**Task 1**

* How did you use connection pooling?

Servlet: We use Context interface to lookup a DataSource through JNDI, and then get a database connection from the DataSource. JNDI allows to store objects which represent physical connections that can be used by an application in a virtual context.

* File name, line numbers as in Github

|  |  |
| --- | --- |
| File Name | Line Number |
| LoginServlet.java | 43 - 59 |
| MainPageServlet.java | 53 - 69 |
| MovieServlet.java | 73 - 89 |
| movieAutoServlet.java | 65 - 81 |
| SingleMovieServlet.java | 62 - 78 |
| SingleStarServlet.java | 62 - 78 |
| checkOutServlet.java | 44 - 60 |
| resultPageServlet.java | 65 - 69 |
| DashboardLogin.java | 42 - 58 |
| Dashboard.java | 98 - 102 |

* Snapshots showing use in your code

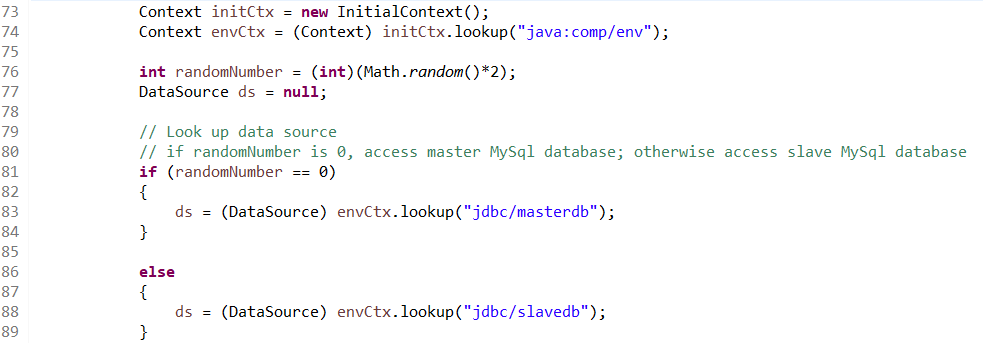
**LoginServlet.java**



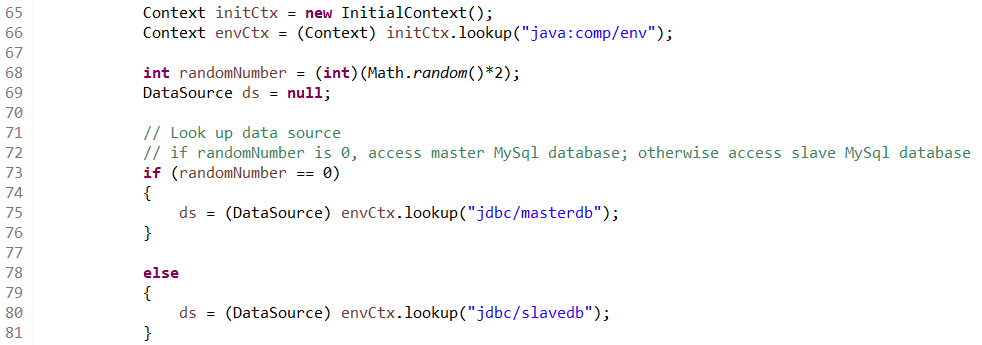
**MainPageServlet.java**



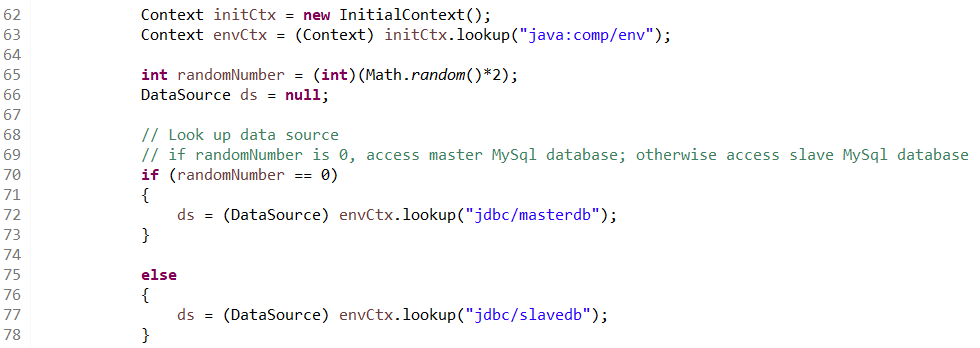
**MovieServlet.java**

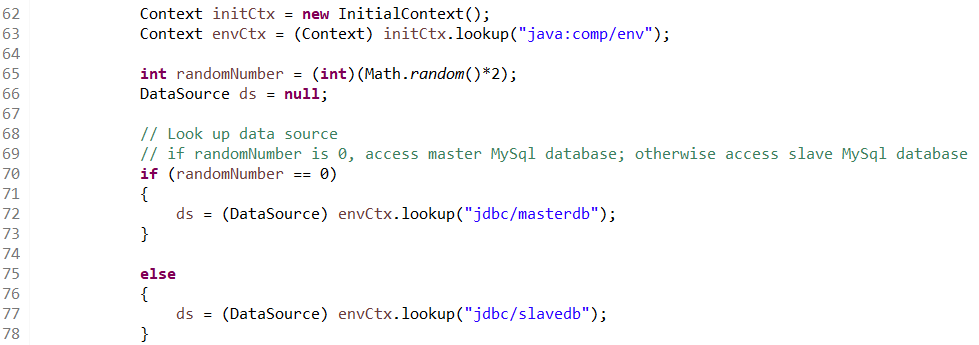


**movieAutoServlet.java**

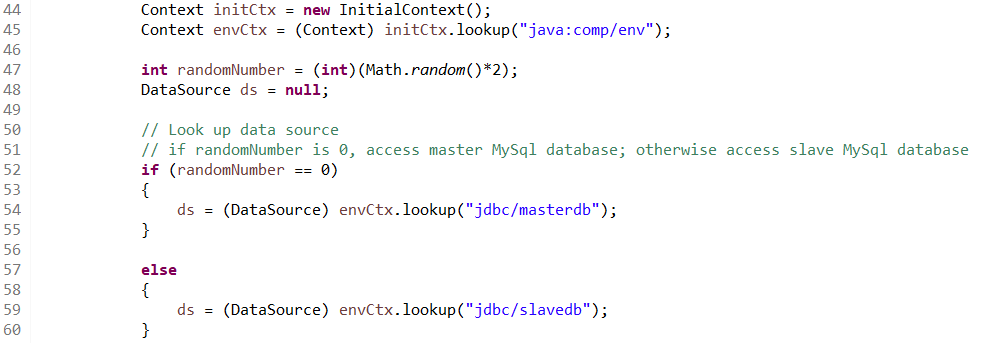


**SingleMovieServlet.java**

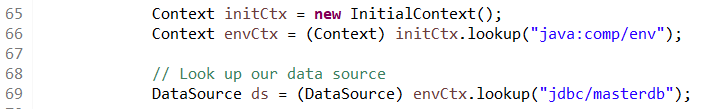


**SingleStarServlet.java** 

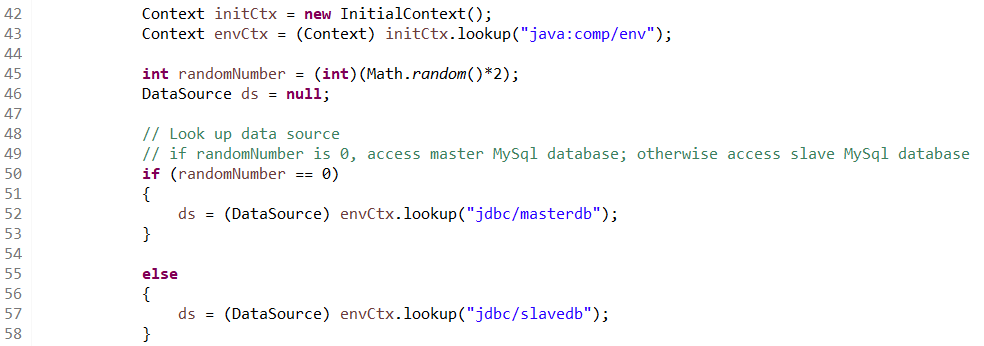
**checkOutServlet.java**



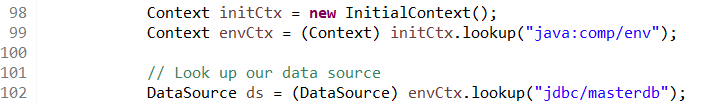
**resultPageServlet.java**



**DashboardLogin.java**



**Dashboard.java**



* How did you use Prepared Statements?

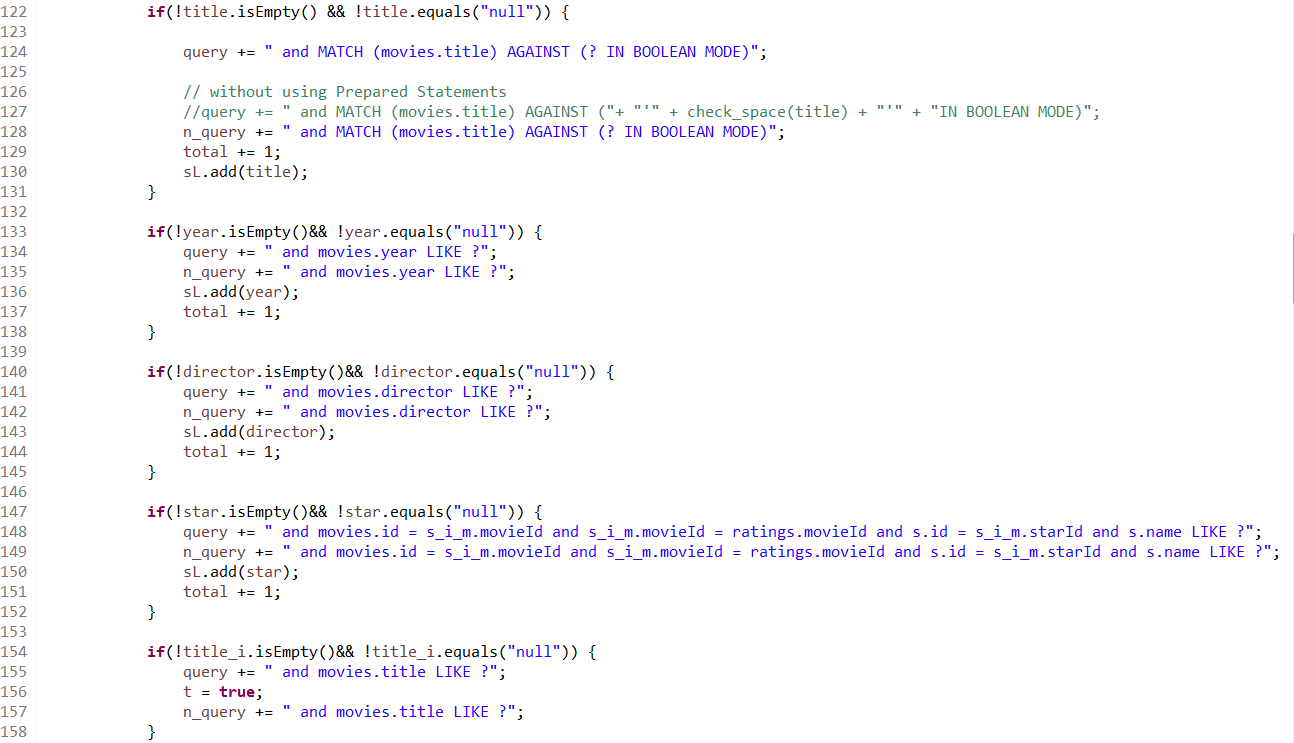
We use Prepared Statements for query the database. Each question mark (?) will be substituted by user input. The search (which is called MovieServlet in our program) servlet has two Prepared Statements, one for movie selection and one for pagination. Prepare statements can improve the query performance when a single session is being used to execute many similar statements since they are indeed cached after their first use.

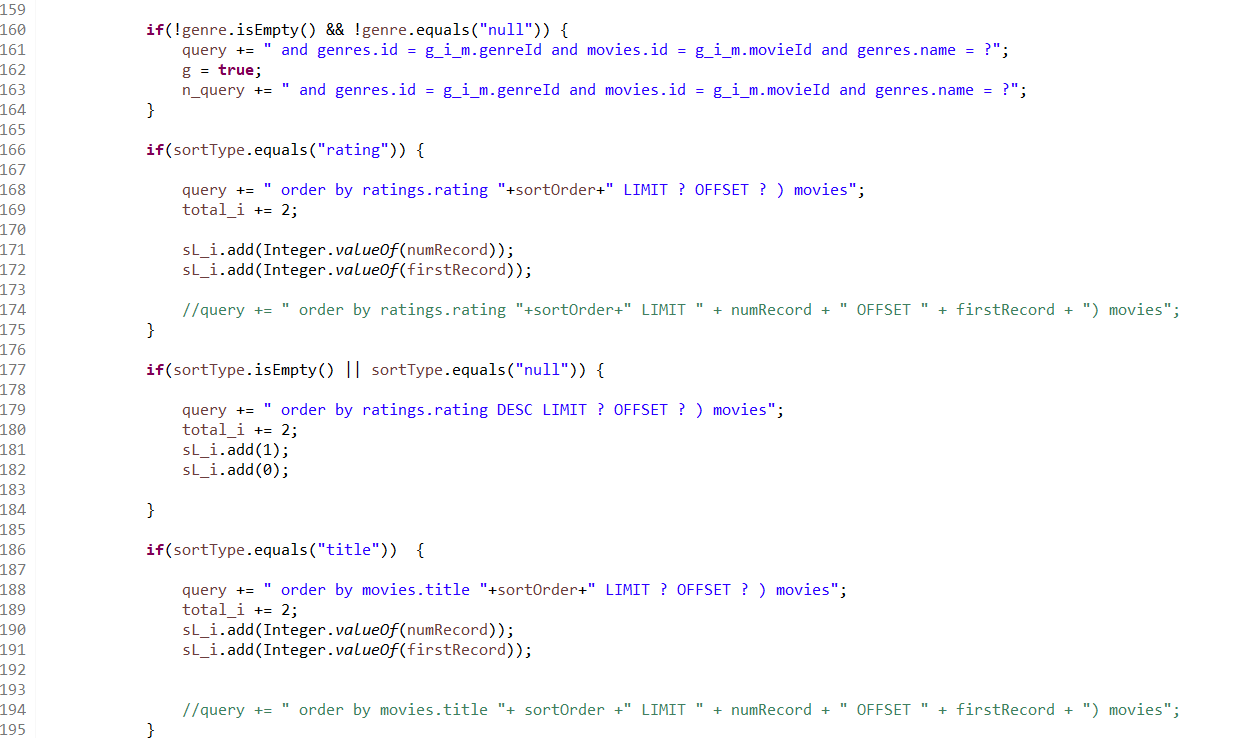
* File name, line numbers as in Github

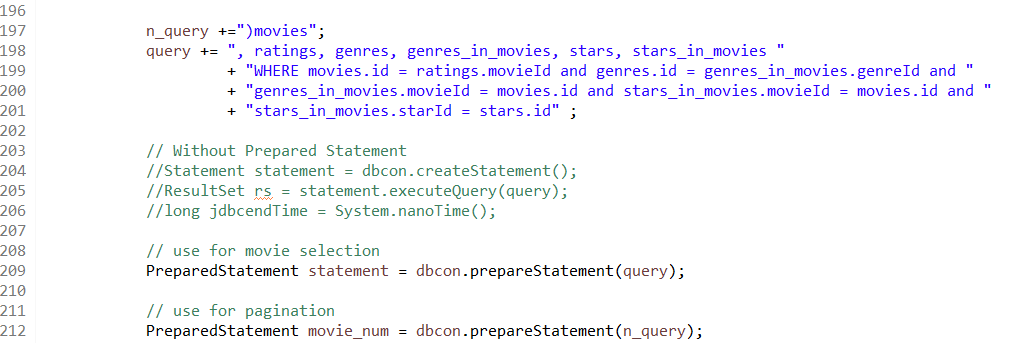
|  |  |
| --- | --- |
| File Name | Line Number |
| MovieServlet.java | 122 - 212 |

* Snapshots showing use in your code

**MovieServlet.java**







**Task 2**

* Address of AWS and Google instances

Google IP: 35.237.60.178

AWS 1 IP: 18.224.110.23

AWS 2 IP: 18.217.212.235

AWS 3 IP: 3.17.191.175

* Have you verified that they are accessible? Does Fablix site get opened both on Google’s 80 port and AWS’ 8080 port?

Yes. All of them are accessible. Fablix site is able to get opened both on Google’s 80 port and AWS’ 8080 port.

* Explain how connection pooling works with two backend SQL (in your code)?

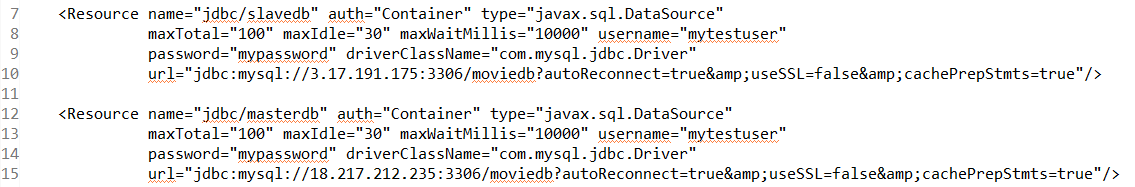
We define two data sources in context.xml. One is master database and one is slave database. The connection pooling for different instances (master/slave) is isolated from each other. Each instance has their own database, thus has their own connection pool. When a connection request is sent to master/slave, the object stored in the pool will return to the application, if the required connection exists in the pool.

* + File name, line numbers as in Github

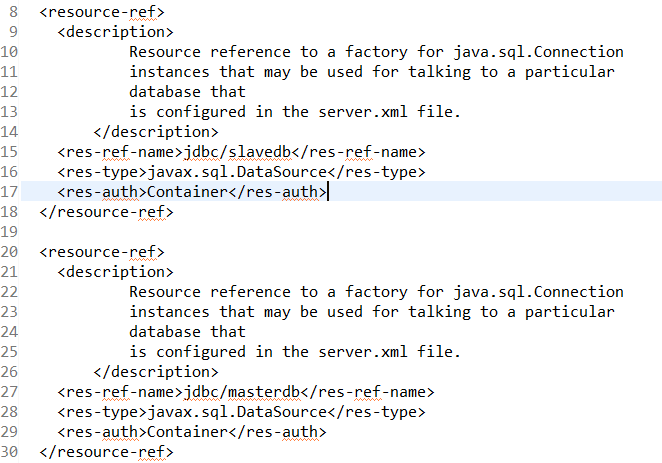
|  |  |
| --- | --- |
| File Name | Line Number |
| context.xml | 7 - 15 |
| web.xml | 8 - 30 |

* + Snapshots

**context.xml**



**web.xml**



* How read/write requests were routed?

We routed the read requests to either of the master/slave database. The write request, however, will only modify and update the master’s database. We only let the servlet which modifies the database accessing masterdb, and the other servlets access either database randomly.

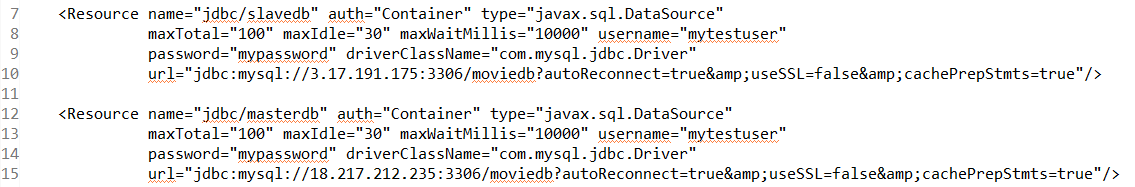
(refer to Task1)

* + File name, line numbers as in Github

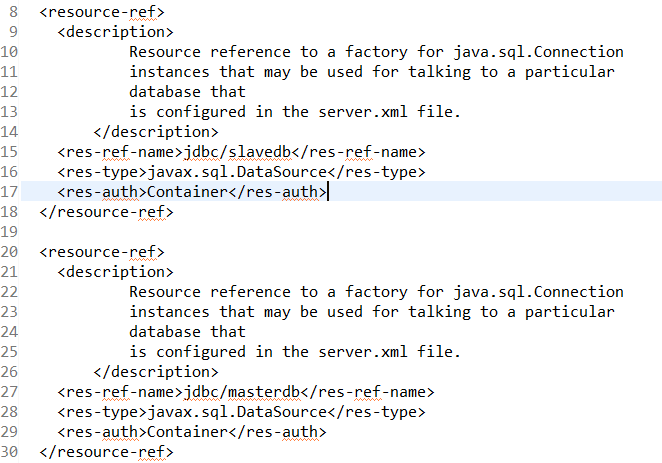
|  |  |
| --- | --- |
| File Name | Line Numbers |
| context.xml | 7 - 15 |
| web.xml | 8 - 30 |
| Dashboard.java | 98 - 102 |
| resultPageServlet.java | 65 - 69 |

* Snapshots

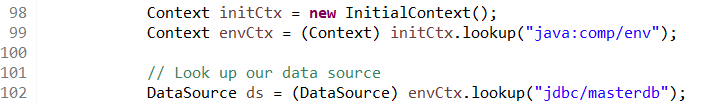
**context.xml**



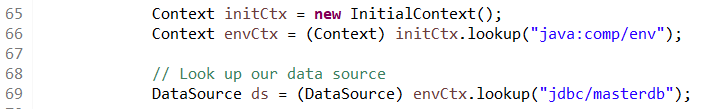
**web.xml**



**DashBoard.java**



**resultPageServlet.java**



**Task 3**

* Have you uploaded the log files to Github? Where is it located?
* Have you uploaded the HTML file (with all sections including analysis, written up) to Github? Where is it located?
* Have you uploaded the script to Github? Where is it located?

* Have you uploaded the WAR file and README to Github? Where is it located?