Task 1

- How did you use connection pooling?

  We put our database info inside the context.xml and set maximum collection to 100 and maximum idle to 30. We also add mapping inside web.xml file so that once we need to use JDBC, we can find the source instead of creating a new connection instance every time.

One issue with our project is that after changing the xml file, when we deploy the war file, the login page does not show up first automatically. We need to add login.html in url in order to see the login page

- File name, line numbers as in Github
- cs122b-spring18-team-8/Project5-task1/fabflix/src/NFT_search.java, Line 56-65
- cs122b-spring18-team-8/Project5-task1/fabflix/WebContent/META-INF/context.xml, Line 16-37

- Snapshots showing use in your code

```
56          try {
57              //Class.forName("org.gjt.mm.mysql.Driver");
58  //            Class.forName("com.mysql.jdbc.Driver").newInstance();
59  //
60  //            Connection dbcon = DriverManager.getConnection(loginUrl, loginUser, loginPasswd);
61                Context initCtx = new InitialContext();
62              Context envCtx = (Context) initCtx.lookup("java:comp/env");
63              DataSource ds = (DataSource) envCtx.lookup("jdbc/moviedb");
64              Connection dbcon = ds.getConnection();
65              long startTime=System.nanoTime();
```

```
16          <Resource name="jdbc/write"
17          auth="Container" type="javax.sql.DataSource"
18                  maxTotal="100"
19                  maxIdle="30"
20                  maxWaitMillis="10000"
21                  username="root"
22                  password="team8"
23                  driverClassName="com.mysql.jdbc.Driver"
24                  url="jdbc:mysql://18.188.57.177:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />
25
26          <Resource name="jdbc/moviedb"
27          auth="Container"
28          type="javax.sql.DataSource"
29                  maxTotal="100"
30                  maxIdle="30"
31                  maxWaitMillis="10000"
32                  username="root"
33                  password="team8"
34                  driverClassName="com.mysql.jdbc.Driver"
35                  url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />
36
37  </Context>
```

- How did you use Prepared Statements?

  We use prepared statements on every search page and use ? inside query for inputs from the user. When we get a new search query, we set autocommit to false first and then add the new inputs to prepared statements. After that, we execute the prepared statements and set autocommit back to true.

  - File name, line numbers as in Github

cs122b-spring18-team-8/Project5-task1/fabflix/src/NFT_search.java, Line 69-79, Line 99-114

  - Snapshots showing use in your code

```
69              String input_query="";
70              String [] key_words = key_word.split(" ");
71              for (String a: key_words)
72              {
73                  input_query+='+';
74                  input_query+=a;
75                  input_query+="* ";
76
77              }
78
79              String m_query = "SELECT title FROM movies WHERE (MATCH (title) AGAINST (? IN BOOLEAN MODE)) ";
```

-

```
98
99          String query="SELECT  movies.id, title, year, director,GROUP_CONCAT(DISTINCT ge.name) as ge_n, GROUP_CONCAT(DISTINCT st.name) a
100             "FROM moviedb.movies, moviedb.stars as st, moviedb.stars_in_movies as sim,   " +
101             "moviedb.genres as ge, moviedb.genres_in_movies as gim " +
102             "where movies.id=sim.movieId AND sim.starId=st.id AND gim.movieId=movies.id " +
103             "AND gim.genreId=ge.id AND movies.title in (";
104         String end= ") GROUP BY movies.id, movies.title, movies.year, movies.director;";
105        // SELECT title FROM movies WHERE MATCH (title) AGAINST ('+The* +f*' IN BOOLEAN MODE) )\r\n" +
106         String final_query=query+m_query+end;//fuzzy_query_final+end;
107         System.out.print(final_query);
108         // Perform the query
109         dbcon.setAutoCommit(false);
110         PreparedStatement preparedstatement = dbcon.prepareStatement(final_query);
111         preparedstatement.setString(1, input_query);
112         ResultSet rs = preparedstatement.executeQuery();
113         long endTime2=System.nanoTime();
114         JsonArray jsonArray = new JsonArray();
115
```

Task 2

- Address of AWS and Google instances
  Original IP Public address:18.188.52.177
  Master IP Public address:18.188.140.39
  Slave IP Public address:18.191.128.117
  Google IP Public address: 35.237.114.137
- Have you verified that they are accessible? Does Fablix site get opened both on Google's 80 port and AWS' 8080 port?
  Yes, I have verified all four ip addresses are accessible, Fabflix site get opened both on Google's 80 port and AWS's 8080 port perfectly.

- Explain how connection pooling works with two backend SQL (in your code)?
  We did follow by steps from project instruction, created two connection resources at the begging, one is for master and another one is for slave. For the writing part, we only use master mysql. For the reading part, it would be either one. The purpose of it is let master and slave to be identical.

    - File name, line numbers as in Github

    File name: UCI-Chenli-teaching/cs122b-spring18-team-8/project5 task2/project5/WebContent/META-INF/context.xml
    Line number: 1-13

    - Snapshots

```
<Context path="/fabflix">

        <Resource name="jdbc/write" auth="Container" type="javax.sql.DataSource"
                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
                password="team8" driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://18.188.140.39:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />

        <Resource name="jdbc/moviedb" auth="Container" type="javax.sql.DataSource"
                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
                password="team8" driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />

</Context>
```

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy "balancer://Session_balancer">
    BalancerMember "http://18.191.128.117:8080/Session" route=1
    BalancerMember "http://18.188.140.39:8080/Session" route=2
ProxySet stickysession=ROUTEID
</Proxy>
<Proxy "balancer://TomcatTest_balancer">
    BalancerMember "http://18.191.128.117:8080/TomcatTest/"
    BalancerMember "http://18.188.140.39:8080/TomcatTest/"
</Proxy>
<Proxy "balancer://Fabflix_balancer">
    BalancerMember "http://18.191.128.117:8080/fabflix/" route=1
    BalancerMember "http://18.188.140.39:8080/fabflix/" route=2
ProxySet stickysession=ROUTEID
</Proxy>
        ProxyPass /TomcatTest balancer://TomcatTest_balancer
        ProxyPassReverse /TomcatTest balancer://TomcatTest_balancer
        ProxyPass /Session balancer://Session_balancer
        ProxyPassReverse /Session balancer://Session_balancer
        ProxyPass /fabflix balancer://Fabflix_balancer
        ProxyPassReverse /fabflix balancer://Fabflix_balancer
```

- How read/write requests were routed?
  Every write request would be sent to the master's mysql, and for every read request, it would be sent to the localhost for each instance. For example, add stars, add movies, or check out, such these actions that change data from database are all writing to Master and slave will copy everything that changed on the master.

    - File name, line numbers as in Github

- File name: UCI-Chenli-teaching/cs122b-spring18-team-8/project5 task2/project5/WebContent/META-INF/context.xml
- Line number: 1-13

```
<Context path="/fabflix">

        <Resource name="jdbc/write" auth="Container" type="javax.sql.DataSource"
                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
                password="team8" driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://18.188.140.39:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />

        <Resource name="jdbc/moviedb" auth="Container" type="javax.sql.DataSource"
                maxTotal="100" maxIdle="30" maxWaitMillis="10000" username="root"
                password="team8" driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/moviedb?autoReconnect=true&amp;useSSL=false&amp;cachePrepStmts=true" />

</Context>
```

- File name: UCI-Chenli-teaching/cs122b-spring18-team-8/project5 task2/project5/src/_Addstar.java
- Line number: 49-72
- Snapshots

```
                        Context initCtx = new InitialContext();
                Context envCtx = (Context) initCtx.lookup("java:comp/env");
                DataSource ds = (DataSource) envCtx.lookup("jdbc/write");
                Connection dbcon = ds.getConnection();

                dbcon.close();
                JsonArray jsonArray = new JsonArray();
                if(!name.isEmpty() && name !=null) {
                        String query = "call add_star( ? ,?);";
                        PreparedStatement statement = dbcon.prepareStatement(query);
                        statement.setString(1, name);
                        if(year.isEmpty() || year==null) {
                            statement.setInt(2, -1);
                        }else {
                            statement.setInt(2, Integer.parseInt(year));
                        }

                        ResultSet rs = statement.executeQuery();

                        JsonObject jsonObject = new JsonObject();
                        jsonObject.addProperty("msg", "successfully added new star");
                        jsonArray.add(jsonObject);
                        statement.close();
                        rs.close();
```

-

Task 3

- Have you uploaded the log files to Github? Where is it located?
  Yes, absolutely, it is inside UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/ And there have single and scaled separately, and each case on in it folder with correct name assigned.

For single: UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/single
For scaled: UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/scaled

- Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?
  Yes, absolutely, it is inside UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/html with all 9 cases pictures.

- Have you uploaded the script to Github? Where is it located?

  Yes, it is inside UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/script
  This script is write in python, so you need get the log file and script file in a same folder, in order to get the calculated result.

- Have you uploaded the WAR file and README to Github? Where is it located?
  Yes, war file and README are located at
  UCI-Chenli-teaching/cs122b-spring18-team-8/project5-task3/
  Which is the root of task3 folder, and we upload 3 tasks separately, so project5-task1,project task2, and project-task3 are all located at UCI-Chenli-teaching/cs122b-spring18-team-8/, easy to find.

  Please ignore the Fabflix.zip and Fabflix.apk inside project5-task1, just forget to remove it.