DOCUMENTATION
BankATMManagement Testing Project Documentation

---

1. Introduction

Overview of the Project

The **BankATMManagement system** is a Java-based application designed to simulate banking operations securely and efficiently. It utilizes **JavaFX** for its user interface and **SQLite** as its lightweight database solution. Key features include:

- **User Account Management**: Facilitating the creation and modification of accounts.
- **Secure Transactions**: Enabling deposits, withdrawals, and fund transfers.
- **Balance Inquiry**: Allowing users to check current account balance.
- **Random Motivational Quotes**: Displayed for user engagement on the UI.

The project includes comprehensive testing to validate functionality, ensure usability, and maintain reliability.

Purpose of Testing and Static Analysis

This project embraces **static** and **dynamic** testing methodologies to improve stability, security, and the quality of the codebase. Goals of testing include:

- Ensuring system **adheres to functional requirements** without failure.
- Proactively identifying **performance issues**, **code smells**, and **security vulnerabilities** through **static analysis** tools.
- Validating application behavior under various **real-world use cases** using unit, integration, and system tests.

---

2. Qodana Static Testing Results

Qodana Usage

Qodana is a static analysis tool used to perform an in-depth inspection of the project codebase. The main areas inspected include:

Enabled Inspections:

- **JavaFX Rules**:
  - Ensures correct usage of GUI components.
- **Object-Oriented Programming**:
  - Validates adherence to core principles like inheritance and encapsulation.
- **Performance & Security**:
  - Detects inefficient code and possible vulnerabilities.

Issues Found & Resolutions

Code Issues:

Problem Identified	Location	Resolution
Redundant Code	AccountInfoController.java	Refactored repetitive logic to improve readability and reduce size.
Security Vulnerability: Missing input validation	ChangePassword() method	Added input validation to prevent **invalid** or **malicious** inputs.

> **Conclusion:** Qodana proved to be highly effective in identifying potential problem areas early in development, preventing bugs and vulnerabilities from surfacing later.

---

3. Testing Analysis

Testing was categorized into **Dynamic Testing** approaches with detailed methodologies, including **boundary value testing**, **code coverage analysis**, and **functional testing**.

Boundary Value Testing (BVT)

Boundary values were defined based on edge-case inputs for critical methods.

ChangePassword Method part of the UserPageController.java file

Test Case ID	oldpass	newpass	passretype	Expected Outcome
1	""	"123456"	"123456"	**Fail**: "Please Fill Up Everything Correctly"
2	"oldpwd"	""	""	**Fail**: "Please Fill Up Everything Correctly"
3	"oldpwd"	"123456"	"654321"	**Fail**: "Password Confirmation Failed"
4	"oldpwd"	"123456"	"123456"	**Pass**: "Password Changed"

AccountInfo Method

Test Case ID	UserID	Expected Outcome
1	""	**Fail**: "Invalid UserID"
2	"1"	**Pass**: Account information found
3	"9999"	**Pass**: Account information not found but no crash (valid response handled).

Worked by Deni Frasheri and Melvina Baci

3.2 Code Coverage Testing achieved by **ChangePasswordTest Class**

The ChangePasswordTest class performs various types of coverage testing for the **ChangePassword method** in the AccountInfoController class.

Here is a detailed analysis of the coverage achieved:

Statement Coverage
Achieved: The tests ensure that various statements within the ChangePassword method are executed.
For example, **setting error messages and executing SQL** statements.

Example: The test **testEmptyNewPassword** ensures that the statement setting the error message "Please Fill Up Everything Correctly." is executed.

Branch Coverage
Achieved: The tests cover different branches of the decision points within the ChangePassword method.
For example, checking

- if the new password is empty
- if the old password is correct
- if the passwords match.

Example: The test **testPasswordMismatch** covers the branch where the new password and retyped password do not match, resulting in the error message "Password Confirmation Failed."

Condition Coverage
Partially Achieved: The tests evaluate different conditions within the ChangePassword method.
However, to fully achieve condition coverage, **each boolean sub-expression** must be evaluated to both true and false.

Example: The test **testEmptyOldPassword** evaluates the condition where :
the old password is empty, but additional tests are needed to evaluate all boolean sub-expressions to both true and false.

MC/DC (Modified Condition/Decision Coverage)

Partially Achieved: The tests ensure that changing the value of conditions (old password correctness, new password match) independently affects the outcome of the decision.
However, to fully achieve MC/DC, **each condition in a decision** must be shown to **independently affect the outcome of the decision**.

Example: The test **testWrongOldPassword** shows that changing the old password to an incorrect value independently affects the outcome, resulting in the error message "Wrong Password."

Coverage Type	Achieved	Description	Example Test
Statement Coverage	Achieved	The tests ensure that various statements within the `ChangePassword` method are executed. For example, setting error messages and executing SQL statements.	`testEmptyNewPassword` ensures that the statement setting the error message "Please Fill Up Everything Correctly." is executed.
Branch Coverage	Achieved	The tests cover different branches of the decision points within the `ChangePassword` method. For example, checking if the new password is empty, if the old password is correct, and if the passwords match.	`testPasswordMismatch` covers the branch where the new password and retyped password do not match, resulting in the error message "Password Confirmation Failed."
Condition Coverage	Partially Achieved	The tests evaluate different conditions within the `ChangePassword` method. However, to fully achieve condition coverage, each boolean sub-expression must be evaluated to both true and false.	`testEmptyOldPassword` evaluates the condition where the old password is empty, but additional tests are needed to evaluate all boolean sub-expressions to both true and false.

MC/DC (Modified Condition/Decision Coverage)	Partially Achieved	The tests ensure that changing the value of conditions (e.g., old password correctness, new password match) independently affects the outcome of the decision. However, to fully achieve MC/DC, each condition in a decision must be shown to independently affect the outcome of the decision.	`testWrongOldPassword` shows that changing the old password to an incorrect value independently affects the outcome, resulting in the error message "Wrong Password."

Additional Tests for Full Coverage(Recommendations) as per the documentations request

Test Name	Description	Purpose
Test with Non-Matching New Passwords	Ensure that the condition where the new password and retyped password do not match is covered.	To cover the condition where the new password and retyped password do not match.
Test with Correct Old Password but Empty New Password	Ensure that the condition where the old password is correct but the new password is empty is covered.	To cover the condition where the old password is correct but the new password is empty.
Test with Correct Old Password and Matching New Passwords	Ensure that the condition where the old password is correct and the new passwords match is covered.	To cover the condition where the old password is correct and the new passwords match.

4. Unit Testing, Integration Testing, System Testing

4.1 Unit Testing

Validates isolated methods and individual components.

Example: Quotes.returnQuotes()

Test Case ID	Input (Random Value)	Expected Output	Actual Output	Result
1	0	"To accomplish great things..."	"To accomplish great things..."	Pass
2	1	"Victory is always possible..."	"Victory is always possible..."	Pass

Mocking was used to ensure reliability and stability when testing random-dependent values.

4.2 Integration Testing

AccountInfoTest tests the interaction between the **AccountInfoController** and **the database connection**, **albeit with the database connection being mocked**. This ensures that the controller behaves correctly when interacting with the database, **without needing an actual database.**

Explanation:

Worked by Deni Frasheri and Melvina Baci

Initialization:

@BeforeAll initializes the JavaFX toolkit and **mocks the static DbConnection** class.
@BeforeEach sets up the controller and its fields before each test.
@AfterAll closes the static mocks after all tests.
Tests:

testEmptyUserID: Tests the controller's behavior when UserID is empty.
testValidUserID: Tests the controller's behavior with a valid UserID and ensures it correctly populates the fields.
testNonexistentUserID: Tests the controller's behavior with a nonexistent UserID and ensures it handles it gracefully.

Integration Testing for AccountInfoController

The integration tests for the AccountInfoController class ensure that **the controller interacts correctly with the database and updates the UI components accordingly.** The tests use the **JavaFX toolkit** and **mock the database connection** to simulate different scenarios.

Setup and Initialization

- JavaFX Toolkit Initialization:

The JavaFX toolkit is initialized in the @BeforeAll method using a CountDownLatch to ensure it starts up correctly before running any tests.

- Mocking Database Connection:

Static mocking of the DbConnection class is done using mockStatic.

Mock objects for **Connection**, **PreparedStatement**, and **ResultSet** are created.

The DbConnection.Connection method is mocked to return the mocked Connection object.

- **Controller** Setup:

Before each test, **the AccountInfoController is instantiated**, and its UI components (**TextField** and **TextArea**) are initialized using Platform.runLater to ensure they are created on the JavaFX Application Thread.

- Closing Mocks:

After all tests, the static mocks are closed in the @AfterAll method.

Test Cases

Test Empty UserID:

1.Sets UserID to an empty string.

2.Calls the AccountInfo method and verifies that all UI components remain empty.

3.Ensures the method handles an empty UserID gracefully without throwing exceptions.

Test Valid UserID:

1.Sets UserID to "1".

2.Mocks the database interactions **to return valid user data.**

2.Calls the AccountInfo method and verifies that the UI components are populated with the correct data.

3.Ensures no exceptions are thrown for a valid UserID.

Test Nonexistent UserID:

1.Sets UserID to "9999".

2.Mocks the database interactions to simulate a nonexistent user.

3.Calls the AccountInfo method and verifies that all UI components remain empty.

4.Ensures the method handles a nonexistent UserID gracefully without throwing exceptions.

4.3 System Testing

Tests the complete application execution and workflow.

Scenario: Application Start-Up with Quotes

Test Case ID	Action	Expected Outcome	Result
1	Login with valid credentials	UserPage displays a random motivational quote.	Pass

---

4.5List of System Tests

AccountInfoController

Test Account Information Retrieval:

a. User initiates the AccountInfo request.

b. System retrieves and displays user information.

Test Change Password with Empty New Password:

a. User initiates the ChangePassword request with an empty new password.

b. System displays an error message.

Worked by Deni Frasheri and Melvina Baci

Test Change Password with Empty Old Password:

a. User initiates the ChangePassword request with an empty old password.

b. System displays an error message.

Test Change Password with Password Mismatch:

a. User initiates the ChangePassword request with mismatched new passwords.

b. System displays a password confirmation error message.

Test Change Password with Wrong Old Password:

a. User initiates the ChangePassword request with an incorrect old password.

b. System displays a wrong password error message.

Test Change Password with Correct Old Password and Matching New Passwords:

a. User initiates the ChangePassword request with the correct old password and matching new passwords.

b. System updates the password and displays a success message.

TransactionController

Test Deposit Transaction:

a. User initiates a deposit transaction.

b. System updates the account balance and displays a success message.

Test Withdrawal Transaction with Sufficient Balance:

a. User initiates a withdrawal transaction with sufficient balance.

b. System updates the account balance and displays a success message.

Test Withdrawal Transaction with Insufficient Balance:

a. User initiates a withdrawal transaction with insufficient balance.

b. System displays an error message.

Test Transfer Transaction:

a. User initiates a transfer transaction.

b. System updates the balances of both accounts and displays a success message.

LoginController

Test Successful Login:

a. User enters valid credentials.

b. System authenticates the user and navigates to the main dashboard.

Test Unsuccessful Login with Invalid Credentials:

a. User enters invalid credentials.

b. System displays an error message.

Worked by Deni Frasheri and Melvina Baci

Test Unsuccessful Login with Empty Credentials:

a. User leaves the username or password field empty.

b. System displays an error message.

Controller Class	Test Name	Description	Steps
AccountInfoController	Test Account Information Retrieval	Ensure the system retrieves and displays user information.	a. User initiates the `AccountInfo` request. b. System retrieves and displays user information.
AccountInfoController	Test Change Password with Empty New Password	Ensure the system displays an error message when the new password is empty.	a. User initiates the `ChangePassword` request with an empty new password. b. System displays an error message.
AccountInfoController	Test Change Password with Empty Old Password	Ensure the system displays an error message when the old password is empty.	a. User initiates the `ChangePassword` request with an empty old password. b. System displays an error message.
AccountInfoController	Test Change Password with Password Mismatch	Ensure the system displays a password confirmation error message when the new passwords do not match.	a. User initiates the `ChangePassword` request with mismatched new passwords. b. System displays a password confirmation error message.
AccountInfoController	Test Change Password with Wrong Old Password	Ensure the system displays a wrong password error message when the old password is incorrect.	a. User initiates the `ChangePassword` request with an incorrect old password. b. System displays a wrong password error message.

AccountInfoController	Test Change Password with Correct Old Password and Matching New Passwords	Ensure the system updates the password and displays a success message when the old password is correct and the new passwords match.	a. User initiates the `ChangePassword` request with the correct old password and matching new passwords. b. System updates the password and displays a success message.
TransactionController	Test Deposit Transaction	Ensure the system updates the account balance and displays a success message for a deposit transaction.	a. User initiates a deposit transaction. b. System updates the account balance and displays a success message.
TransactionController	Test Withdrawal Transaction with Sufficient Balance	Ensure the system updates the account balance and displays a success message for a withdrawal transaction with sufficient balance.	a. User initiates a withdrawal transaction with sufficient balance. b. System updates the account balance and displays a success message.
TransactionController	Test Withdrawal Transaction with Insufficient Balance	Ensure the system displays an error message for a withdrawal transaction with insufficient balance.	a. User initiates a withdrawal transaction with insufficient balance. b. System displays an error message.

TransactionController	Test Transfer Transaction	Ensure the system updates the balances of both accounts and displays a success message for a transfer transaction.	a. User initiates a transfer transaction. b. System updates the balances of both accounts and displays a success message.
LoginController	Test Successful Login	Ensure the system authenticates the user and navigates to the main dashboard for valid credentials.	a. User enters valid credentials. b. System authenticates the user and navigates to the main dashboard.
LoginController	Test Unsuccessful Login with Invalid Credentials	Ensure the system displays an error message for invalid credentials.	a. User enters invalid credentials. b. System displays an error message.
LoginController	Test Unsuccessful Login with Empty Credentials	Ensure the system displays an error message when the username or password field is empty.	a. User leaves the username or password field empty. b. System displays an error message.

5. Conclusion

Summary

The **BankATMManagement system** underwent rigorous and thorough **static** and **dynamic testing**. The following were ensured:

- Comprehensive **code quality improvements** using Qodana.
- **Unit testing** validated methods independently.
- **Integration testing** ensured seamless interaction between modules.
- **System testing** verified the complete workflow.

Key Achievements:

- Fixed major **security vulnerabilities**.
- Improved **code maintainability** by addressing code smells and inefficiencies.
- Achieved 100% **statement and branch test coverage** for critical methods.

Reflection

Improvements Achieved:

- Stronger validation safeguards for user input.
- Exception handling improved system stability.

Future Scope:

- **Stress Testing**:
  - Introduce extensive stress tests under high data loads.
- **Automated Regression Testing**:
  - A robust suite to automatically identify side effects from future changes.
- **Edge Case Expansion**:
  - Include invalid characters, very large strings, and boundary data testing for all methods.