize so had to be replaced, and the strings don’t end in double quotes, they end in things like “#8221”. Refer to <https://ankitshrivastava.wordpress.com/2008/07/24/creating-webservice-using-jboss-422-and-eclipse-europa/> . Even when you correct all of that, the web.xml Just Plain Doesn’t Work ™.

So, between this is and time constraints, I decided to go with a more robust and widely used server.

(Seriously, I do hope that between learning from Wikipedia articles, and all the broken urls, that this course gets revised soon!).

One more note before I get to the testing: you’ll find these tests identical to that of TME2. The two assignments are basically the same, but with different communication protocols, so in TME2 I kept things modular so I could just swap out the CORBA code for web server code. Definitely a time when keeping things uncoupled is beneficial ☺ . The database is identical as well: if you still have it from TME2, you won’t need to recreate it.

You may have to change P2PServer’s reference to mysql-connector-java-5.1.38-bin.jar, found in the workspace/P2PServer folder, since Eclipse appears to be insisting on hardcoding it’s location to the absolute path on my computer.

The SQL script p2p.sql will create the MySQL database and two user accounts. If the connection string needs to be changed, it can be found in line 20 of Database.java of the P2PServer package.

Two pre-built clients can be found in the Clients folder. For debugging, I ran the server in eclipse and the clients from the command line. I have hardcoded the clients to default to listening on port 3000, and using 3001 and 3002 for the jasonb/asdf and angelinaj/asdf users. This is so that they can both be run on the same machine for ease of testing.

For this first test, do not have the server running!

1. Test that the client gracefully displays an error message when it’s unable to connect to the server.
2. Start a client on the command line with “java -jar client.jar”
3. Ensure that a graceful connection error is displayed

For the following tests, have the server running, whether on the command line or from within eclipse.

1. Test that clients connect to the server and can log in successfully.
2. Start a client on the command line with “java -jar client.jar”
3. Login with credentials of either jasonb/asdf or angelinaj/asdf
4. Be presented with an authorized message, the IP and port you are using, and an options menu.
5. Test that clients that attempt to connect to the server with invalid credentials are rejected.
6. Start a client on the command line with “java -jar client.jar”
7. Login with any credentials other than the ones from the previous test.
8. Be presented with a rejection message.
9. Test that you can view all files currently owned by the client (in the Files folder).
10. Start a client on the command line with “java -jar client.jar”
11. Login with credentials from test 2.
12. Enter 1: “view all files”.
13. Be presented with a list of files in the Files folder, some possibly flagged as shared.
14. Verify that these match the files found in the Files folder for the client being tested.
15. Test that you can flag a file as shared.
16. Start a client on the command line with “java -jar client.jar”
17. Login with credentials from test 2.
18. Enter 2: “share a file”
19. Be presented with a list of files in the Files folder that are not currently shared
20. Enter the number beside the file to share.
21. Enter 1: “view all files”
22. Confirm that the file is now flagged as shared
23. Test that you can unflag a file as shared.
24. Start a client on the command line with “java -jar client.jar”
25. Login with credentials from test 2.
26. Enter 3: “unshare a file”
27. Be presented with a list of files in the Files folder that are flagged as shared.
28. Select a number beside the file to stop sharing
29. Enter 1: “view all files”
30. Confirm that the file is no longer flagged as shared.
31. Search for a file that you already own.
32. Start a client on the command line with “java -jar client.jar”
33. Login with credentials from test 2.
34. Enter 4: “search for a file”.
35. Enter a filename found in the Files folder.
36. Be presented with notification that you already own this file.
37. Search for a non-existent file.
38. Start a client on the command line with “java -jar client.jar”
39. Login with credentials from test 2.
40. Enter 4: “search for a file”.
41. Enter a filename not found in the Files folder of either client. For example, some sort of gibberish filename.
42. Be presented with a notification that the file does not exist.
43. Attempt to download a shared file while no peer owning the file is connected.
44. Share a file with one peer, that is not shared by another (see test 5).
45. Shut down that peer.
46. Start another peer, and search for that file.
47. Enter “y” to attempt to download.
48. Be presented with a message about inability to connect to any peers with this file (the client is given a list of possible addresses by the server, and tries each).
49. Download a shared file while a peer owning that file is connected.
50. Share a file with one peer, that is not shared by another (see test 5)
51. Keep that peer running.
52. Start another peer, and search for that file.
53. Enter “y” to attempt to download.
54. Be presented with a completion message.
55. Verify that the file is now found in the Files folder.