### Test Cases for LoginApp

#### 1. Test Case 1: Valid User Login

* Description: Verify that the application authenticates and welcomes a user when valid email and password are provided.
* Test Steps:
  1. Insert a valid record in the User table (e.g., email = "test@example.com", password = "password123", and name = "Test User").
  2. Provide the same email during login.
  3. Check if the returned userName matches the name stored in the database.
* Expected Result: A successful login message: "Welcome, Test User!".

#### 2. Test Case 2: Invalid User Login

* Description: Verify that the application shows an error message when an unregistered email is provided.
* Test Steps:
  1. Do not insert a record for email = "invalid@example.com" in the database.
  2. Try to login with this email.
  3. Capture the login message or error.
* Expected Result: A failed login message: "User not found.".

#### 3. Test Case 3: Database Connection Failure

* Description: Verify that the application gracefully handles database connectivity issues.
* Test Steps:
  1. Provide incorrect database credentials (e.g., invalid DB\_URL or DB\_PASSWORD).
  2. Run the application and try to log in.
* Expected Result: An appropriate error message should be displayed, and the stack trace should log the exception.

#### 4. Test Case 4: Empty Input Validation

* Description: Verify that the application handles empty email and password fields gracefully.
* Test Steps:
  1. Leave the email and password fields empty.
  2. Try to login.
* Expected Result: The application should not proceed with the database query and should show a warning: "Email and Password cannot be empty.".

#### 5. Test Case 5: SQL Injection Protection

* Description: Verify that the application prevents SQL injection.
* Test Steps:
  1. Provide an email value like "test@example.com' OR '1'='1".
  2. Attempt to login.
  3. Verify if the database query is executed safely and does not bypass authentication.
* Expected Result: The login should fail, and the application should not allow unauthorized access.

**JUnit Test Code for LoginApp**  
  
import org.junit.jupiter.api.\*;

import static org.junit.jupiter.api.Assertions.\*;

import java.sql.\*;

class LoginAppTest {

private static final String DB\_URL = "jdbc:mysql://localhost:3306/softwaretesting";

private static final String DB\_USER = "root";

private static final String DB\_PASSWORD = "12345678";

private LoginApp loginApp;

@BeforeAll

static void setupDatabase() throws Exception {

try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {

String createTable = """

CREATE TABLE IF NOT EXISTS User (

Email VARCHAR(255) PRIMARY KEY,

Name VARCHAR(255) NOT NULL,

Password VARCHAR(255) NOT NULL

)

""";

conn.createStatement().execute(createTable);

String insertUser = "INSERT INTO User (Email, Name, Password) VALUES (?, ?, ?)";

PreparedStatement stmt = conn.prepareStatement(insertUser);

stmt.setString(1, "test@example.com");

stmt.setString(2, "Test User");

stmt.setString(3, "password123");

stmt.executeUpdate();

}

}

@AfterAll

static void cleanupDatabase() throws Exception {

try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {

conn.createStatement().execute("DROP TABLE IF EXISTS User");

}

}

@Test

void testValidLogin() {

loginApp = new LoginApp();

String userName = loginApp.authenticateUser("test@example.com");

assertEquals("Test User", userName, "Valid login failed.");

}

@Test

void testInvalidLogin() {

loginApp = new LoginApp();

String userName = loginApp.authenticateUser("invalid@example.com");

assertNull(userName, "Invalid login should return null.");

}

@Test

void testEmptyInputValidation() {

// Simulate empty input validation

loginApp = new LoginApp();

// Cannot automate GUI-related validation in unit tests directly, this would need a functional test framework

}

@Test

void testDatabaseConnectionFailure() {

LoginApp appWithInvalidDB = new LoginApp() {

@Override

public String authenticateUser(String email) {

try (Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/invalidDB", DB\_USER, DB\_PASSWORD)) {

return null;

} catch (Exception e) {

return "DB\_ERROR";

}

}

};

String userName = appWithInvalidDB.authenticateUser("test@example.com");

assertEquals("DB\_ERROR", userName, "Database connection failure not handled properly.");

}

}  
  
  
**Add Input Validation: Modify the LoginAction class to handle empty fields:**

@Override

public void actionPerformed(ActionEvent e) {

String email = emailField.getText();

String password = new String(passwordField.getPassword());

if (email.isEmpty() || password.isEmpty()) {

JOptionPane.showMessageDialog(null, "Email and Password cannot be empty.", "Validation Error", JOptionPane.ERROR\_MESSAGE);

return;

}

String userName = authenticateUser(email);

if (userName != null) {

JOptionPane.showMessageDialog(null, "Welcome, " + userName + "!", "Login Successful", JOptionPane.INFORMATION\_MESSAGE);

} else {

JOptionPane.showMessageDialog(null, "User not found.", "Login Failed", JOptionPane.ERROR\_MESSAGE);

}

}