Homework 4

Michael Galyen

Department of Computer Science, University of Maryland Global Campus

Professor Justin Boswell

February 28, 2022

Table of Contents

Introduction

Please set the String, “log” (line 51), in the ShowAccount.java file to the path to the log.txt file in your project directory. The maximum amount of inactive time allowed for a session is currently set to 30 seconds to be easy to test but please feel free to change that in the Authenticate.java file (line 102) to whatever is easiest for use of the software. The source files for this assignment can be seen running successfully in figures A through C.

Figure A. Image of the home page of the assignment program running in Google Chrome.

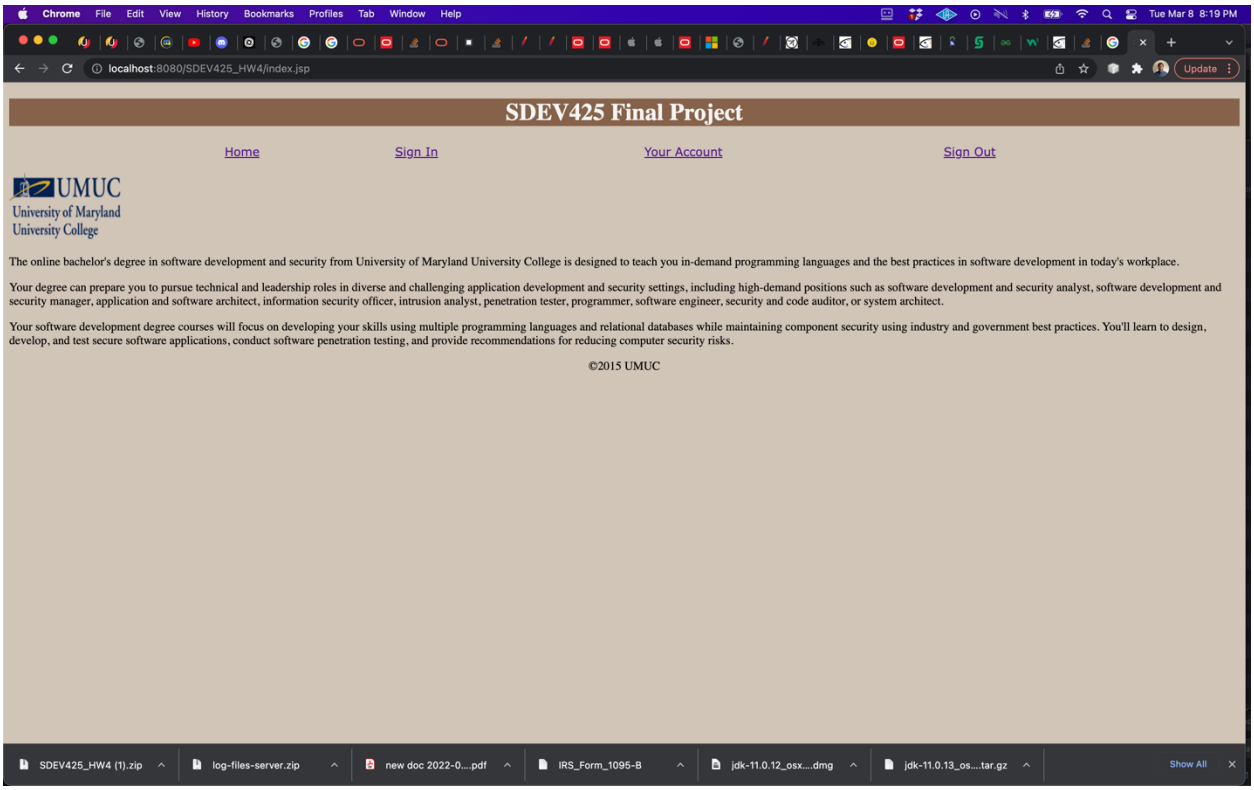


Figure B. Image of the assignment program after successfully logging in.

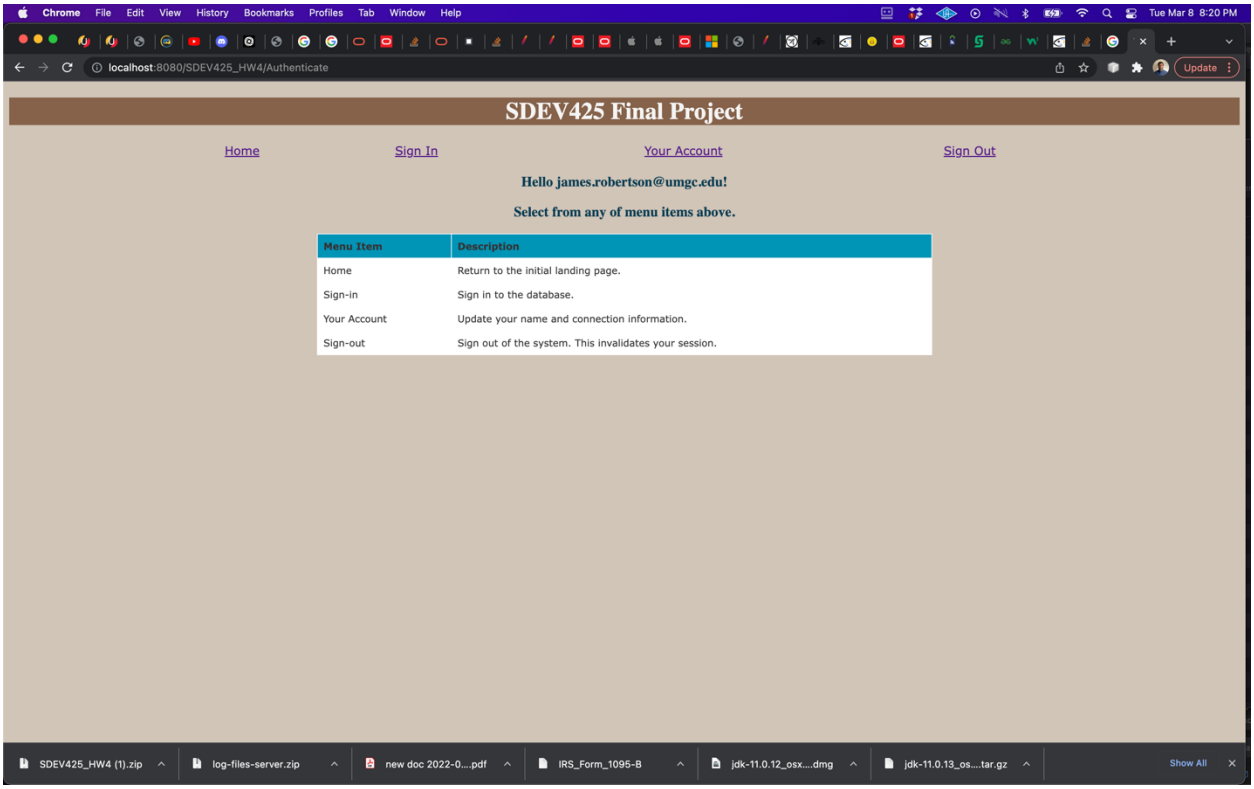
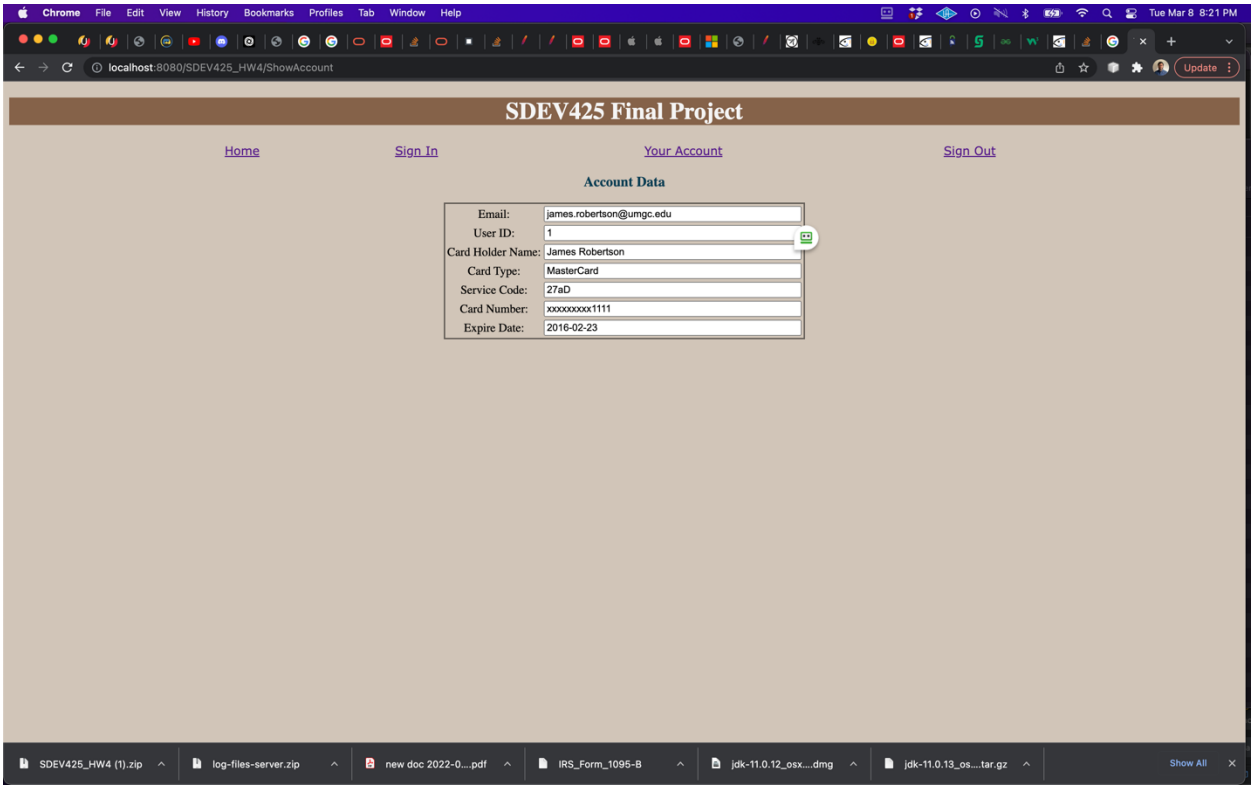


Figure C. Image of the assignment program after logging in and accessing the account information page.



PCI DSS 2.1

"Always change ALL vendor-supplied defaults and remove or disable unnecessary default accounts before installing a system on the network. This includes wireless devices that are connected to the cardholder data environment or are used to transmit cardholder data (PCI Standards Council, 2015)." To implement this requirement, both the test.admin@umgc.edu account and the test.customer@umgc.edu account were dropped from the SDEV_USERS table as seen in figures 1 through 4. Instead of just changing the passwords on these default accounts, deleting the accounts entirely achieves an additional goal of taking away those

default usernames that any hacker familiar with this system would know existed, making the system less susceptible to dictionary attacks aimed at the default usernames.

Figure 1. Image of the SQL commands used to drop the two default test accounts from the system.

```
11    DELETE FROM SDEV425.SDEV_USERS WHERE USER_ID = 2;
12    DELETE FROM SDEV425.SDEV_USERS WHERE USER_ID = 3;
```

Figure 2. Image of the SQL history showing that the commands were run.

SELECT * FROM SDEV425.USER_INFO FETCH FIRST 100 ROWS ONLY	3/7/22, 8:07 PM
SELECT * FROM SDEV425.SDEV_USERS FETCH FIRST 100 ROWS ONLY	3/7/22, 8:07 PM
DELETE FROM SDEV425.SDEV_USERS WHERE USER_ID = 3	3/7/22, 8:07 PM
DELETE FROM SDEV425.SDEV_USERS WHERE USER_ID = 2	3/7/22, 8:07 PM

Figure 3. Image of the SDEV_USERS table after dropping the two default test accounts.

Max. rows: 100 Fetched Rows: 1 Matching Rows:

#	USER_ID	EMAIL	FIRSTNAME	LASTNAME	CITY
1	1	james.robertson@umgc.edu	Jim	Robertson	Adelphi

Figure 4. Image of the USER_INFO table after only adding the user info for the one real user.

SELECT * FROM SDEV425.USE... ×

Max. rows: 100 Fetched Rows: 1

#	USER_ID	PASSWORD
1	1	mypassword

PCI DSS 3.2

“Do not store sensitive authentication data after authorization (even if it is encrypted). See table below. Render all sensitive authentication data unrecoverable upon completion of the

authorization process. Issuers and related entities may store sensitive authentication data if there is a business justification, and the data is stored securely (PCI Standards Council, 2015).” To satisfy this requirement, the CAV_CCV2, FULLTRACKDATA, and PIN columns of the CUSTOMERACCOUNT table were dropped so that information would not be permanently retained. This was accomplished using SQL statements that were executed against the SDEV425 database and they can be seen below in Figures 5 – 7. When sensitive information such as PIN numbers, full track data, or CAV_CC2 data are stored, it creates a much more dangerous situation if a user’s data is either exfiltrated or recovered by attackers after a leak. With this much information, an attacker would have enough data to make fraudulent charges easily which also makes stores of this type of data more desirable as a target.

Figure 5. Image of the SQL commands that will delete the rows that should not be stored from the CUSTOMERACCOUNT table.

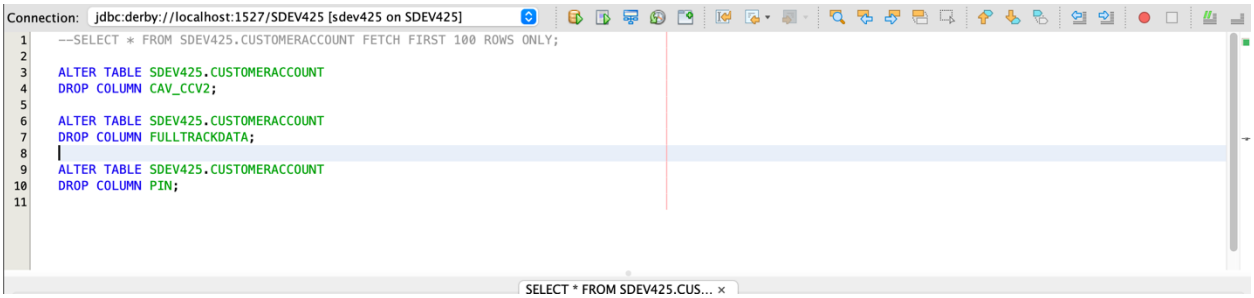


Figure 6. Image of the SQL history showing that the commands were run.

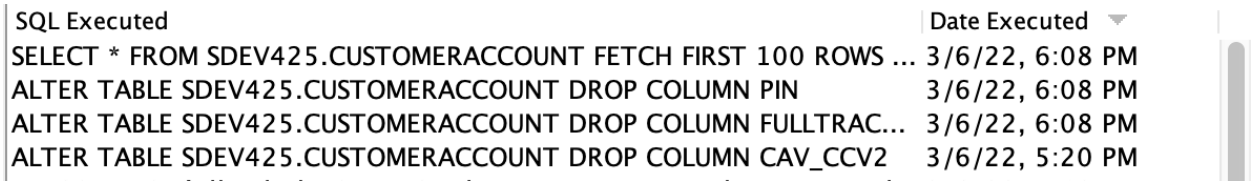
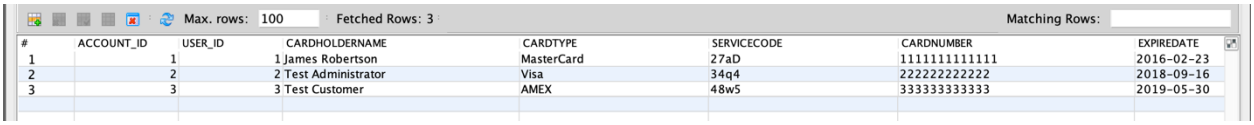


Figure 7. Image of the CUSTOMERACCOUNT table after running the above SQL commands. Note the absence of the CAV_CCV2, FULLTRACKDATA, and PIN columns.



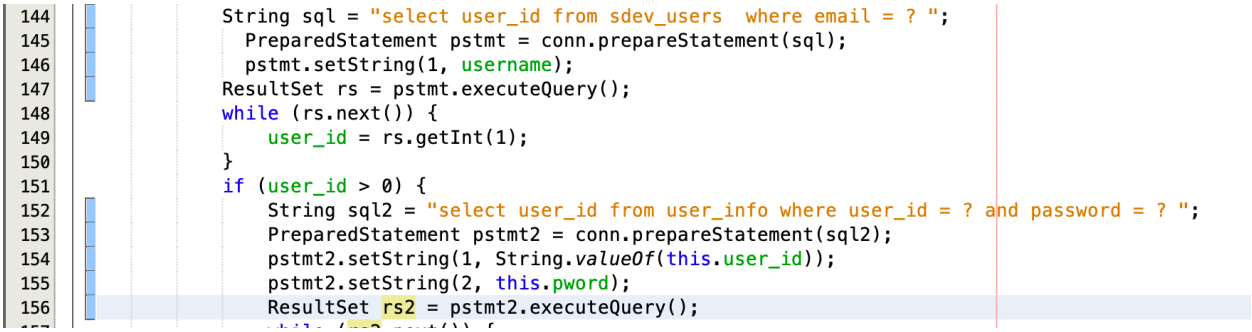
Max. rows: 100 Fetched Rows: 3 Matching Rows:

#	ACCOUNT_ID	USER_ID	CARDHOLDERNAME	CARDTYPE	SERVICECODE	CARDNUMBER	EXPIREDATE
1	1	1	James Robertson	MasterCard	27aD	1111111111111	2016-02-23
2	2	2	Test Administrator	Visa	34q4	222222222222	2018-09-16
3	3	3	Test Customer	AMEX	48w5	333333333333	2019-05-30

PCI DSS 6.6

“Ensure all public-facing web applications are protected against known attacks, either by performing application vulnerability assessment at least annually and after any changes, or by installing an automated technical solution that detects and prevents web-based attacks (for example, a web-application firewall) in front of public-facing web applications, to continually check all traffic (PCI Standards Council, 2015).” To satisfy this requirement, the source code was first inspected for common weaknesses. This resulted in the discovery of two instances of a SQL string being built with unvalidated user input in the Authenticate.java file which can lead to SQL injection. SQL injections can result in many negative consequences. Among them are data manipulation, denial of service, and an attacker taking control of a database. This was corrected by using prepared statements with the help of the java.sql.PreparedStatement library as can be seen in figure 8. Prepared statements allow user input to be parameterized such that the user input cannot be mistaken for SQL commands if malicious data is input.

Figure 8. Image of the two strings of SQL commands in Authenticate.java that were parameterized using prepared statements.



```
144             String sql = "select user_id from sdev_users  where email = ? ";
145               PreparedStatement pstmt = conn.prepareStatement(sql);
146               pstmt.setString(1, username);
147             ResultSet rs = pstmt.executeQuery();
148             while (rs.next()) {
149                 user_id = rs.getInt(1);
150             }
151             if (user_id > 0) {
152                 String sql2 = "select user_id from user_info where user_id = ? and password = ? ";
153                 PreparedStatement pstmt2 = conn.prepareStatement(sql2);
154                 pstmt2.setString(1, String.valueOf(this.user_id));
155                 pstmt2.setString(2, this.pword);
156                 ResultSet rs2 = pstmt2.executeQuery();
```

PCI DSS 3.3

"Mask PAN when displayed (the first six and last four digits are the maximum number of digits you may display), so that only authorized people with a legitimate business need can see the full PAN. This does not supersede stricter requirements that may be in place for displays of cardholder data, such as on a point-of-sale receipt (PCI Standards Council, 2015)." In accordance with PCI DSS requirement 3.3, when an account holder requests to view their account information on the web page, the account number appears masked with only the last 4 digits displayed. The last 4 digits are preceded by an "x" in place of each number in the credit card number as shown in figure 9. This is accomplished by the addition of a method called maskCC() to the ShowAccount.java file shown in figure 10. The next step would be to mask the data in the database itself so that it's protected even from employees with admin access to the database. In that scenario, the method that was used to mask the data on output in this program could be done away with.

Figure 9. Image of account data being displayed with the credit card number masked.

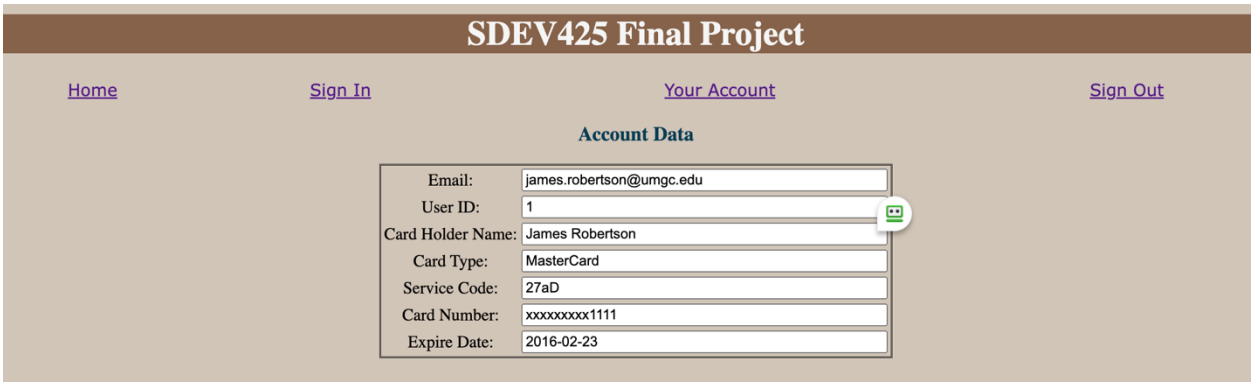


Figure 10. Image of the maskCC() method in the Authenticate.java file that masks credit card numbers before they are served to the web page.

```
    protected void maskCC(String CC) {
45      StringBuilder base = new StringBuilder(15);
46
47
48      String[] tempList = CC.split("");
49      for (int i = 0; i < tempList.length; i++) {
            if (i > 8) {
51              base.append(tempList[i]);
52          }
53          else {
54              base.append("x");
55          }
56      }
57      maskedCC = base.toString();
58  }
```

PCI DSS 10.1

"Implement audit trails to link all access to system components to each individual user (PCI DSS Council)." This was accomplished by adding a logging method called eventLogger() to the ShowAccount.java file. The eventLogger method logs system and session data to a local file

called log.txt every time account data is accessed, or an attempt is made to access the data. The data collected includes username if a user is logged in, location in the file and the file that created the log message, the date, the time, the session creation time, and the epoch time of the log message creation. The eventLogger() method can be seen in figure 11 and an image of the log file after being populated can be seen in figure 12. Without proper logging, audits cannot be conducted effectively, and many security incidents will go undetected. To be realistically useful in an audit, log files need to have information that links relevant system components to users that interact with them and include timestamps and session information.

Figure 11. Image of the eventLogger() method in the ShowAccount.java file.

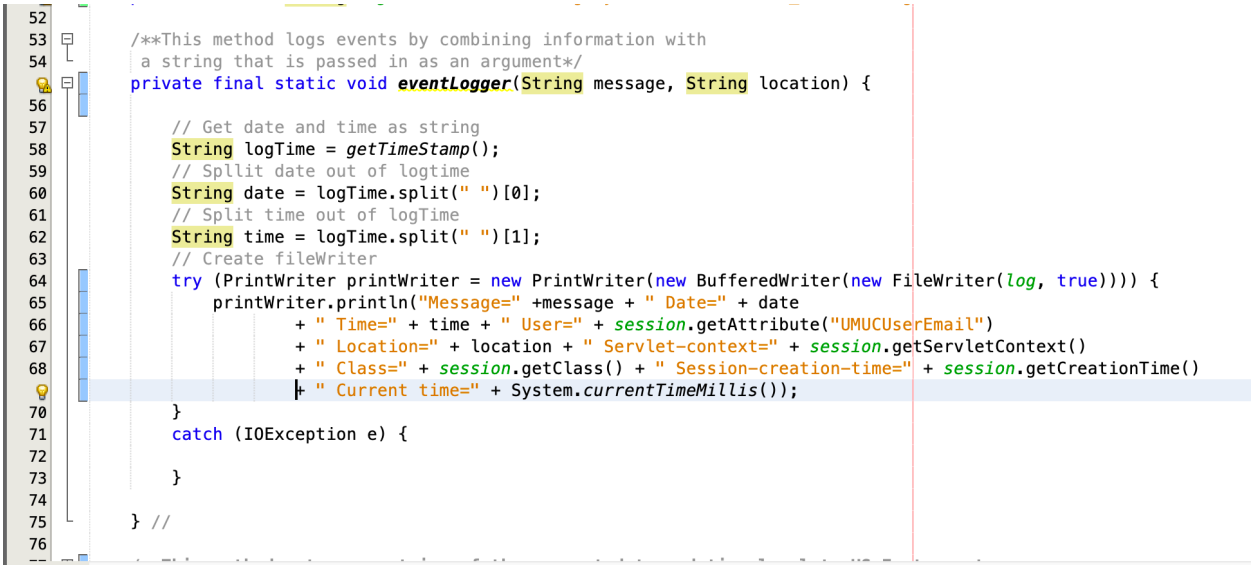
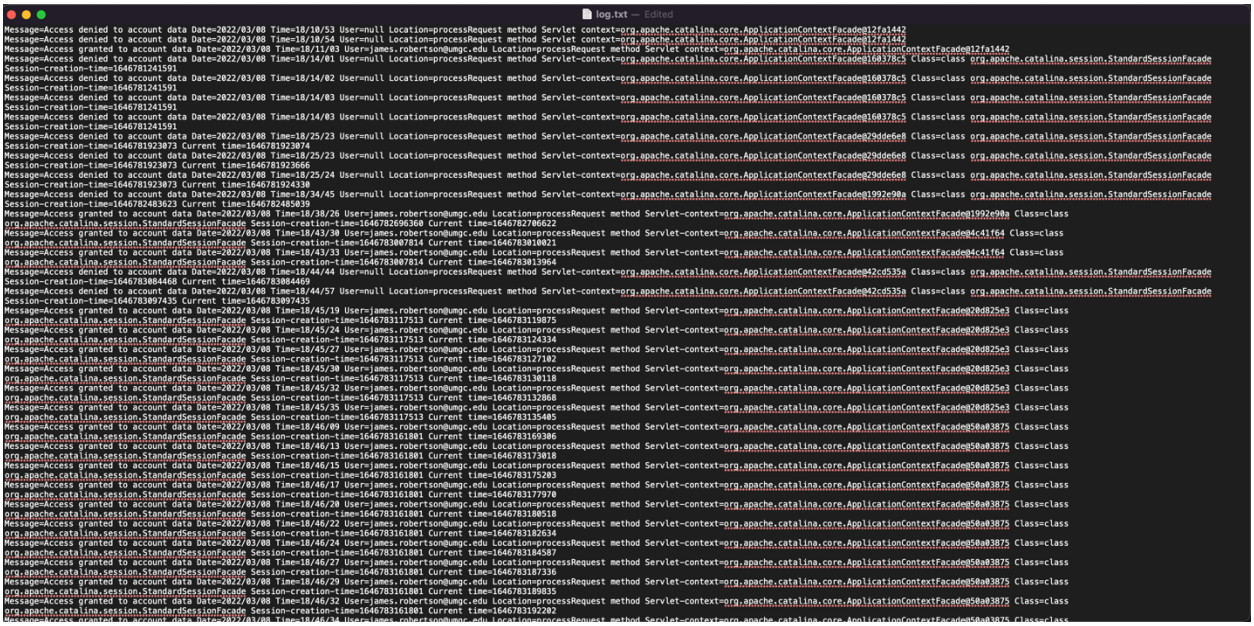


Figure 12. Image of the populated log.txt file.



PCI DSS 6.3

"Develop internal and external software applications including web-based administrative access to applications in accordance with PCI DSS and based on industry best practices. Incorporate information security throughout the software development life cycle. This applies to all software developed internally as well as bespoke or custom software developed by a third party (PCI DSS Council, 2015)." One of the best practices that was implemented in this software was the use of a session time out after inactivity. In the Authenticate.java file, the line pictured in figure 13, sets the maximum value in seconds for inactivity before the session becomes inactive. This is a best practice because users can easily leave a session running with sensitive information on the screen which could then be seen by anyone that can see the computer screen thus exposing the data.

Figure 13. Image of the line of code that sets the max value for inactive time in Authenticate.java.

References

PCI Standards Council. (2015). PCI DSS Quick Reference. Retrieved March 6, 2022, from https://learn.umgc.edu/d2l/le/content/628222/viewContent/24884138/View