

Professor Carl Eyler

SDEV425 6381: Mitigating Software Vulnerabilities

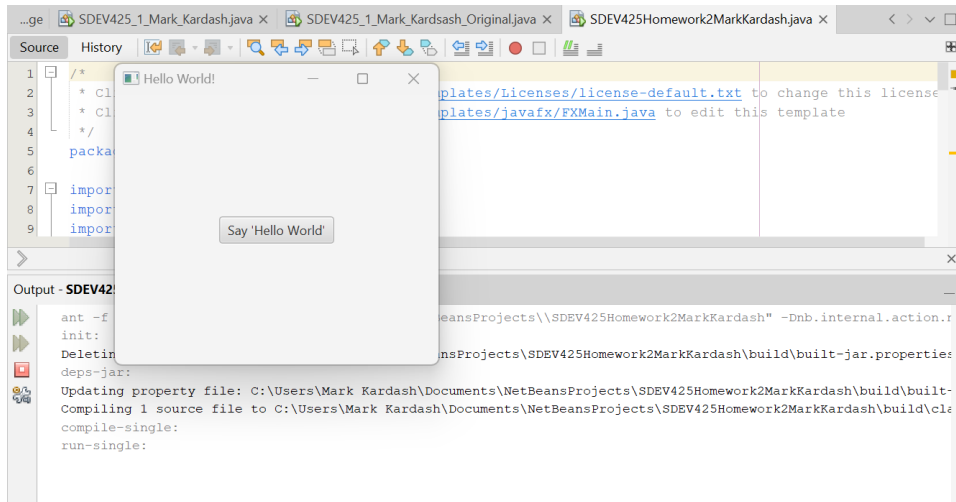
Homework 2

27 January 2024

University of Maryland Global Campus

This document contains the written (typed) parts of SDEV425 Homework 2. It will start with providing screenshots for confirmation that both the “Hello, World!” and “Login” JavaFX apps function correctly.

Below is a screenshot of the “Hello, World!” application running correctly.



For whatever reason, the app only runs correctly if I right-click on the source file (SDEV425Homework2MarkKardash.java), and click “Run File”. Simply clicking on the green arrow actually gives me a “BUILD FAILED” error, as shown below:

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/javafx/FXMain.java to edit this template
4   */
5   package sdev425homework2markkardash;
6
7   import javafx.application.Application;
8   import javafx.event.ActionEvent;
9   import javafx.event.EventHandler;

```

```

Output - SDEV425Homework2MarkKardash (jfxs-run)
- do-jar-delete-manifest:
- do-jar-without-libraries:
- do-jar-with-libraries:
- jfx-copylibs-warning:
- jfx-copylibs:
Java 15 has removed Nashorn, you must provide an engine for running JavaScript yourself. GraalVM JavaScript curre
BUILD FAILED
E:\Users\Mark_Kardash\Documents\NetBeansProjects\SDEV425Homework2MarkKardash\nbproject\jfx-impl.xml:1251: The fol
C:\Users\Mark_Kardash\Documents\NetBeansProjects\SDEV425Homework2MarkKardash\nbproject\jfx-impl.xml:1259: Unable
Total time: 1 second

```

I am unsure why, but it says I would need to install a JavaScript running engine. However, since I was able to run the file through the Run File command, I consider this a success.

As for the Login app, when I clicked on the green arrow, it said, weirdly enough, that the build was successful, yet gave me the same message about Java 15 having removed Nashorn:

```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package sdev425loginappkardash;
7
8   import javafx.application.Application;
9   import javafx.event.ActionEvent;
10  import javafx.event.EventHandler;

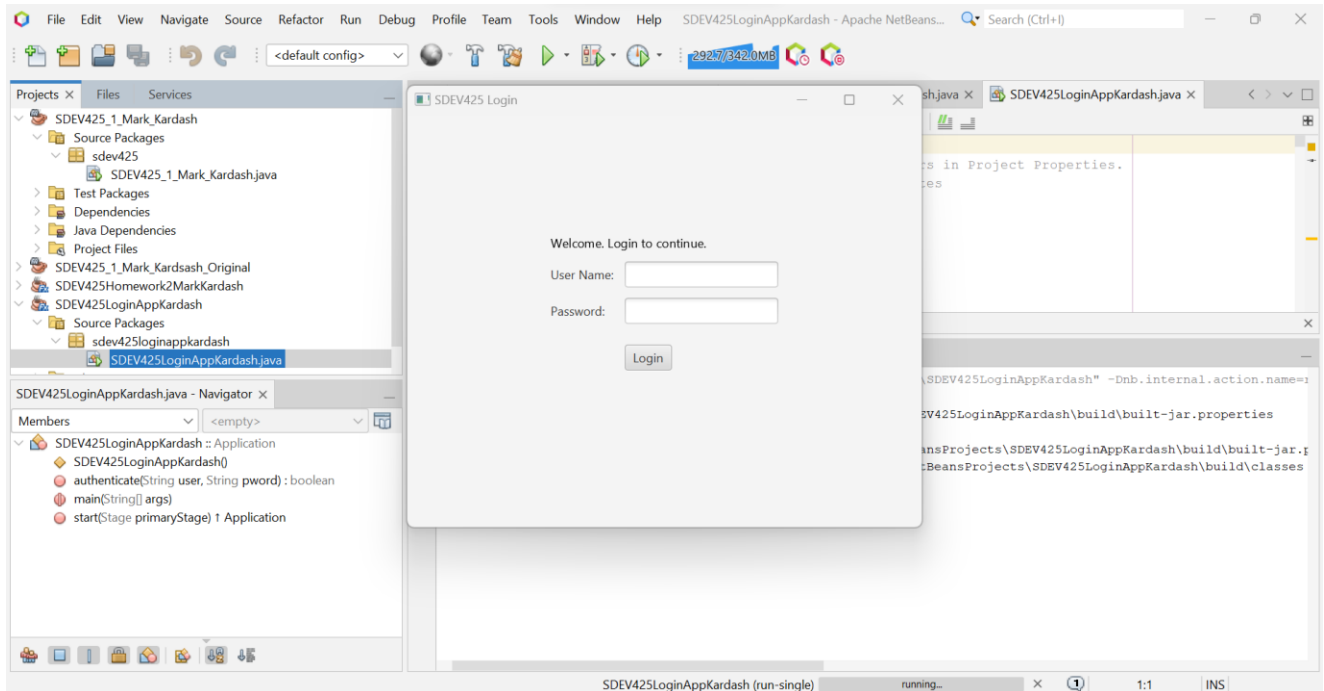
```

```

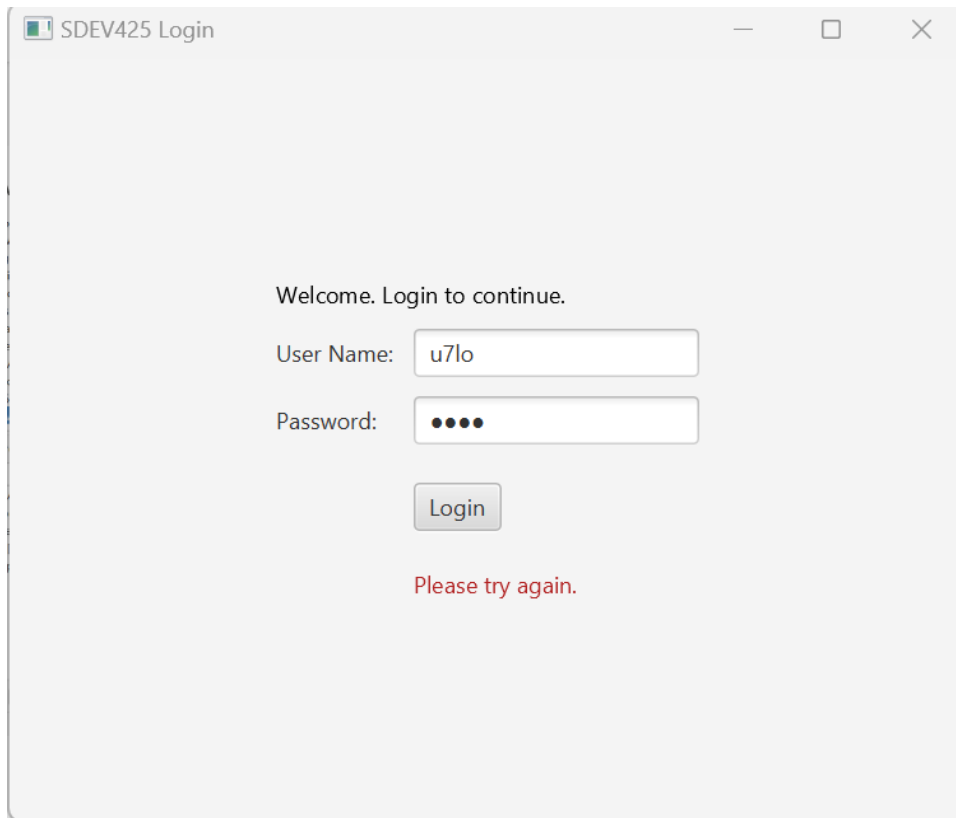
Output - SDEV425LoginAppKardash (jfxs-run)
-init-macrodef-copylibs:
- do-jar-copylibs:
- do-jar-delete-manifest:
- do-jar-without-libraries:
- do-jar-with-libraries:
- jfx-copylibs-warning:
- jfx-copylibs:
Java 15 has removed Nashorn, you must provide an engine for running JavaScript yourself. GraalVM JavaScript curre
Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)

```

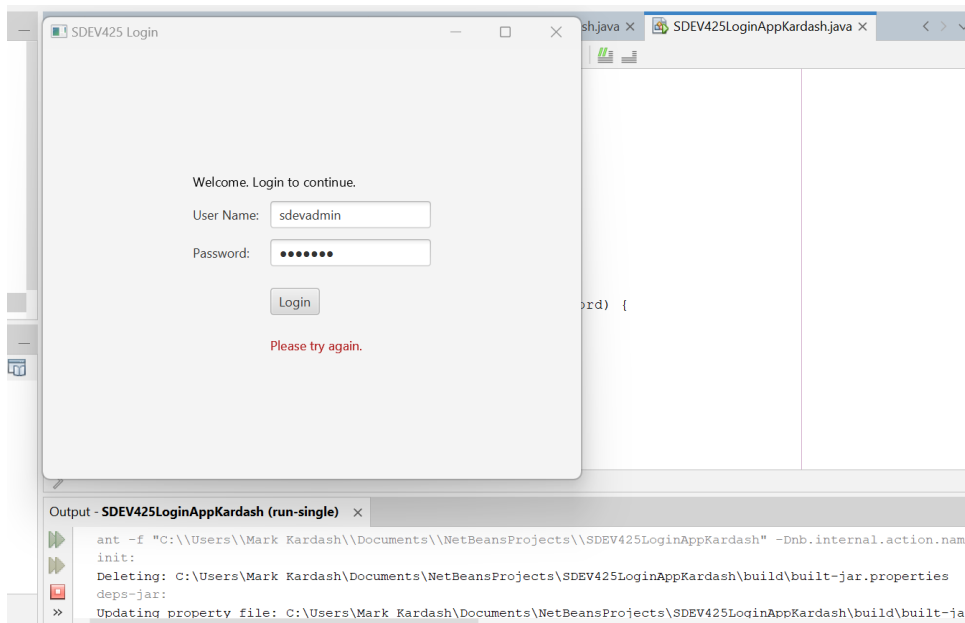
However, right-clicking on the source file and selecting “Run File” again helped the app run correctly:



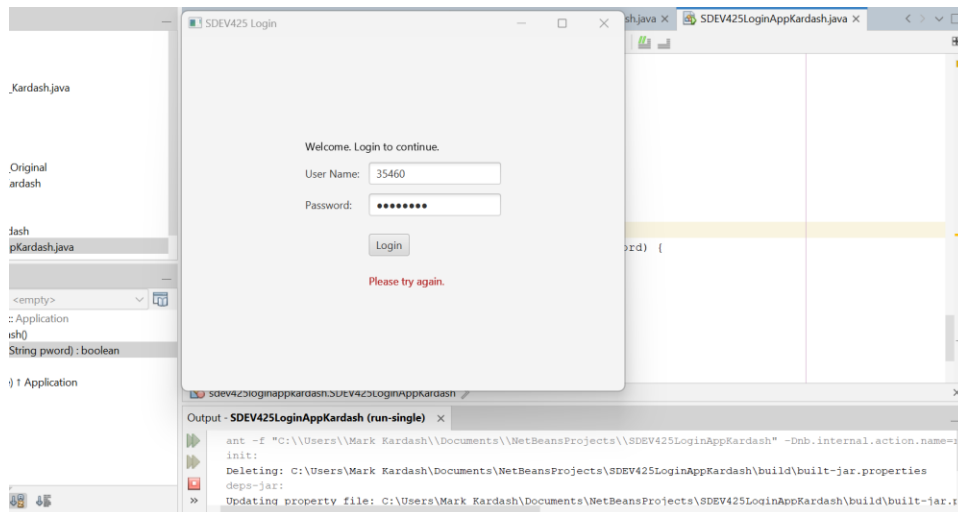
As expected, putting in incorrect credentials will leave the user with a “Please Try Again” error:



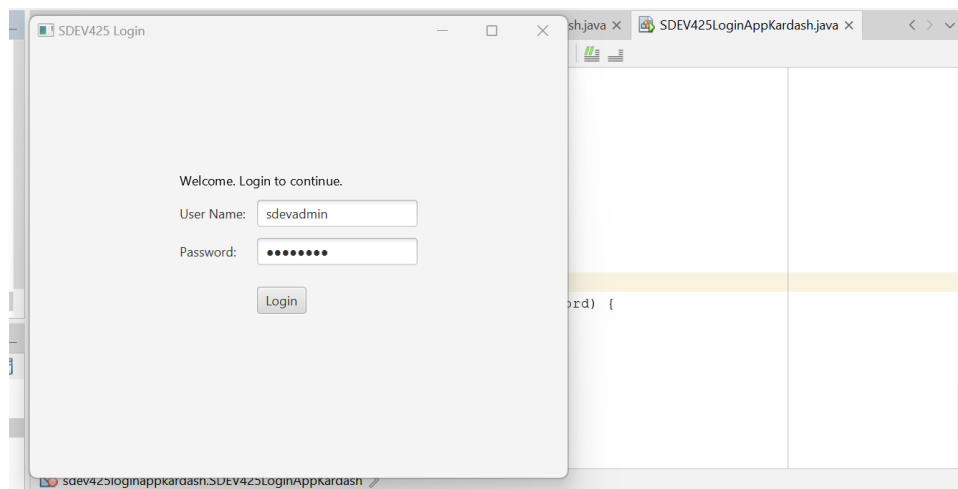
The same will happen if only the password is incorrect:

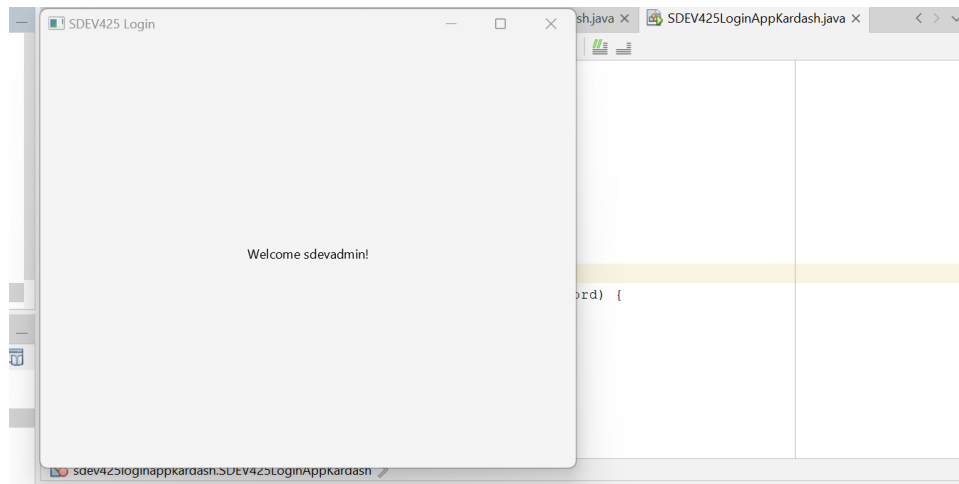


And the same will also happen with an incorrect username, but correct password:



However, with all the correct credentials, the app allows us to properly log in, displaying the Welcome message:





Summary of Login Application Functionality:

After examining the Login application given to me, I have reached the conclusion that, although large in terms of code, is quite simple in functionality. Its build begins with the standard import of a package, which, in this case, is named “sdev425loginappkardash”. Then, several commands are imported into the app: The first one is `application.Application`, which basically allows the main class to extend to the entire application, covering all of its functions, and allowing it to run correctly. This is followed by the import of a series of other functions, each of which will be explained in more detail below.

The actual code of the application begins with setting up the class, `SDEV425LoginAppKardash`. It is the main class, and, to make it function correctly, the phrase “extends `Application`” is added. This way, the developer ensures that the class applies to the entire program, instead of just covering the first action directly underneath it. Within the class, a `Start` method is included. Its purpose is to setup a new stage, or main window, where all the details of the upcoming program will be displayed. This new stage is named `PrimaryStage`. Now, the “`.setTitle`” function is used to put in the main “SDEV425 Login” title, which appears at the top of the window. The `GridPane` function, of which a new instance, “`grid`” is created, allows the designer to place the

remaining details wherever they want to, splitting the stage into rows and columns, like a grid.

The “setAlignment” and “pos” (position) functions are responsible for WHERE on the grid items are placed. Thus, the command “grid.setAlignment.pos(CENTER)” ensures that any fields (in this case, “User Name” and “Password” fields are at the center of the stage. As for the “grid.setHgap(10)” and “grid.setVGap(10)” commands, they allow the fields to have an equal horizontal and vertical distance (“gap”) from the sides of the main window (stage), and from each other. Next, the scene, which is the slightly smaller window inside the stage, is set up. We use a new instance of the Text function, named “sceneTitle”, to create the title above the “User Name” and “Password” fields: “Welcome. Login to continue.” The command “grid.add(sceneTitle, 0, 0, 2, 1)” ensures that the grid is split into 2 columns and 1 row, which are going to be the fields for the credentials. The names of the fields are created using the “Label” function, with two labels set up, one named “userName”, and the other “Password”. Under each of them, a text field is added, titled “userTextField”, for users to actually enter their credentials. We then add the labels to the grid using the “grid.add()” command. With the “Password” field, however, things get slightly more complicated, as we have to use a separate function called PasswordField, to make sure the contents of the field cannot be read with the naked eye. Once this is done, the field, entitled “pwBox”, is added to the grid (“grid.add(pwBox, 1, 2)”). Using the Button function, we also create the “Login” button necessary for the app, and add it to the grid. The Text instance “actiontarget”, which comes next, is meant to establish the text that the user enters, as the target of any “action” (such as the pressing of the “Login” button) that the user may take in the app.

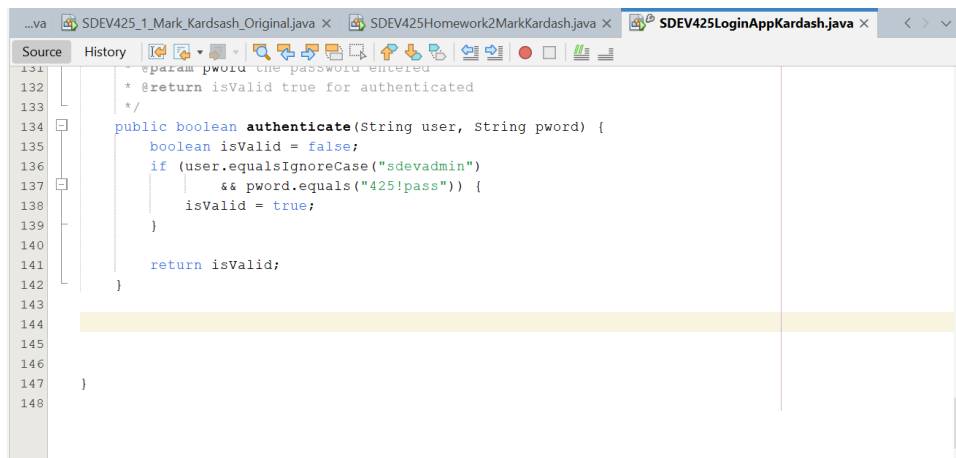
From here, one of the most crucial phases of the program’s working process begins. The command “btn.setOnAction(new EventHandler<ActionEvent>())” sets the “Login” button to

action, which means evaluating the user's credentials each time the button is clicked, and displaying the appropriate message. To do this, a new "Action Event" (the process in which the app will be analyzing the credentials) is started. It is done via an "Event Handler", a method that directs the course of the action. In the Event Handler, an "if" loop presents two possibilities using the Boolean variable "isValid". If the credentials are correct, and the variable is true, the program sets up a new "GridPane" function called "grid2". It aligns the text to the center via "setAlignment", sets the appropriate horizontal and vertical distances via "setHgap" and "setVgap", creates a new instance of Text called "scenetitle" where the welcome message is printed, sets the correct dimensions for the scene, puts the stage (outer window) and scene (inner window) together via "primaryStage.setScene(scene);" command, and finally, executes the ".show" command to make the entire window visible. On the other hand, if the credentials are invalid, the program adds the actiontarget (the "Login" button) to the grid, sets the grids dimensions, as well as the color of the "Please Try Again" message to be displayed via ".setFill()" command. It then writes the actual text of the error message via ".setText" command. Finally, at the bottom of the code the dimensions for the scene are set. That is, the size of the smaller window within the main window. The main method, called "main", simply collectively initiated the program via "launch(args)" command.

Lastly, in the Boolean function "authenticate", the "isValid" variable is set to "false" by default. This is done so that it can later be set to "true" if the credentials are valid. The valid credentials are included in another "if" loop, and a "return" statement returns the variable "isValid". If the credentials entered by the user are correct, the "isValid" variable is true, and, when returned, will give the user a welcome message. If the user enters incorrect credentials, the "isValid" variable is false, and will therefore return an error message.

Implementing Security Controls on SDEV425 Login Application:

Now that the Login app has been properly described, it is time to implement some security controls upon it. To get it out of the way, we start with the easiest one, which is adding a timestamp to the application. In this way, we would follow the AU-8: Timestamps security control. According to the website CSF Tools, supported by NIST, the programmer must use “internal system clocks to generate time stamps for audit records” (“AU-8: Time Stamps”, para.1). In our case, we must build a function that would create and display timestamps for our app. Moreover, this will be a function that displays the timestamp in minutes. The details of its creation are shown below.



```

131  * @param pword the password entered
132  * @return isValid true for authenticated
133  */
134  public boolean authenticate(String user, String pword) {
135      boolean isValid = false;
136      if (user.equalsIgnoreCase("sdevadmin")
137          && pword.equals("425!pass")) {
138          isValid = true;
139      }
140
141      return isValid;
142  }
143
144
145
146
147
148

```

Figure 1: Ending of original code without any modifications

We will add our new function at the end of the big “if” loop.

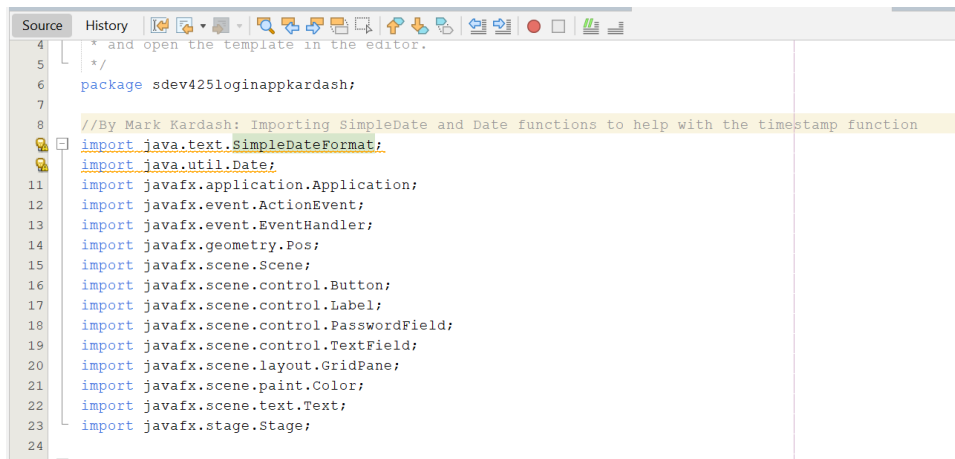


Figure 2: Importing new functions to use within the Timestamp function

In the figure above, we have imported the Date and SimpleDateFormat functions to use for the timestamp. The former will make the date appear, while the latter will get the date into the desired “YY/MM/DD” format. After this, we will convert the entire date into minutes to get the actual timestamp.

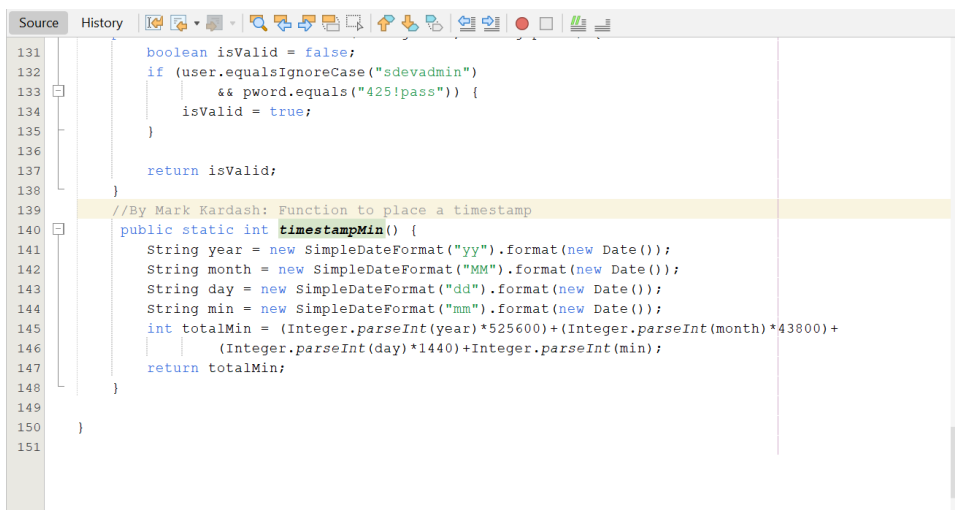


Figure 3: Adding timestamp function

As can be seen from the above screenshot, the function “timestampMins” has been added to the code. It works by taking in the current date, in “YY/MM/DD” format, converting all of it into

minutes (by multiplying by the number of minutes in each), and returning it all as a single timestamp. A slightly more detailed breakdown is presented below:

```
String year = new SimpleDateFormat("yy").format(new Date());
String month = new SimpleDateFormat("MM").format(new Date());
String day = new SimpleDateFormat("dd").format(new Date());
String min = new SimpleDateFormat("mm").format(new Date());
```

Figure 3.2: Formatting the Date

In the above screenshot, the date is put into a “YY/MM/DD” format.

```
int totalMin = (Integer.parseInt(year)*525600)+(Integer.parseInt(month)*43800)+
               (Integer.parseInt(day)*1440)+Integer.parseInt(min);
```

Figure 3.3: Converting date into minutes

Above, the date is converted into minutes by multiplying the year, month, and date by the number of minutes in each of them.

```
return totalMin;
```

Figure 3.4: Returning a timestamp

Here, the total number of minutes in the date is returned, thus becoming a timestamp.

Test Run with Timestamp:

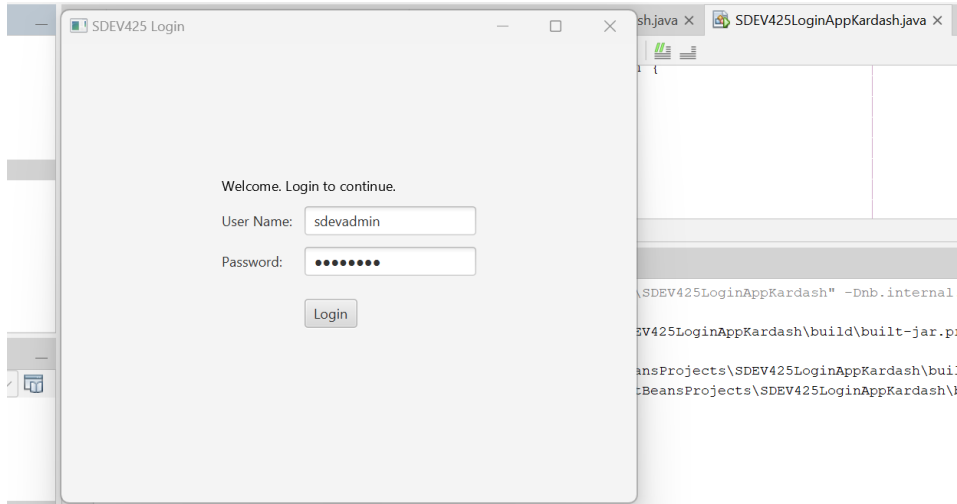


Figure 4.1: Entering Credentials for Test Run

Although the newly created function has not yet been used, we must ensure that the program functions well with it added.

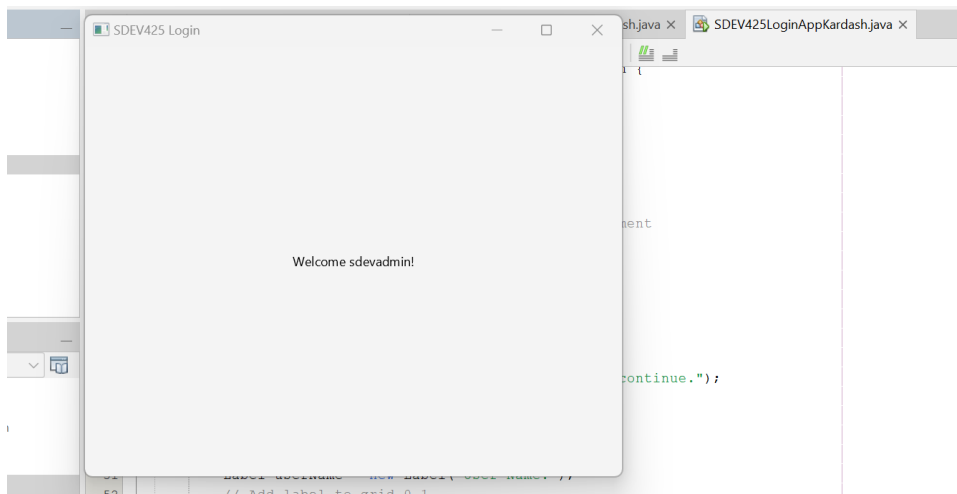


Figure 4.2: Successful login with added function.

The timestampMin function proved to not affect the program negatively.

Now that the timestamp control has been implemented, it is time to secure the hypothetical user's credentials. One of the best ways to do this is to use AC-7 – Unsuccessful Logon Attempts. According to the NIST Special Publication 800-53A Revision 5, this control involves setting a specific number of unsuccessful login attempts before the user is logged out of the system (NIST Special Publication 800-53A Rev.5, n.d.). The website CSF Tools, supported by NIST, states that after the number of unsuccessful login attempts is exceeded, the system should lock the user out until unlocked by an administrator, after a certain amount of time (AC-7, n.d.).

To implement this change, we must include an integer variable called “invalidAttempts”, which will be used to handle the times that a user logs in with incorrect credentials. The variable will be initiated right below the main class.

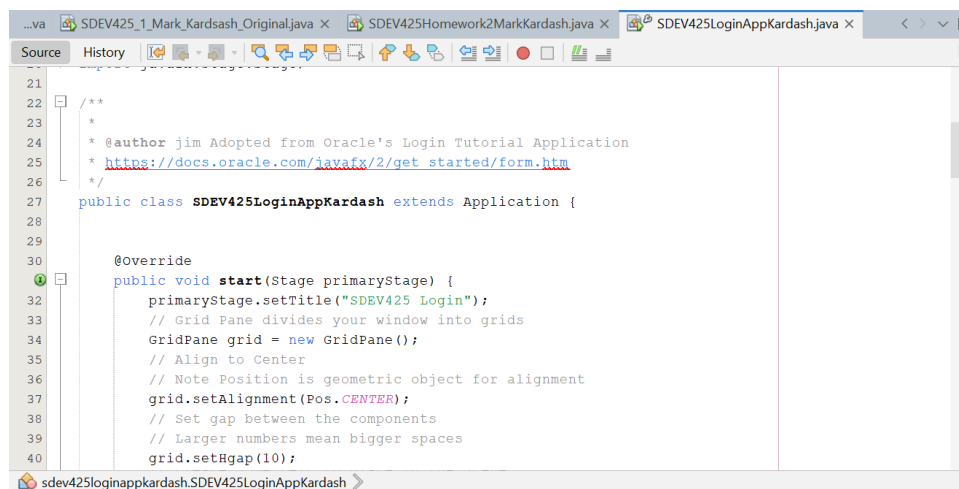
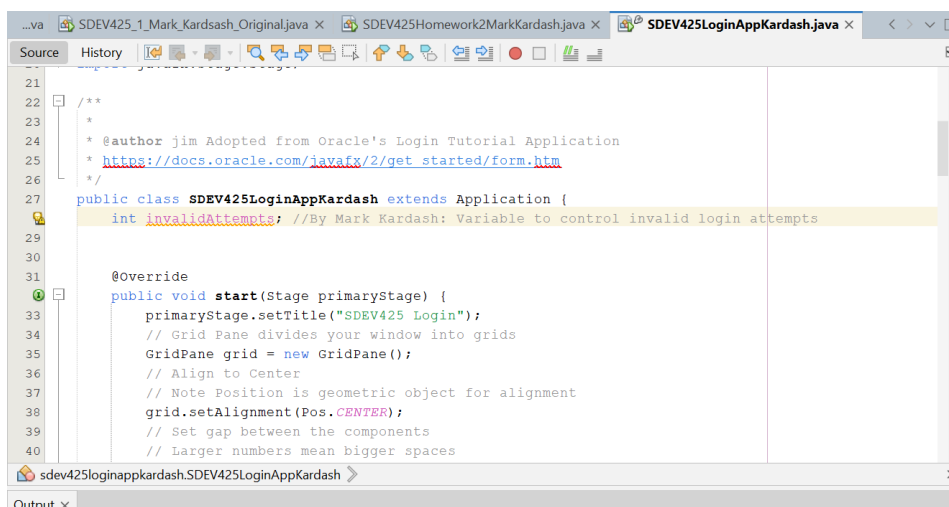


Figure 5.1: Original code without any attempt count.

As can be seen above, the application does not have any code restricting the number of access attempts.



```

21
22 /**
23  *
24  * @author jim Adopted from Oracle's Login Tutorial Application
25  * https://docs.oracle.com/javafx/2/get\_started/form.htm
26  */
27 public class SDEV425LoginAppKardash extends Application {
28     int invalidAttempts; //By Mark Kardash: Variable to control invalid login attempts
29
30
31     @Override
32     public void start(Stage primaryStage) {
33         primaryStage.setTitle("SDEV425 Login");
34         // Grid Pane divides your window into grids
35         GridPane grid = new GridPane();
36         // Align to Center
37         // Note Position is geometric object for alignment
38         grid.setAlignment(Pos.CENTER);
39         // Set gap between the components
40         // Larger numbers mean bigger spaces

```

Figure 5.2: Updated code with new “invalidAttempts” variable.

The new variable is now added to the code.

Now, our task is to put the new variable to its intended use. For this, the action event within the Event Handler would have to be significantly modified. However, for the attempt restriction to work, yet another security control would have to be put in place: “AUD-3: Content of Audit Records”. The reasoning behind this control is to document every action and event that occurs within the system and generate records of these actions to be reviewed by developers. The reason why AUD-3 intertwines with A-7 in this application is that each attempt to access the system, whether successful or failed, would have to be documented in a log. This way, the system can recognize when the login limit is reached, and, using the appropriate mechanism, lock the app in time. Before actually restricting the attempts, we will focus on their documentation. For this purpose, we create a new function named generateAttemptLog. This function will be responsible for documenting the user’s access attempts into a specially created log text file, using the bufferedWriter function. For the process to be better understood, the logic of these actions will be broken down into screenshots with explanations.

```

        isValid = true;
    }

    return isValid;
}

//By Mark Kardash: Function to place a timestamp
public static int timestampMin() {
    String year = new SimpleDateFormat("yy").format(new Date());
    String month = new SimpleDateFormat("MM").format(new Date());
    String day = new SimpleDateFormat("dd").format(new Date());
    String min = new SimpleDateFormat("mm").format(new Date());
    int totalMin = (Integer.parseInt(year)*525600)+(Integer.parseInt(month)*43800)+
        (Integer.parseInt(day)*1440)+Integer.parseInt(min);
}

```

Figure 6.1: Original Code prior to adding the new function

The generateAttemptLog function will be placed between the end of the original code, and the beginning of the “timestampMin” function.

```

7 //By Mark Kardash: Importing SimpleDateFormat and Date functions to help with the timestamp function
9 import java.text.SimpleDateFormat;
10 import java.util.Date;
11
12 //By Mark Kardash: Importing functions necessary for generateAttemptLog
13 import java.io.BufferedWriter;
14 import java.io.FileWriter;
15 import java.io.IOException;
16
17 import javafx.application.Application;
18 import javafx.event.ActionEvent;
19 import javafx.event.EventHandler;
20 import javafx.geometry.Pos;
21 import javafx.scene.Scene;
22

```

Figure 6.2 : New Functions Imported for generateAttemptLog.

Before proceeding with the function creation, we import all the necessary elements we will need for it: BufferedWriter, FileWriter, and IOException

By checking the username, this function will determine if previous login attempts have been made. Every attempt will be documented in a specially created log called “attemptLog”.


```

public static void generateAttemptLog (String username) {

    String timestamp = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date());
    int currentMins = timestampMin();

    // By Mark Kardash: Starting a log via BufferedWriter
    String attemptLog = "Username: " + username + ", Failed login time: " + timestamp + " -" +
        currentMins + "\n";
    BufferedWriter writer = null;

    //By Mark Kardash: Writing a new log via BufferedWriter
    try {
        writer = new BufferedWriter(new FileWriter("attemptLog.txt"));
        writer.write(attemptLog);
    }
    //By Mark Kardash: Catching any possible exceptions
    catch (IOException io) {
        System.out.println("File IO Exception" + io.getMessage());
    }
}

```

Figure 7.1: “generateAttemptLog” function – Part 1

As seen above, while setting up the “generateAttemptLog” function, we use the “timeStampMin” function within it, to know exactly when each attempt to login was made. We set up a string to document the attempts, and use BufferedWriter to turn it into a log. We also throw an exception in case the log fails to be created.

```

//By Mark Kardash: "Try" loop to close file
finally {
    try {
        if (writer != null) {
            writer.close();
        }
    }
    //By Mark Kardash: Catching any possible exception to closing the file
    catch (IOException io) {
        System.out.println("Error: File not properly closed." + io.getMessage());
    }
}
}

```

Figure 7.2: “generateAttemptLog” function – Part 2.

Here, we include a “try” loop inside a “finally” statement, to close the file when we are done with the function. Should the file fail to close properly, we include an “IOException” with an error message.

Test Run with “generateAttemptLog” fuction.

As before, we must now test whether the program functions properly with the addition of this new code.

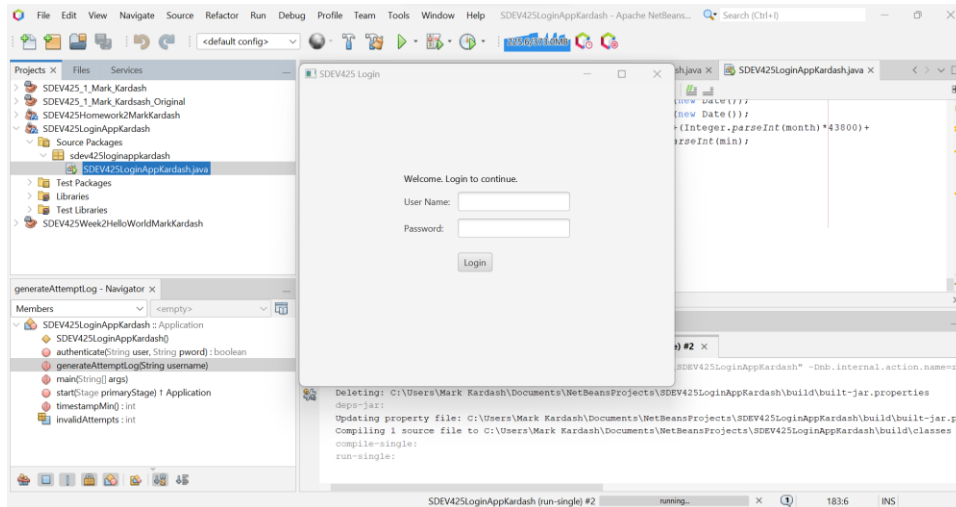


Figure 8.1: Successful window appearance with “generateAttemptLog” function.

The first step of the test went by successfully, with the Login window opening as expected.

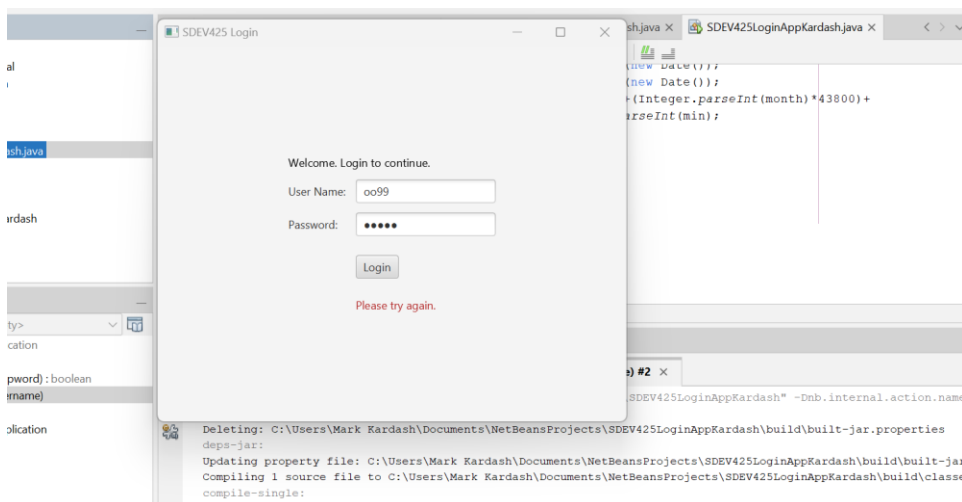


Figure 8.2: Invalid Credentials First Attempt

Upon entering invalid user credentials for the first time, the error message is displayed as expected.

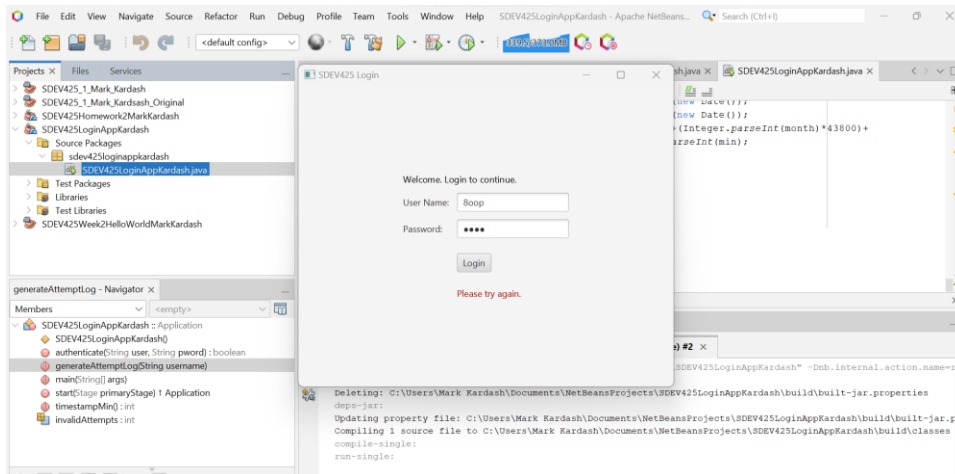


Figure 8.3: Invalid Credentials Second Attempt

The same thing can be seen with the second attempt as well.

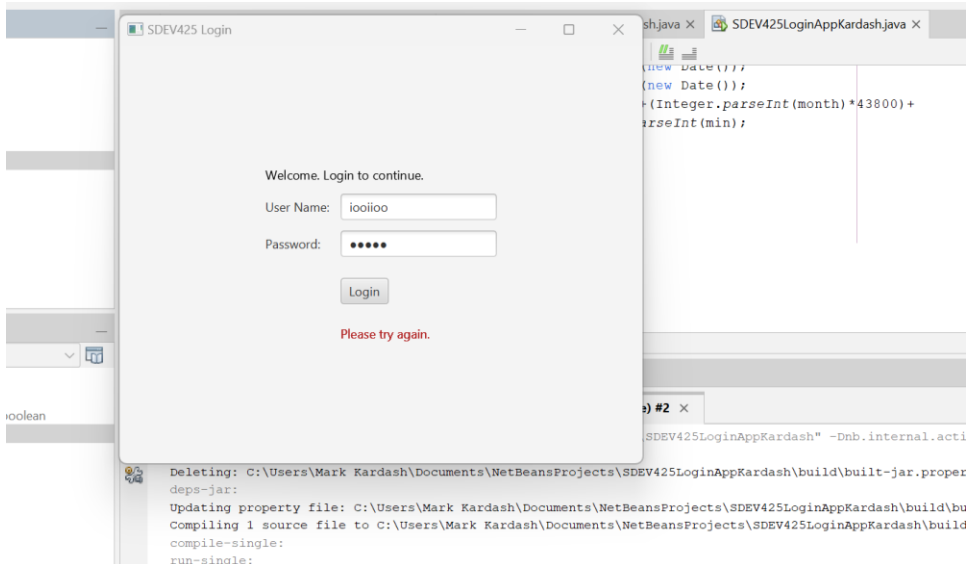
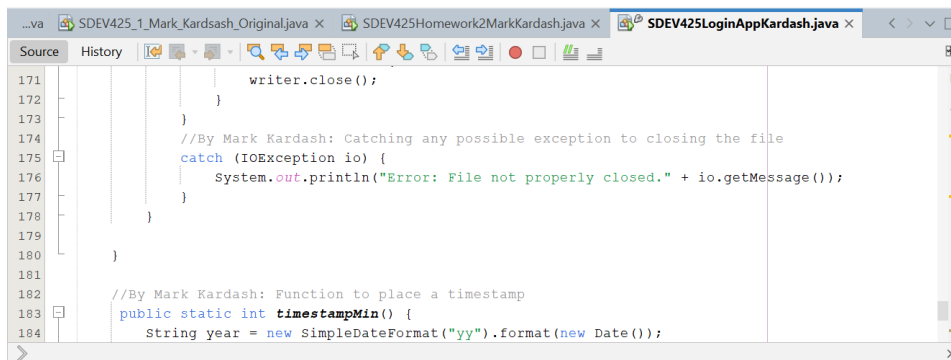


Figure 8.4: Invalid Credentials Third Attempt

On the third attempt, the program encourages the user to try again, and does not lock them out of the system. This is because, although the login attempts are being recorded, there has not been any action to restrict their number yet. This option will be added further down the line, as the program is perfected.

With the task of documentation now completed, we can move forward to the actual restriction of login attempts, and thus, the implementation of AC-7. This is where we create a function called “logReader”. Its purpose will be to read the previously created “attemptLog.txt” file, and, based on the number of attempts recorded, take the appropriate course of action, whether locking the system or allowing for more attempts.



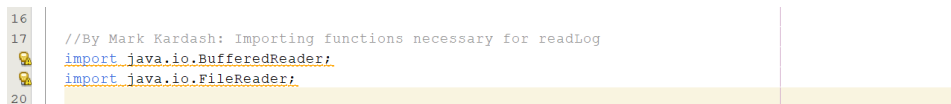
```

171         writer.close();
172     }
173 }
174 //By Mark Kardash: Catching any possible exception to closing the file
175 catch (IOException io) {
176     System.out.println("Error: File not properly closed." + io.getMessage());
177 }
178 }
179 }
180 }
181 }
182 //By Mark Kardash: Function to place a timestamp
183 public static int timestampMin() {
184     String year = new SimpleDateFormat("yy").format(new Date());

```

Figure 9.1: Code before addition of “readLog” function.

The “readLog” function will be placed between the “generateLogAttempt” and the “timestampMins” functions.



```

16
17 //By Mark Kardash: Importing functions necessary for readLog
18 import java.io.BufferedReader;
19 import java.io.FileReader;
20

```

Figure 9.2: Importing functions for “readLog”

Before writing the function itself, we import BufferedReader and FileReader, both of which we will soon need.



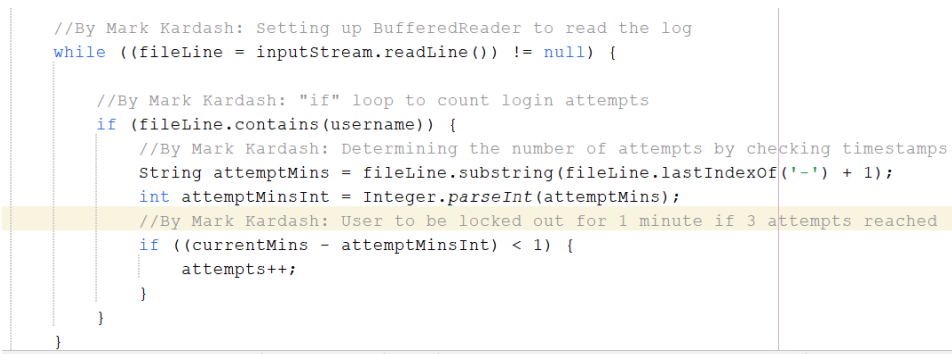
```

187 public static boolean readLog(String username) {
188
189     boolean recentAttempt = true;
190     int attempts = 0;
191     int currentMins = timestampMin();
192
193     BufferedReader inputStream = null;
194     String fileLine;
195     String filename = "attemptlog.txt";
196
197     try {
198         inputStream = new BufferedReader(new FileReader(filename));
199

```

Figure 9.3: “readLog” function-Part 1

In this part of the “readLog” function, we initialize the variables for the attempt count and log via `BufferedReader` and other functions. We then include a “try” loop to read the file.



```

//By Mark Kardash: Setting up BufferedReader to read the log
while ((fileLine = inputStream.readLine()) != null) {

    //By Mark Kardash: "if" loop to count login attempts
    if (fileLine.contains(username)) {
        //By Mark Kardash: Determining the number of attempts by checking timestamps
        String attemptMins = fileLine.substring(fileLine.lastIndexOf('-') + 1);
        int attemptMinsInt = Integer.parseInt(attemptMins);
        //By Mark Kardash: User to be locked out for 1 minute if 3 attempts reached
        if ((currentMins - attemptMinsInt) < 1) {
            attempts++;
        }
    }
}

```

Figure 9.4: “readLog” function-Part 2

Within the newly created “try” loop, we build an “if”-loop, where we check the timestamps recorded in “attemptlog.txt”. If 3 attempts are recorded, the program will lock the user out for 1 minute.

```

214 //By Mark Kardash: Catching exception
215 } catch (IOException io) {
216     System.out.println("File IO exception" + io.getMessage());
217 } finally {
218     //By Mark Kardash: Closing the files
219     try {
220         if (inputStream != null) {
221             inputStream.close();
222         }
223         //By Mark Kardash: Exception for closing the files.
224     } catch (IOException io) {
225         System.out.println("Error: File failed to close properly." + io.getMessage());
226     }
227 }
228 if (attempts == 0) {
229     recentAttempt = false;
230 }
231 return recentAttempt; //By Mark Kardash: Counting by the most recent attempt
232 }

```

Figure 9.5: “readLog” function-Part 3

Here, we come to the last part of the function, catching an exception should it fail, closing files and streams, as well as catching another exception if they fail to close. Finally, we return the count of the user’s most recent login attempt, as this is how they will be properly documented.

With the “readLog” function constructed, it is time to actually use it within the code. For this, we will modify the action event within the event planner, and the main “if” -loop for the code.

```

Source History
public void handle(ActionEvent e) {
96     // Authenticate the user
97     boolean isValid = authenticate(userTextField.getText(), pwBox.getText());
98     // If valid clear the grid and Welcome the user
99     if (isValid) {
100         grid.setVisible(false);
101         GridPane grid2 = new GridPane();
102         // Align to Center
103         // Note Position is geometric object for alignment
104         grid2.setAlignment(Pos.CENTER);
105         // Set gap between the components
106         // Larger numbers mean bigger spaces
107         grid2.setHgap(10);
108         grid2.setVgap(10);
109         Text scenetitle = new Text("Welcome " + userTextField.getText() + "!");
110         // Add text to grid 0,0 span 2 columns, 1 row
111         grid2.add(scenetitle, 0, 0, 2, 1);
112         Scene scene = new Scene(grid2, 500, 400);
113         primaryStage.setScene(scene);
114         primaryStage.show();

```

Figure 10.1: “If”-loop prior to new modifications

This is how the “if”-loop code of the app looked before it was modified to restrict login attempts.

```

@Override
public void handle(ActionEvent e) {

    //By Mark Kardash: Counting attempts to login
    String username = userTextField.getText();
    boolean recentAttempt = readLog(username);
    if (recentAttempt == false) {
        invalidAttempts = 0;
    }
    // Authenticate the user

```

Figure 10.2: Adding login attempt counter

The first step in this complicated process is adding a system to count the user's login attempts, which will be reset to 0 via an "if" loop if no recent attempts have been made.

```

//By Mark Kardash: Additional "if" loop to count attempts
if (invalidAttempts < 3) {
    // Authenticate the user
    boolean isValid = authenticate(userTextField.getText(), pwBox.getText());
    // If valid clear the grid and Welcome the user
    if (isValid) {
        grid.setVisible(false);
        GridPane grid2 = new GridPane();
        // Align to Center

```

Figure 10.3: Creating "If"-loop for counting attempts

Now, we place the original "if" loop for the variable "isValid" into a new one, which will ensure that the user does not get more than three login attempts.

```

    } else {
        final Text actiontarget = new Text();
        grid.add(actiontarget, 1, 6);
        actiontarget.setFill(Color.FIREBRICK);
        actiontarget.setText("Please try again.");

        //By Mark Kardash: Adding by increments of 1 to reach 3 invalid attempts
        generateAttemptLog(username);
        invalidAttempts++;
    }
}
}
});

```

Figure 10.4: Incrementing attempts by 1

In order to properly count the number of attempts, we make it so that each time a new invalid attempt is made the counter adds it to the total, until it reaches 3.

Now comes the time for the final stage of implementing AC-7. That is, building the code that will restrict login attempts. It will be built as the “else” part of the new “if” loop, placed almost immediately after the previous code.

```

else {
    //By Mark Kardash: Locking user out if attempt counter reaches 3
    grid.setVisible(false);
    GridPane grid3 = new GridPane();
    // By Mark Kardash: Putting object and message in the center if counter reaches 3
    grid3.setAlignment(Pos.CENTER);
    // By Mark Kardash: Setting gaps and dimensions
    grid3.setHgap(10);
    grid3.setVgap(10);
    //By Mark Kardash: Warning message about too many login attempts
    Text scenetitle = new Text("Too many login attempts! Please try again in 1 minute!");
    // By Mark Kardash: Add text to grid 0,0 span 2 columns, 1 row
    grid3.add(scenetitle, 0, 0, 2, 1);
    Scene scene = new Scene(grid3, 500, 400);
    primaryStage.setScene(scene);
    primaryStage.show();
}

```

Figure 10.5: Function to control login attempts

In the above function, we set the grid to be visible by the user, and initiate a new grid pane. We set the gaps of the grid from the window, and display the message about too many logins. We then chose the dimensions for the message and its surroundings, and make it visible to the user via the “show” command.

Test Run for Login Attempts Restrictions:

Now that we are done with implementing AC-7 – Unsuccessful Logon Attempts, we must test whether the app runs as intended with all of the new additions.

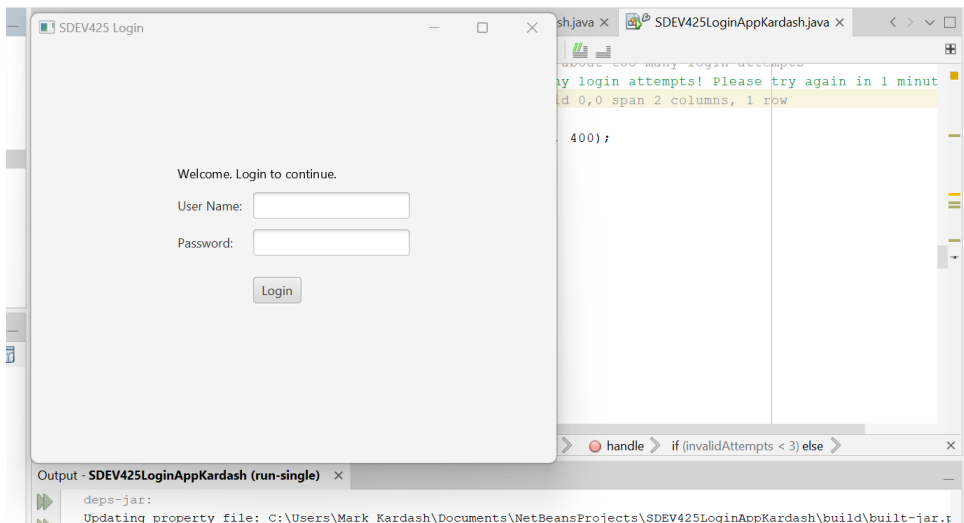


Figure 11.1: The Login window appears successfully

Upon running the app, we see that, as expected, it asks us for credentials, which means that the first phase of the test has been completed successfully. We also see that the document “attemptLog.txt” has been created, and, for the sake of this assignment, moved into the source code of the project for greater convenience.









 build	2/6/2024 9:50 PM	File folder
 dist	2/3/2024 5:30 PM	File folder
 nbproject	2/3/2024 5:07 PM	File folder
 src	2/3/2024 5:07 PM	File folder
 test	2/4/2024 8:18 PM	File folder
 attemptLog	2/6/2024 9:59 PM	Text Document
 build	2/3/2024 5:07 PM	Microsoft Edge HTM
 manifest.mf	2/3/2024 5:07 PM	MF File

Figure 11.2: The file “attemptLog.txt” in the program root folder.

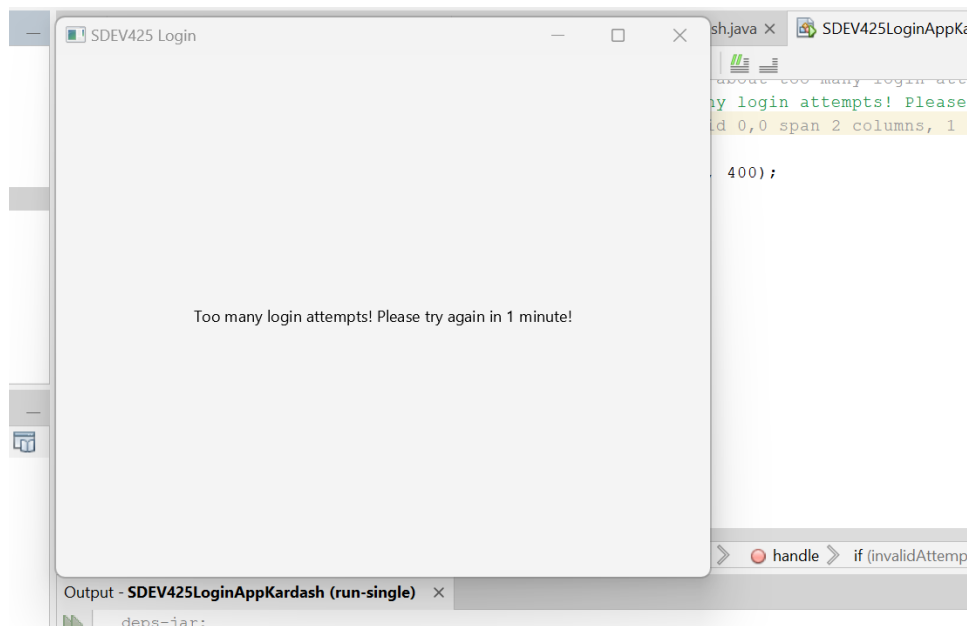


Figure 11.3: Message about too many login attempts

After entering invalid credentials and clicking on the “Login” button three consecutive times, the expected message appears on the screen. With both the log having been created and the message displayed, we can declare this test as successful.

Being finally done with AU-8, AC-7, and AU-3, I decided to move on to implementing the control IA-2(1): Identification and Authentication (Organizational Units)| Network Access to Privileged Accounts. As explained by CSF Tools, in this control, “The information system uniquely identifies and authenticates organizational users (or processes acting on behalf of organizational users).” (IA-2, Control Statement, n.d.). In other words, companies abiding by this rule employ various techniques, such as token, biometrics, or multi-factor authentication, to identify employees (CSF, IA-2, n.d.). For my application, I was thinking of doing multi-factor authentication, perhaps with a message sent to their phone number or email. However, after struggling with it for a bit, I must admit that I have no idea how to properly do this, so I had no choice but to skip this part of the assignment.

With the implementation of the previous control ending up a huge bust, I decided to move on to the easiest one: AC-8: System Use Notification. CSF Tools describes this control as a measure to warn users about certain characteristics of the system, especially any privacy issues they may experience (AC-8, n.d.). For example, the app may warn users about whether they are entering a government-controlled system, any monitoring they will be subject to, and the danger of facing legal consequences should they access the system unauthorized. In the case of our application, this is as simple as setting up a new “Text” instance, and printing out a message.

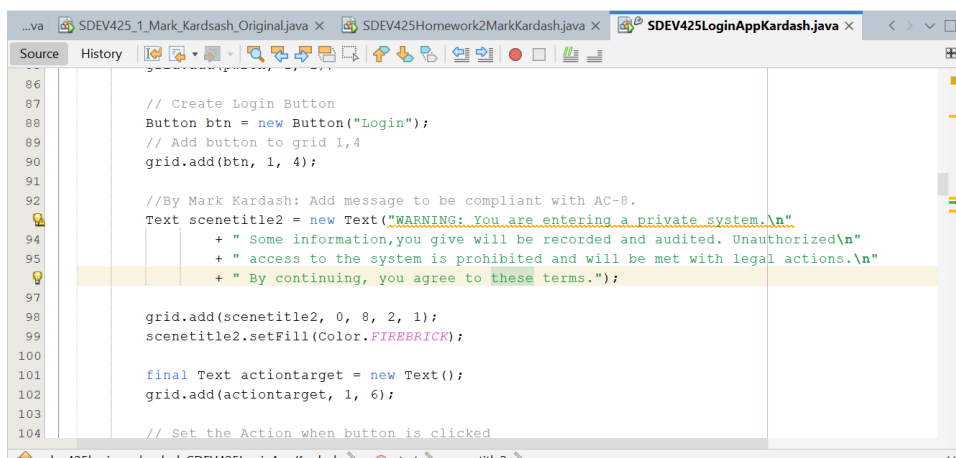


Figure 13: Warning message added

To be compliant with AC-8, we add a message for the user, about the nature of our system, the possibility of their information being recorded, as well as the possibility of legal actions should unauthorized access occur. The message is added right after the “Login” button.

Test Run with Warning Message:

As with all other enhancements, I had to re-test the program, to make sure it runs as intended with the new change.

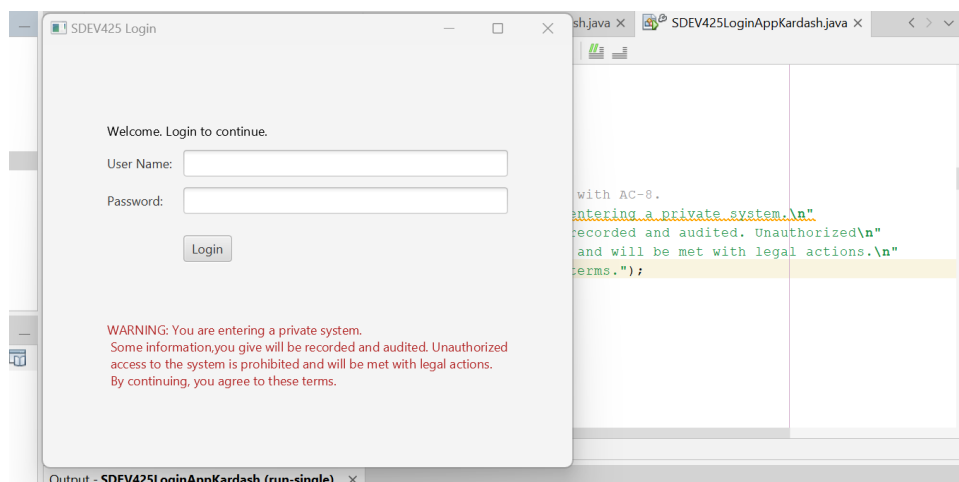


Figure 15: Warning message appears on Login window

The appearance of the warning message on the window confirms the successful result of the test.

Having implemented all the security measures (except IA-2(1)), I decided to enhance the program by adding another user to it. This will be a visiting user, whose access to the system will be limited, but authorized. All I need to do for this is add an extra button to the main stage (Window) and declare a new action event to handle it.

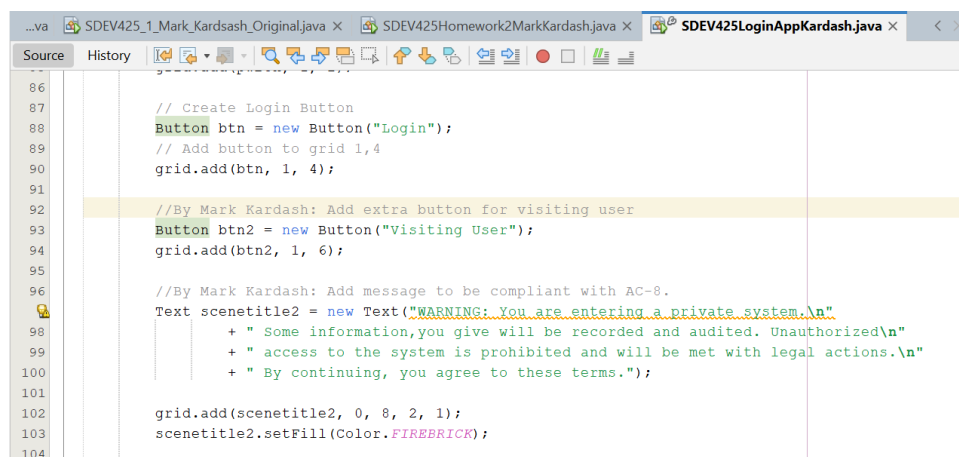


Figure 16.1: Adding second button to main stage.

A second button has been added to the main stage, for a visiting user without credentials to login.

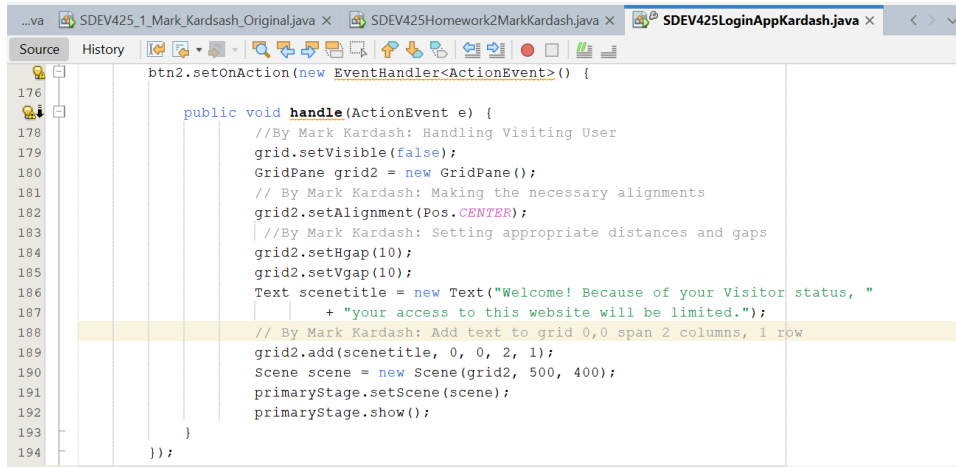


Figure 16.2: New event handler and action event

We now set up a second event handler, and, within it, a new action event, for the second button. It has the exact same characteristics as the first, making the grid visible, centering it, specifying dimensions, and displaying a welcome message. In this case, the message informs the visiting user that their access will be limited.

Test Run with New Visiting User:

Let us now see how the program deals with a new user being added to it.

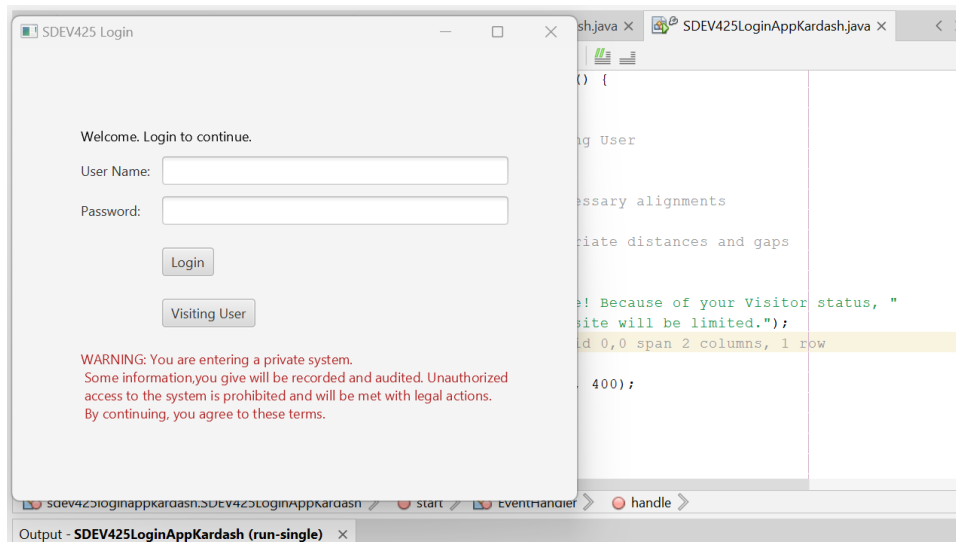


Figure 17.1: Visiting User Button Displayed

As we run the program, we see the “Visiting User” button displayed in the Login window, making the first step of the test a success.

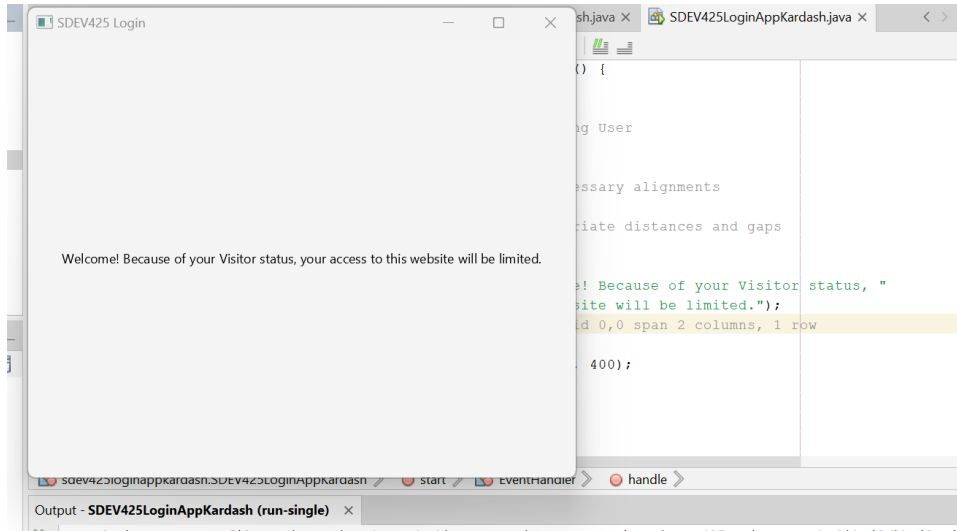


Figure 17.2: Visiting User Welcome Message Displayed

Clicking on the “Visiting User” button displays the appropriate welcome message

Summary:

This assignment was far more difficult and time consuming than the first one. I had difficulties getting some functions to work correctly, and I did not manage to provide two-step authentication for the app. However, with the exception of this one unapplied control, everything went quite smoothly at the end, with all tests that I ran passing successfully.

References

AC-7: Unsuccessful Logon Attempts. (n.d.). CSF Tools. <https://csf.tools/reference/nist-sp-800-53/r5/ac/ac-7/>

AC-8: System Use Notification. (n.d.). CSF Tools. <https://csf.tools/reference/nist-sp-800-53/r5/ac/ac-8/>

AU-3: Content Of Audit Records. (n.d.). CSF Tools. <https://csf.tools/reference/nist-sp-800-53/r4/au/au-3/>

AU-8: Time Stamps. (n.d.). CSF Tools. <https://csf.tools/reference/nist-sp-800-53/r5/au/au-8/>

IA-2: Identification And Authentication (Organizational Users). (n.d.). CSF Tools.

<https://csf.tools/reference/nist-sp-800-53/r4/ia/ia-2/>

National Institute of Standards and Technology. (n.d.). *NIST Special Publication 800-53A*

Revision 5-Assessing Security and Privacy Controls in Information Systems and

Organizations. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53Ar5.pdf>