Fall 2022 CMP_SC 3330 Final Project Documentation

Ethan Mick
University of Missouri
December 2022

I. Overview

Whether it be over a topic one is studying for at a University, a favorite TV show, certification, etc., being able to quiz oneself over trivia/key concepts is a very valuable resource to have at one's disposal. While many applications that provide this type of service already exist out there, I thought designing a "lite" version of such an application would be a properly-sized project for this class given the time-constraints/complexity expectations.

"Quizzer47" is a simple Java Swing GUI application that allows users to create accounts for themselves and take quizzes as signed in users. I designed it to utilize a JDBC connection to a mySQL database for user authentication/storage. Users can login to their accounts with their login info, and if the info is correct, they'll be brought to their personalized home page (New accounts can also be created). From there, they can navigate to the Quiz page, where they can then choose the quiz that they want to take. I also included functionality for allowing users to view their profile information, change their passwords, and logout.

II. Functionality Rundown

As I summarized above, the functionality of the project includes a user login system, a quiz taking UI, and a profile viewing UI.

A. User Login System

When LoginForm.java is run, the user is brought to a login page asking for username and password input. There are 2 buttons below this, one called "Submit" and one called "Create new user". Submit reads the user input in the text fields and checks to see if it

matches an existing user in the SQL database. If it does, the user is brought to their specific home page (current personalization functionality includes a background color based on the user's input for their favorite color when they created their account). When logging in, a temporary "User" object is instantiated and used as a parameter for all subsequent pages visited while the user remains logged in. If a user attempts to login to an account that does not exist, or attempts to create an account with an already existing username and/or unacceptable information fields, an error message will be displayed.

B. Quizzes

When a user clicks on the quizzes button from their home page, they are brought to a new quiz page. This is the page where all currently takeable quizzes are accessible. Each quiz is its own object that feeds specific information to the quiz page when a user selects that quiz. Because of this, I only provided one fully functioning quiz for this iteration of the project (again due to time constraints). Nonetheless, I believe I made it clear how new quizzes would be added to the application.

Additionally, quizzes in their current form in Quizzer47 are sets of 10 4-choice multiple choice questions. When a user starts a quiz, they are brought to a page displaying the first question. They make their decision on the answer, and then click next. Users cannot return to questions they've already answered. When next is pressed on the 10th question, the user's choices are compared with the answers (stored in that specific quiz class) and a score is determined. A score of 6 or higher indicates a passing score. The score is displayed to the user, and then they are brought back to the quiz home page where they can take another quiz.

C. Profile

When a user clicks on the profile button from their home page, they are brought to a page similar to the registration page, except there is no password field and the button at the bottom says "Change Password" instead. On this page, users can view their profile information, and if they click "Change Password", a form is opened that allows users to

input a new password and submit it. When the user clicks “submit”, a SQL update query is run for that user (key value username) and their password is successfully changed.

III. Required Rubric Elements

a. **Classes (more than one):**
    i. LoginForm.java
    ii. RegisterForm.java
    iii. HomePage.java
    iv. ProfilePage.java
    v. QuizPage.java
    vi. ChangePasswordForm.java
    vii. User.java
b. **At least one Subclass:**
    i. SeinfeldQuiz.java is a subclass of Quiz.java **(Line 8 of SeinfeldQuiz.java)**
    ii. Items i. Through vi. Above are technically subclasses of javax.swing.JFrame, but that is purely for code simplification
c. **At least one Abstract Class and/or Interface:**
    i. Quiz.java is an abstract class **(Line 7 of Quiz.java)**
    ii. PageInterface.java is an interface **(Line 15 of PageInterface.java)**
d. **The application should use exceptions to handle errors:**
    i. Try-catch statement in PageInterface.java
        1. **runPage(), lines 22-26**
        2. **setBackgroundColor(), lines 33-38**
    ii. Try-catch statement in LoginForm.java **(Lines 105-137)**
    iii. Try-catch statement in RegisterForm.java
        1. **RegisterForm() constructor, lines 162-181**
        2. **isInteger(), lines 195-200**
    iv. Try-catch statement in ChangePasswordForm.java **(Lines 67-78)**
e. **At least one collection class:**
    i. 4 ArrayList<String> variables in QuizPage.java **(Lines 29-30, usage at various instances throughout the file)**

ii. Multiple ArrayList-returning abstract methods in Quiz.java
    1. **getQuestions(), getChoices(), getAnswers()**
    2. **scoreQuiz() (static method), line 23, line 26 (usage))**
iii. Multiple ArrayLists in SeinfeldQuiz.java **(Lines 33, 37, 53, 73-84 (.add's))**

f. Overall structure / User experience

i. With LoginForm.java as the launch file, the application presents itself in an easy to understand way and all functionality is clearly presented to the user in the form of buttons and information pop-ups.

ii. While a more refined (meticulously worked on over months) Java Swing project might implement the specificities of the GUI more smoothly, I appreciated the relative ease of creating classes for each page that will be displayed to the user. Some are fairly generic, while others are reused with different parameters in different ways. This permitted me to create a Swing app in the time constraints required for the project, and to do so in a way that effectively utilizes the core concepts we've gone over in class this semester.

IV. Notes for Grader Testing

I used JDBC and a mySQL database for the user data because I wanted to get familiar with using it and overall I really enjoy the functionality that it provides. However, I realize not everyone testing may have JDBC hooked up as I do in Eclipse, NetBeans, etc. so I chose to omit any kind of "Create table" functionality that might permit ease of usage on another machine. (Additionally, for testing the DB I created temporary root access that would certainly be machine-specific) **(Ex: LoginForm.java line 109)**. With all of this considered, I'd like to clarify that using this application as I designed it with the database is not my intention for grading, unless graders want to set up a database in the way in which I did. In other words, what I'm saying is, if you attempt to use the database, an exception will likely occur, but this is not a "crash" as the rubric describes it.

To get around this for grading, though, I hard-coded 2 "existing" users into LoginForm.java **(lines 89-106)**, as well as one "to-exist" user in RegisterForm.java

(lines 152-160) so that both of these pathways to the Homepage can be tested. These are the only ways to gain access to quizzer47 when testing, and should be considered the only inputs for these fields in the context of the application-to-be-graded (unless, again, you wanna mimic my setup). Each test profile also has a unique “favorite color” field, so that that functionality can be displayed. Additionally, ChangePassword will recognize if a test user is being used and the password will not be changed, as these must stay constant for testing **(lines 60-69)**.

A. **LoginForm.java info:**

Username	**Password**
user	pass
123	456

B. **RegisterForm.java info:**

Username	**Password**	**Fav Color**	**Age**	**Email**	**Phone**
test	pass	green	21	c@t	1234567890