**Project Engineering – Journal**

**Week 1 (7th Oct – 13th Oct)**

**Overall Task**

Using Docker, set up local instance of Apache Cassandra database with basic command line access.

* Learn about Docker, Cassandra set-up and basic DB interaction through command line
* Complete installation of Cassandra instance
* Capable of executing basic DB commands

**Work Done**

* Installed Docker on local machine
* Installed Kitematic (graphical user interface to run and manage containers in Docker) on local machine
* Used Kitematic to install container using image from Docker image library
* Used Homebrew on the Command Line Interface (CLI) to install Python (brew install python)
* Used Homebrew to install Apache Cassandra (and CQLSH, Cassandra Query Language Shell)
* On DB, set up Keyspace and first table
* Executed commands to insert and delete data from table

**Issues & Solutions**

Connection error when trying to access local Cassandra instance from CLI. This was because the Docker container ports were not exposed to an external connection.

*Solution:* The exposing or publishing of ports for a Docker container cannot be done on a running container. It must be done at container creation. Running the following command created a new container running Cassandra and having the port 9042 open for connection:

* docker run -p 9042:9042 --name cassandra -d project-local-cassandra



**Week 2 (14th Oct – 20th Oct)**

**Overall Task**

* Use IntelliJ to execute DB commands
* Design and setup initial tables for project data storage
* Display data from successful DB read command

**Work Done**

* Configured IntelliJ project to include local Cassandra data store
* Imported DataStax driver jar file into IntelliJ project
* Made multiple attempts to correctly configure DB to allow connection from a Java application to execute queries/statements

**Issues & Solutions**

* Multiple compile-time errors when trying to establish Cluster connection with Cassandra instance. Errors are primarily to do with logging configuration and missing JAR files.

*Solution*

At present issue remains unsolved after downloading and adding multiple JAR files to project structure in IntelliJ.

**Week 3 (21th Oct – 27th Oct)**

**Overall Task**

Resolve previous Cassandra DB configuration issue in order to:

* Use IntelliJ to execute DB commands
* Design and setup initial tables for project data storage
* Display data from successful DB read command

**Work Done**

* Continued attempt to configure DB by manually adding JAR files, libraries and dependencies to IntelliJ Java Project. **I was unable to accomplish successful setup by this method.**
* Created a new Maven Java project in IntelliJ.
* Added needed Cassandra dependencies (DataStax) to project pom.xml file.
* Imported extra maven libraries to successfully configure project to connect to DB instance.
* Ran basic DB class to create a test keyspace in DB. This was successful.
* Used IntelliJ to create a keyspace and a table within that keyspace

**Issues & Solutions**

1. The project-to-DB connection config issue of last week continued into this week. The primary exceptions and errors were around logging and missing class definitions. After researching many hours online, the problem and solution seemed to be outside the scope of the project.

*Solution*

Once the complexity of the problem was understood I decided that I may change to MySQL rather than use Apache Cassandra as my project database. Cassandra is an enterprise-level application and as such, configuration was proving time-consuming.

My final attempt to solve the issue before the I made the switch yielded the result I wanted. I created a Maven Project in IntelliJ and within the dependencies of the pom.xml file I added the needed DataStax driver libraries and logging libraries. Once these imports to the project were made I was able to connect to my local Cassandra instance and execute DB queries/statements.

**Week 4 (28th Oct – 3rd Nov)**

**Overall Task**

* Design and setup initial tables for project data storage
* Display data from successful DB read command

**Work Done**

* Added code to read and write to DB
* Printed DB read data to console in IntelliJ





* Introduced Prepared Statements to DB code
* Added basic server class to project
* Added basic mock client application to test server functionality
* Added Database Access Object classes
* Successfully sent string message from mock client and stored in DB table

**Issues & Solutions**

No Issues

*Solution*

**Week 5 (4th Nov – 10th Nov)**

**Overall Task**

* Use mock-client to simulate basic user registration
* Successfully write multiple messages from user(s)
* Display messages to mock-client console(s)

**Work Done**

* Added code for basic user authentication
* Added user registration feature
* Added User class
* Implemented both username and password authentication

**Issues**

Passwords for a user in the DB are currently being stored as strings that are visible upon query. I need to implementing some kind of hashing algorithm to make passwords unreadable and more secure.

**Week 6 (11th Oct – 17th Nov)**

**Overall Task**

* Successfully retrieve message details from DB table for a message, send from server to mock-client and print to console for the user

**Work Done**

* Successfully “logged-in” as an already registered user, sent a string message from mock-client to server as user, stored message in messages table along with user\_id of the user.

**Issues & Solutions**

Server code throwing the following exception: com.datastax.driver.core.exceptions.InvalidQueryException.

The root of the problem is incorrect DB schema design. Cassandra is designed to provide fast read performance through querying a table’s PRIMARY KEY. The exception here is being thrown due to querying a field that is not part of the PRIMARY KEY.

*Solution*

The short term solution is to add “ALLOW FILTERING” to the query string to be executed. On a large-scale basis, data filtering could produce performance issues for the DB. For this project it can be used to provide an immediate solution. This isn’t best practice however, so at a later stage I plan to remodel the necessary DB tables to allow for more efficient querying.

**Week 7 & 8 - (18th Oct – 1st Dec)**

**Overall Task**

* Code is currently at demonstration level for end of semester
* Goal is to attempt implementation of Tomcat server into project for demo
* Make any necessary improvements to project for Christmas demonstration

**Work Done**

* Added a simple GUI to mock-client
* Added GUI functionality to send message to local server application and store in DB
* Server application returns stored string message to client, GUI is able to display returned string

**Issues & Solutions**

While trying to implement the client GUI I encountered this issue:

* An infinite loop is needed to prevent the client and server programs from exiting prematurely. If it exits, the socket channel being used to transfer data closes on the client side or server side. This will cause broken pipe exceptions. From this, a bug was causing the server to “receive” limitless empty strings from a broken client. This fills up the DB table with faulty entries in a very short period of time. However, using loops will cause the GUI to not render or function correctly. So for a GUI, no loop is allowed.

*Solution*

* Loop removed from the client code and server code, allowing GUI to function correctly.
* To fix the broken pipe problem: Instead of a socket channel being established per client application, a channel is opened from client to server each time the send button is pressed by the client. The server processes the request, then no longer uses the thread for the current request. The server waits for a new request from the client. Socket channels connections are made per each message being sent. This eliminates the bug of endless empty strings being received by the server.