Chanun Sumphanphaisal 25051704
Elton Xue 52611936

CS122B Project 5 Report

Task 1

For Task 1, I incorporated connection pooling and prepared statements in a variety of areas in order to boost the efficiency of Fablix. Most notably, I improved the speed of the application by implementing these in servlets that require searching.

- /FullTextSearch

```
92
93          try {
94  //          Class.forName("com.mysql.jdbc.Driver").newInstance();
95
96  //            Connection dbcon = DriverManager.getConnection(loginUrl, loginUser, loginPasswd);
97
98              // the following few lines are for connection pooling
99              // Obtain our environment naming context
100
101             Context initCtx = new InitialContext();
102             if (initCtx == null)
103                 out.println("initCtx is NULL");
104
105             Context envCtx = (Context) initCtx.lookup("java:comp/env");
106             if (envCtx == null)
107                 out.println("envCtx is NULL");
108
109             // Look up our data source
110             DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
111
112             if (ds == null)
113                 out.println("ds is null.");
114
115             Connection dbcon = ds.getConnection();
116             if (dbcon == null)
117                 out.println("dbcon is null.");
118
```

Lines 101 - 117 /Project1/src/FullTextSearch.java:

For the “/FullTextSearch” servlet, I incorporated connection pooling by creating an Context object and then using that to retrieve a DataSource object associated with “jdbc/TestDB”, which is in the “context.xml” file. After successfully looking up the data source, I got the existing database connection from it to engage in connection pooling.

```
146     String stars_query = "SELECT name FROM (SELECT starId FROM stars_in_movies WHERE stars_in_movies.movieId = ?) AS starIds JOIN stars ON starIds.starId = stars.id";
147     String genres_query = "SELECT name FROM (SELECT genreId FROM genres_in_movies WHERE genres_in_movies.movieId = ?) AS genreIds JOIN genres ON genreIds.genreId = genres.id";
148
149
150     PreparedStatement stars_pst = dbcon.prepareStatement(stars_query);
151     stars_pst.setString(1, m_id);
152
153     PreparedStatement genres_pst = dbcon.prepareStatement(genres_query);
154     genres_pst.setString(1, m_id);
155
156     ResultSet rs_stars = stars_pst.executeQuery();
157     ResultSet rs_genres = genres_pst.executeQuery();
158
```

```
204             String searchStarsQuery = "SELECT * FROM stars AS alias WHERE MATCH(name) AGAINST (? IN BOOLEAN MODE)";
205
206             PreparedStatement search_pst = dbcon.prepareStatement(searchStarsQuery);
207             search_pst.setString(1, matchString);
208
209             ResultSet rs = search_pst.executeQuery();
210
```

Lines 146 - 158 and 204 - 210 in /Project1/src/FullTextSearch.java:
For prepared statements in “/FullTextSearch”, I implemented it in the above lines. From lines 146 - 158, I implemented it by creating 2 prepared statements for searching for the stars and genres associated with a movie when doing a full text search. From lines 204 - 210, I created 1 prepare statement in the case that a user clicks on a star when doing a full text search.

- /Search

```
86
87          try {
88              //Class.forName("com.mysql.jdbc.Driver").newInstance();
89
90              // the following few lines are for connection pooling
91              // Obtain our environment naming context
92
93              Context initCtx = new InitialContext();
94              if (initCtx == null)
95                  out.println("initCtx is NULL");
96
97              Context envCtx = (Context) initCtx.lookup("java:comp/env");
98              if (envCtx == null)
99                  out.println("envCtx is NULL");
100
101             // Look up our data source
102             DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
103
104             if (ds == null)
105                 out.println("ds is null.");
106
107             Connection dbcon = ds.getConnection();
108             if (dbcon == null)
109                 out.println("dbcon is null.");
110
```

Lines 93 - 110 in /Project1/src/Search.java:
Almost identical to the “/FullTextSearch” servlet, the “/Search” servlet also uses connection pooling in the same way, where you create a DataSource from Context and get the existing connection from the data source.

- /SearchSingleStar

```
64
65          try {
66              //Class.forName("org.gjt.mm.mysql.Driver");
67  //              Class.forName("com.mysql.jdbc.Driver").newInstance();
68
69  //              Connection dbcon = DriverManager.getConnection(loginUrl, loginUser, loginPasswd);
70
71              Context initCtx = new InitialContext();
72              if (initCtx == null)
73                  out.println("initCtx is NULL");
74
75              Context envCtx = (Context) initCtx.lookup("java:comp/env");
76              if (envCtx == null)
77                  out.println("envCtx is NULL");
78
79              // Look up our data source
80              DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
81
82              if (ds == null)
83                  out.println("ds is null.");
84
85              Connection dbcon = ds.getConnection();
86              if (dbcon == null)
87                  out.println("dbcon is null.");
88
89              String searchQuery = "SELECT * FROM stars AS alias WHERE name LIKE ?";
90              System.out.println(searchQuery);
91
92              PreparedStatement pst = dbcon.prepareStatement(searchQuery);
93              pst.setString(1, "%" + star + "%");
94
95              ResultSet rs = pst.executeQuery();
```

Lines 71 - 95 in /Project1/src/SearchSingleStar.java:
For connection pooling, the “/SearchSingleStar” servlet uses the same idea as connection pooling in “/FullTextSearch” and “/Search”. Near the end of the code, I also used prepared statements so that I could search for a single star when a user clicks on that star’s name in the MovieList. I use prepared statements by creating it and then setting the first string to be the star’s name.

- context.xml

```
1  <Context path="/Search">
2
3      <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
4          maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
5          password="cs122bfablix" driverClassName="com.mysql.jdbc.Driver"
6          url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />
7
8  </Context>
9
10
```

Entire “context.xml” file in in /Project1/WebContent/META-INF/context.xml:
Inside this file, I manage the connection to the database. The username for my database is root and the password is cs122bfablix.

- web.xml

```
13  <resource-ref>
14    <description>
15            Resource reference to a factory for java.sql.Connection
16            instances that may be used for talking to a particular
17            database that
18            is configured in the server.xml file.
19        </description>
20    <res-ref-name>
21            jdbc/TestDB
22        </res-ref-name>
23    <res-type>
24            javax.sql.DataSource
25        </res-type>
26    <res-auth>Container</res-auth>
27  </resource-ref>
```

Lines 13-27 in /Project1/WebContent/WEB-INF/web.xml:
In this file, I created a “resource-ref” tag in order to create a reference to “jdbc/TestDB” so that my servlets can refer to it.

Task 2

AWS:

- Load Balancer IP: 54.176.223.140
- Master IP: 54.176.223.140
- Slave IP: 54.241.187.173

Google:

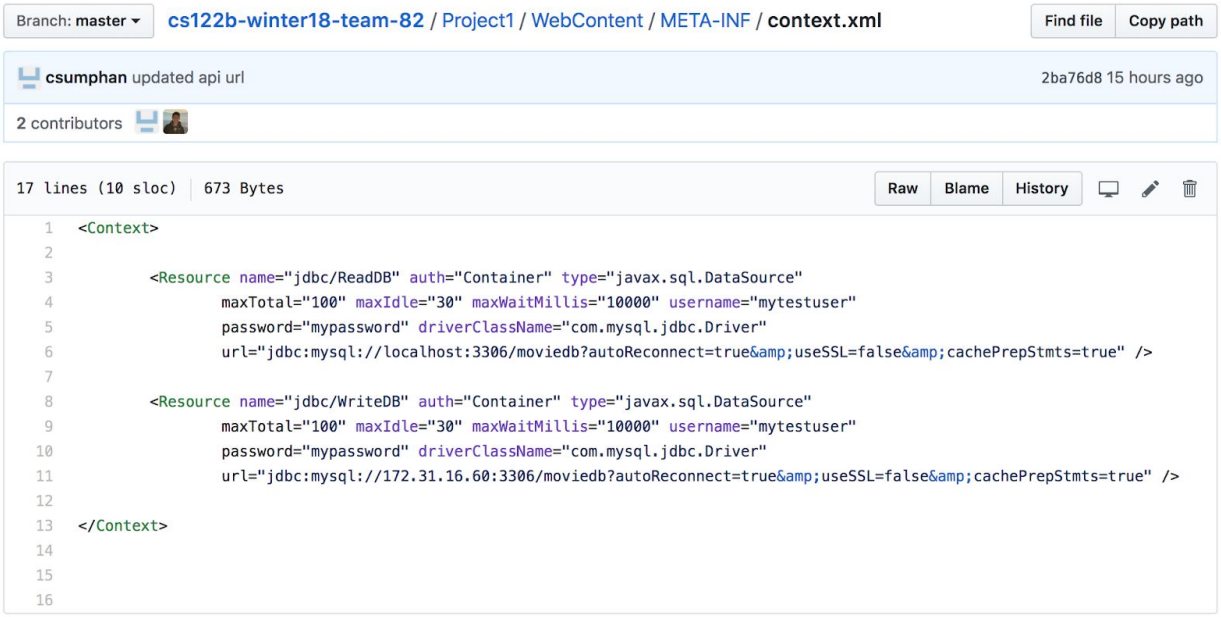
- Load Balancer IP: 35.230.22.63

Yes, I have verified that Fablix is accessible by port 80 with the Google IP address and by port 8080 with the AWS IP address.

Connection Pooling with Two Backend SQL
In both backend tomcat servers, there is a connection setup between the master and the slave database instances. If a request is routed to the slave instance by the load balancer, the slave instance will connect to the master databases if it is a write request, otherwise it will use its own database to satisfy a read request.

The context.xml configuration sets up the connection pool on each server to both a readDB and a writeDB (master). In the configuration, since either master or slave can complete a read request, the mySQL url is set to localhost. If there is a write request, the mySQL url is set to the master mySQL instance.

Lines 1-13 in /Project1/WebContent/META-INF/context.xml



The <resource-ref> tag in the web.xml file links and creates a reference to the two database instances to be used by the servlet.

Line 13-43 on /Project1/WebContent/WEB-INF/web.xml

```
<resource-ref>
  <description>
                    Resource reference to a factory for java.sql.Connection
                    instances that may be used for talking to a particular
                    database that
                    is configured in the server.xml file.
            </description>
  <res-ref-name>
                    jdbc/ReadDB
            </res-ref-name>
  <res-type>
                    javax.sql.DataSource
            </res-type>
  <res-auth>Container</res-auth>
</resource-ref>

  <resource-ref>
  <description>
                    Resource reference to a factory for java.sql.Connection
                    instances that may be used for talking to a particular
                    database that
                    is configured in the server.xml file.
            </description>
  <res-ref-name>
                    jdbc/WriteDB
            </res-ref-name>
  <res-type>
                    javax.sql.DataSource
            </res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

How Read/Write Requests were Routed?

In the image directly above, the references for the data source resources are set, and can then be looked up by the Tomcat servlets when needed. Read and write requests are routed by the purpose of the servlet. If the servlet is just a read request like searching for movies or searching for stars, then the data source is set to “ReadDB” (url is localhost). If the servlet is a write request like addMovies or addStars, then the data source is set to “WriteDB” (url is master IP).

In the Search servlet, the datasource used is ReadDB, since this servlet does not require any updates to the database.

Line 93 - 109 in /Project1/src/Search.java

```
Context initCtx = new InitialContext();
if (initCtx == null)
    out.println("initCtx is NULL");

Context envCtx = (Context) initCtx.lookup("java:comp/env");
if (envCtx == null)
    out.println("envCtx is NULL");

// Look up our data source
DataSource ds = (DataSource) envCtx.lookup("jdbc/ReadDB");

if (ds == null)
    out.println("ds is null.");

Connection dbcon = ds.getConnection();
if (dbcon == null)
    out.println("dbcon is null.");
```

In the AddMovie servlet, the datasource used is WriteDB, since this servlet requires updates to the database.

Line 70 - 86 in /Project1/src/AddMovie.java

```
            Context initCtx = new InitialContext();
if (initCtx == null)
    out.println("initCtx is NULL");

Context envCtx = (Context) initCtx.lookup("java:comp/env");
if (envCtx == null)
    out.println("envCtx is NULL");

// Look up our data source
DataSource ds = (DataSource) envCtx.lookup("jdbc/WriteDB");

if (ds == null)
    out.println("ds is null.");

Connection dbcon = ds.getConnection();
if (dbcon == null)
    out.println("dbcon is null.");

// Declare our statement
Statement addStatement = dbcon.createStatement();
```

Task 3

Yes, we have uploaded all files requested for task 3 in the git repo. The file location is listed below.

Log File - Located in /Project1/ logs

- Each log file is labeled according to each cases outline in 3.3

HTML File - Located at /apache-jmeter-4.0/project5.html

Script File - Located at /Project1/logs/script.py

War File - Located at /Project1.war (on the most outer directory in git repo)

README File - Located at /README.md (on the most outer directory in git repo)