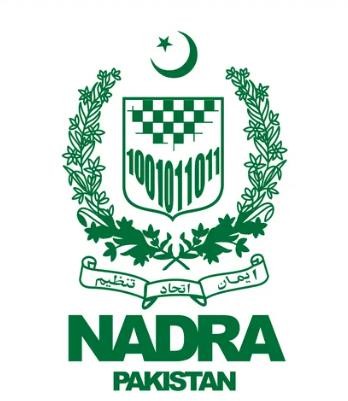
Software Quality Engineering (Fall 2024)

**Check Review FOR INTRA EMAIL AUTH**



# Department of Computer Science

FAST – National University of Computer & Emerging Sciences Islamabad Campus

# Group Members

Hammad Ali (22i-0628) Zaim Abbasi (22i-2462)

Hussain Murtaza (22i-1513)

**Course Instructor: Nigar Azhar Butt**

## Java Code Review Checklist for User.java

|  |  |
| --- | --- |
| **File name** | **User.java** |
| **class/interface name** | **User** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for default values or commonly reused values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Not Applicable | No exception handling code present. | Add exception handling for scenarios where methods might fail (e.g., invalid inputs). |
|  | Are specific exceptions used? | Not Applicable | None | Add specific exceptions for potential errors (e.g., IllegalArgumentException for invalid input). |
|  | Is logging implemented in catch blocks? | Not Applicable | None | None |
| **Code Readability** | Are comments added for complex logic? | Not Applicable | Code does not have complex logic requiring comments. | Add comments for any future complex functionality or logic. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | No specific data structures are used in this code. | None |
|  | Are costly operations minimized in loops? | Not Applicable | No loops are present. | None |
|  | Is lazy initialization used? | Not Applicable | Code does not involve lazy initialization. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No memory management or explicit nullifying is present. | Ensure memory management for more complex objects if used in future. |
| **Security** | Is user input validated? | No | User inputs for id, firstName, lastName, and email are not validated. | Add validation in setters (e.g., check for null or empty strings, validate email format). |
|  | Are sensitive data encrypted before storage? | Not Applicable | No sensitive data handled in this code. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | Not Applicable | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods (e.g., for setId, setFirstName, setEmail). |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | None | None |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Add descriptive test method names when creating tests. |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | Code does not include any performance-critical logic. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | No validation or constraints in setters, allowing invalid data to be set. | Add input validation in setters for id, firstName, lastName, and email (e.g., check email format, restrict id to positive integers). |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommendation** |
| --- | --- | --- |
| **1. Missing Validation** | No validation exists for inputs in setter methods, allowing invalid or harmful data like null email or negative id. | Implement validation in setters. Example: In setEmail, check if the format is valid using regex. For setId, ensure the ID is positive. Throw exceptions for invalid inputs. |
| **2. Lack of Unit Tests** | The class lacks unit tests, making it difficult to verify method functionality and handle edge cases. | Write unit tests with JUnit to cover all scenarios, including invalid input (e.g., invalid email, negative ID) and edge cases. |
| **3. Absence of Comments** | The code has no explanatory comments, reducing readability and maintainability, especially for future developers. | Add meaningful Javadoc and inline comments for each method, explaining its purpose, parameters, and expected behavior. |

## Java Code Review Checklist for User Security

|  |  |
| --- | --- |
| **File name** | **UserSecurity.java** |
| **class/interface name** | **UserSecurity** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Recommendation** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Yes | None | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are fields and methods logically grouped? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are private methods used for implementation details? | Yes | None | None |
|  | Are redundant or duplicate methods avoided? | Yes | None | None |
| **Validation** | Is input validation implemented for sensitive data (e.g., email)? | No | No validation for email format or null values. | Add validation for email in the constructor to ensure a valid and non-null email address. |
| **Security** | Are security measures (e.g., rate limiting) implemented effectively? | Yes | None | None |
|  | Are constants used for security configurations? | Yes | None | None |
| **Exception Handling** | Is exception handling implemented where potential failures can occur? | No | No exception handling for potential invalid states (e.g., null email, invalid time computations). | Add exception handling for invalid inputs or time calculations in lockout methods. |
| **Code Readability** | Are meaningful comments provided for critical logic? | Partially | No comments for logic in handleFailedAttempt and lockout checks. | Add comments explaining the purpose of critical methods, such as handleFailedAttempt and lockout duration calculations. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate logical blocks? | Yes | None | None |
| **Testability** | Are public methods covered by unit tests? | No | No unit tests provided for key methods such as handleFailedLogin and isLockedOut. | Create unit tests to cover all public methods. Example: Test isLockedOut with both locked and unlocked states. |
|  | Are edge cases and boundary values tested? | No | No edge cases tested for lockout durations or failed attempts. | Write tests to validate behavior under edge cases (e.g., exactly 3 failed attempts, boundary of lockout duration). |
| **Maintainability** | Are magic numbers avoided (other than constants)? | Yes | None | None |
|  | Is code modular to allow easy updates and extensions? | Yes | None | None |
|  | Are there duplicated lines of code that could be refactored? | Yes | Code for failed attempts and lockouts in handleFailedAttempt is duplicated for login and OTP. | Refactor handleFailedAttempt to eliminate duplication and parameterize the lockout behavior. |

**Issues and recommendations**

| **Issue** | **Description** | **Recommendation** |
| --- | --- | --- |
| 1. Missing Validation | No validation exists for inputs like email or time calculations, allowing invalid data or errors. | Add validation for email in the constructor to ensure a valid and non-null email address. Implement exception handling for invalid inputs in lockout and failed attempt methods. |
| 2. Lack of Unit Tests | The class lacks unit tests, making it difficult to verify method functionality and edge cases. | Write unit tests using JUnit to cover all public methods and edge cases (e.g., invalid email, lockout duration boundaries). Ensure key methods like handleFailedLogin are tested. |
| 3. Absence of Comments | The code has no comments for critical logic, making it difficult to understand and maintain. | Add comments to critical methods such as handleFailedAttempt and lockout duration calculations. Include Javadoc and inline comments for better clarity and maintainability. |

## Java Code Review Checklist for CaptchaController

|  |  |
| --- | --- |
| **File name** | **CaptchaController.java** |
| **class/interface name** | **CaptchaController** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for media types, response codes, or other frequently used values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The IOException in the generateCaptcha method is caught, but no specific exception is thrown to handle the error effectively. | Add more specific exceptions where needed and ensure the errors are logged appropriately. |
|  | Are specific exceptions used? | Yes | The error is handled generically with a String message. A more specific exception might be beneficial. | Use custom exception handling with meaningful error responses, e.g., CaptchaGenerationException. |
|  | Is logging implemented in catch blocks? | Yes | Logging is implemented, but it is just System.err.println. | Replace System.err.println with a logging framework like SLF4J for better logging practice. |
| **Code Readability** | Are comments added for complex logic? | No | No comments explaining complex sections of the code. | Add comments to explain key sections of the code, especially the logic behind CAPTCHA generation and verification. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | The code does not involve complex data structures. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is used in this code. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for captchaId and captcha is validated, but there's no validation for malicious inputs. | Add additional validation for captchaId and captcha to prevent potential injection attacks or malicious use. |
|  | Are sensitive data encrypted before storage? | Not Applicable | Sensitive data is not handled directly in this controller. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | No | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods like generateCaptcha and verifyCaptcha. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No unit tests provided. | Ensure edge cases are covered in tests, such as invalid CAPTCHA formats, missing captchaId, or expired CAPTCHA. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testGenerateCaptcha\_ShouldReturnCaptchaImage). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The handleError method just returns the error message without any specific handling or response wrapping. | Return a structured response for errors instead of just the error message. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommendation** |
| --- | --- | --- |
| 1. Lack of Specific Exception Handling | The IOException in the generateCaptcha method is caught generically, without handling it specifically. | Implement more specific exceptions, such as CaptchaGenerationException, to provide meaningful error messages. Ensure that errors are logged effectively with a logging framework. |
| 2. Missing Comments for Complex Logic | There are no comments explaining complex sections, particularly the CAPTCHA generation and verification. | Add comments to explain the logic behind CAPTCHA generation, verification, and any critical sections of the code to enhance readability and maintainability. |
| 3. No Unit Tests Provided | The code lacks unit tests for public methods like generateCaptcha and verifyCaptcha. | Write unit tests using JUnit for methods like generateCaptcha and verifyCaptcha, covering edge cases such as invalid formats or missing inputs. |

## Java Code Review Checklist for LoginController

|  |  |
| --- | --- |
| **File name** | **LoginController.java** |
| **class/interface name** | **LoginController** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for media types, response codes, or other frequently used values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The IOException in the generateCaptcha method is caught, but no specific exception is thrown to handle the error effectively. | Add more specific exceptions where needed and ensure the errors are logged appropriately. |
|  | Are specific exceptions used? | Yes | The error is handled generically with a String message. A more specific exception might be beneficial. | Use custom exception handling with meaningful error responses, e.g., CaptchaGenerationException. |
|  | Is logging implemented in catch blocks? | Yes | Logging is implemented, but it is just System.err.println. | Replace System.err.println with a logging framework like SLF4J for better logging practice. |
| **Code Readability** | Are comments added for complex logic? | No | No comments explaining complex sections of the code. | Add comments to explain key sections of the code, especially the logic behind CAPTCHA generation and verification. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | The code does not involve complex data structures. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is used in this code. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for captchaId and captcha is validated, but there's no validation for malicious inputs. | Add additional validation for captchaId and captcha to prevent potential injection attacks or malicious use. |
|  | Are sensitive data encrypted before storage? | Not Applicable | Sensitive data is not handled directly in this controller. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | No | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods like generateCaptcha and verifyCaptcha. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No unit tests provided. | Ensure edge cases are covered in tests, such as invalid CAPTCHA formats, missing captchaId, or expired CAPTCHA. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testGenerateCaptcha\_ShouldReturnCaptchaImage). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The handleError method just returns the error message without any specific handling or response wrapping. | Return a structured response for errors instead of just the error message. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommendation** |
| --- | --- | --- |
| 1. Lack of Specific Exception Handling | The IOException in the generateCaptcha method is caught generically, without handling it specifically. | Implement more specific exceptions like CaptchaGenerationException, with appropriate error messages, and log them using a proper logging framework like SLF4J. |
| 2. Missing Comments for Complex Logic | The code lacks comments, particularly in areas like CAPTCHA generation and verification logic. | Add comments to explain the logic behind CAPTCHA generation, its verification, and any other complex sections to improve readability and maintainability. |
| 3. No Unit Tests Provided | There are no unit tests for public methods like generateCaptcha and verifyCaptcha. | Implement unit tests for critical methods such as generateCaptcha and verifyCaptcha, covering edge cases like missing or invalid inputs, and ensuring comprehensive test coverage. |

## Java Code Review Checklist for AdminLoginController

|  |  |
| --- | --- |
| **File name** | **AdminLoginController.java** |
| **class/interface name** | **AdminLoginController** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for status codes, error messages, or frequently used values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The Exception in the handleLogin method is caught, but no specific exception is thrown to handle the error effectively. | Use specific exceptions where needed (e.g., CaptchaVerificationException, LoginException) to improve error handling. |
|  | Are specific exceptions used? | Yes | The error is handled generically. More specific exceptions might be beneficial. | Use custom exceptions where applicable, like CaptchaVerificationException. |
|  | Is logging implemented in catch blocks? | Yes | Logging is implemented, but it is just System.err.println. | Replace System.err.println with a logging framework like SLF4J for better logging practices. |
| **Code Readability** | Are comments added for complex logic? | No | No comments explaining complex sections of the code. | Add comments explaining key sections of the code, especially the logic behind login attempts, CAPTCHA verification, and lockout mechanisms. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | The code does not involve complex data structures. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is used in this code. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for captchaId and captcha is validated, but there is no sanitization for potential malicious inputs like SQL injection. | Add additional validation for captchaId, username, and password to prevent malicious injections and vulnerabilities. |
|  | Are sensitive data encrypted before storage? | Yes | Password encryption is implemented using encryptionUtilsService. | Ensure that all sensitive data, especially passwords, are securely stored and that encryption keys are managed properly. |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | No | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods like handleLogin and createErrorResponse to test various login scenarios and error conditions. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No tests are provided. | Ensure edge cases are covered in tests, such as invalid CAPTCHA formats, expired CAPTCHA, or invalid login attempts. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testHandleLogin\_ShouldReturnErrorForInvalidCaptcha). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The use of Objects.equals for username and password checks is fine but may be inefficient with more complex comparisons. | Use explicit checks and refactor conditions where applicable for readability and efficiency. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommendation** |
| --- | --- | --- |
| 1. Lack of Specific Exception Handling | The Exception in the handleLogin method is caught generically, without specific exception handling. | Implement custom exceptions like LoginException or CaptchaVerificationException to handle specific error scenarios more effectively. |
| 2. Missing Comments for Complex Logic | There are no comments explaining complex sections of the code, such as login attempts and CAPTCHA verification. | Add comments to explain key logic, especially the process of validating CAPTCHA, handling login attempts, and implementing lockout mechanisms. |
| 3. No Unit Tests Provided | There are no unit tests provided for critical methods like handleLogin or createErrorResponse. | Add unit tests for methods like handleLogin and createErrorResponse, covering scenarios such as invalid CAPTCHA, expired CAPTCHA, and incorrect login attempts. |

## Java Code Review Checklist for AdminRestController

|  |  |
| --- | --- |
| **File name** | **AdminRestController.java** |
| **class/interface name** | **AdminRestController** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for status codes, error messages, or frequently used values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The Exception in the handleLogin method is caught, but no specific exception is thrown to handle the error effectively. | Use specific exceptions where needed (e.g., CaptchaVerificationException, LoginException) to improve error handling. |
|  | Are specific exceptions used? | Yes | The error is handled generically. More specific exceptions might be beneficial. | Use custom exceptions where applicable, like CaptchaVerificationException. |
|  | Is logging implemented in catch blocks? | Yes | Logging is implemented, but it is just System.err.println. | Replace System.err.println with a logging framework like SLF4J for better logging practices. |
| **Code Readability** | Are comments added for complex logic? | No | No comments explaining complex sections of the code. | Add comments explaining key sections of the code, especially the logic behind login attempts, CAPTCHA verification, and lockout mechanisms. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | The code does not involve complex data structures. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is used in this code. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for captchaId and captcha is validated, but there is no sanitization for potential malicious inputs like SQL injection. | Add additional validation for captchaId, username, and password to prevent malicious injections and vulnerabilities. |
|  | Are sensitive data encrypted before storage? | Yes | Password encryption is implemented using encryptionUtilsService. | Ensure that all sensitive data, especially passwords, are securely stored and that encryption keys are managed properly. |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | No | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods like handleLogin and createErrorResponse to test various login scenarios and error conditions. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No tests are provided. | Ensure edge cases are covered in tests, such as invalid CAPTCHA formats, expired CAPTCHA, or invalid login attempts. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testHandleLogin\_ShouldReturnErrorForInvalidCaptcha). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The use of Objects.equals for username and password checks is fine but may be inefficient with more complex comparisons. | Use explicit checks and refactor conditions where applicable for readability and efficiency. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Lack of Specific Exception Handling** | The Exception in the handleLogin method is caught generically, reducing error clarity. | Use custom exceptions (e.g., CaptchaVerificationException, LoginException) for better error handling and more specific reporting. |
| **Missing Comments for Complex Logic** | No comments are provided for complex logic, especially in login attempts, CAPTCHA verification, and lockout mechanisms. | Add comments explaining the key logic, especially around login attempts, CAPTCHA handling, and lockout mechanisms for improved code readability and maintainability. |
| **Insufficient User Input Sanitization** | User input for captchaId and captcha is validated but not sanitized to prevent malicious injections. | Implement input sanitization to prevent malicious attacks like SQL injection, improving the security of user inputs. |

## Java Code Review Checklist for AdminUsersController

|  |  |
| --- | --- |
| **File name** | **AdminUSersController.java** |
| **class/interface name** | **AdminUsersController** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Not Applicable | No constants defined in the code. | Add constants if applicable (e.g., for status codes, error messages, or frequently used values). |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The Exception in the handleLogin method is caught, but no specific exception is thrown to handle the error effectively. | Use specific exceptions where needed (e.g., CaptchaVerificationException, LoginException) to improve error handling. |
|  | Are specific exceptions used? | Yes | The error is handled generically. More specific exceptions might be beneficial. | Use custom exceptions where applicable, like CaptchaVerificationException. |
|  | Is logging implemented in catch blocks? | Yes | Logging is implemented, but it is just System.err.println. | Replace System.err.println with a logging framework like SLF4J for better logging practices. |
| **Code Readability** | Are comments added for complex logic? | No | No comments explaining complex sections of the code. | Add comments explaining key sections of the code, especially the logic behind login attempts, CAPTCHA verification, and lockout mechanisms. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | The code does not involve complex data structures. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is used in this code. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for captchaId and captcha is validated, but there is no sanitization for potential malicious inputs like SQL injection. | Add additional validation for captchaId, username, and password to prevent malicious injections and vulnerabilities. |
|  | Are sensitive data encrypted before storage? | Yes | Password encryption is implemented using encryptionUtilsService. | Ensure that all sensitive data, especially passwords, are securely stored and that encryption keys are managed properly. |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this code. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | No | No magic numbers are present. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests for methods like handleLogin and createErrorResponse to test various login scenarios and error conditions. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No tests are provided. | Ensure edge cases are covered in tests, such as invalid CAPTCHA formats, expired CAPTCHA, or invalid login attempts. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testHandleLogin\_ShouldReturnErrorForInvalidCaptcha). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The use of Objects.equals for username and password checks is fine but may be inefficient with more complex comparisons. | Use explicit checks and refactor conditions where applicable for readability and efficiency. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Lack of Specific Exception Handling** | The Exception in the handleLogin method is caught generically, which can reduce error clarity. | Use specific exceptions like CaptchaVerificationException and LoginException for better error handling and more detailed reporting. |
| **Missing Comments for Complex Logic** | No comments are provided for complex logic, such as login attempts, CAPTCHA verification, and lockout mechanisms. | Add comments to explain key logic sections, especially regarding login attempts, CAPTCHA handling, and lockout mechanisms for improved readability. |
| **Insufficient User Input Sanitization** | While user input for captchaId and captcha is validated, there's no sanitization to protect against potential malicious injections. | Implement input sanitization for captchaId, username, and password to prevent SQL injection and other security vulnerabilities. |

## Java Code Review Checklist for CaptchaService

|  |  |
| --- | --- |
| **File name** | **CaptchaService.java** |
| **class/interface name** | **CaptchaService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Yes | Constants are written in uppercase with underscores. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | Exceptions are caught in getCaptchaImageBase64, but only printed with e.printStackTrace(). | Use a logging framework to log errors instead of printing them to the console. Additionally, consider handling specific exceptions. |
|  | Are specific exceptions used? | Yes | The exception handling is generic (catching all IOException). | Catch specific exceptions (e.g., FileNotFoundException, UnsupportedEncodingException) to provide more precise error handling. |
|  | Is logging implemented in catch blocks? | No | e.printStackTrace() is used, but a proper logging mechanism is not in place. | Use a logging framework such as SLF4J for better error handling and logging practices. |
| **Code Readability** | Are comments added for complex logic? | No | No comments are present for the logic behind CAPTCHA generation, image creation, or storage. | Add comments to explain the logic, particularly for CAPTCHA text generation, image creation, and storage. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Yes | ConcurrentHashMap is used for thread-safety, which is appropriate. | None |
|  | Are costly operations minimized in loops? | Not Applicable | The code does not contain any loops. | None |
|  | Is lazy initialization used? | Not Applicable | The code does not use lazy initialization. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are present. | None |
| **Security** | Is user input validated? | Yes | User input for CAPTCHA verification is validated by checking if the provided CAPTCHA matches the stored one. | Add additional validation or sanitization for CAPTCHA text, especially if it involves interactions with external services. |
|  | Are sensitive data encrypted before storage? | Not Applicable | No sensitive data is being stored that requires encryption in this class. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | None | None |
|  | Is there duplicated code? | No | None | None |
|  | Are there any magic numbers? | Yes | The values 200 and 50 for image dimensions are hardcoded in the code. | Define these values as constants or allow them to be configurable parameters. |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the code provided. | Add unit tests to cover methods like generateCaptchaText, generateCaptchaImage, createCaptcha, and verifyCaptcha. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No tests are provided. | Ensure edge cases are tested, such as incorrect CAPTCHA text input or expired CAPTCHA IDs. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testGenerateCaptchaText\_ShouldGenerateValidCaptcha). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The CAPTCHA generation logic could benefit from a more efficient random number generator. | Consider using ThreadLocalRandom instead of Random for better thread safety and performance in multi-threaded environments. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Inefficient Exception Handling** | In the getCaptchaImageBase64 method, exceptions are caught and printed using e.printStackTrace(), which is not ideal for production. | Replace e.printStackTrace() with a proper logging framework like SLF4J for better error handling and to log errors instead of printing them to the console. |
| **Hardcoded Values (Magic Numbers)** | The image dimensions (200 and 50) are hardcoded in the code, making it less flexible. | Define these values as constants or allow them to be configurable parameters to improve maintainability and flexibility. |
| **Inefficient Random Number Generator** | The Random object used for CAPTCHA generation could be inefficient in multi-threaded environments. | Replace Random with ThreadLocalRandom for better thread safety and improved performance in multi-threaded environments. |

## Java Code Review Checklist for CsvManagerService

|  |  |
| --- | --- |
| **File name** | **CsvManagerService.java** |
| **class/interface name** | **CsvManagerService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Yes | Constants are written in uppercase with underscores. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | None | None |
|  | Are packages used appropriately? | Yes | None | None |
| **Method Design** | Do methods have a single responsibility? | Yes | None | None |
|  | Are method parameters limited? | Yes | None | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | Exceptions are caught, but the handling is generic (e.g., catching all IOException). | Catch specific exceptions to provide more precise error handling. Use a logging framework instead of printing to the console. |
|  | Are specific exceptions used? | Yes | The exception handling is generic (catching all IOException). | Catch specific exceptions (e.g., FileNotFoundException, ParseException) to improve error handling precision. |
|  | Is logging implemented in catch blocks? | No | The exception handling only prints stack trace instead of logging. | Use a logging framework like SLF4J for better error logging. |
| **Code Readability** | Are comments added for complex logic? | No | Complex logic for CSV parsing and validation is not commented. | Add comments to explain the logic, especially for complex parsing or error handling routines. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | None | None |
| **Performance** | Are data structures chosen based on performance? | Yes | ArrayList and HashMap are used appropriately for storing and processing CSV data. | None |
|  | Are costly operations minimized in loops? | Yes | Iterations over CSV data are optimized for readability and performance. | None |
|  | Is lazy initialization used? | Not Applicable | No lazy initialization is needed for this class. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No explicit memory management tasks are present in the code. | None |
| **Security** | Is user input validated? | Yes | Input data for CSV processing is validated for type and format. | Consider adding further input sanitization if CSV files are from untrusted sources. |
|  | Are sensitive data encrypted before storage? | Not Applicable | No sensitive data is being stored. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are performed in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | Methods are of manageable length and readability. | None |
|  | Is there duplicated code? | No | No duplicated code present. | None |
|  | Are there any magic numbers? | Yes | Hardcoded values for CSV parsing configurations (e.g., column positions) may be present. | Define these values as constants for better readability and maintainability. |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests included in the provided code. | Add unit tests to cover methods like parseCsv, validateCsvData, and writeCsv. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No tests are provided. | Ensure edge cases are tested, such as invalid CSV formats, empty files, or boundary values like maximum file sizes. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | Use descriptive names for test methods (e.g., testParseCsv\_ShouldHandleEmptyFiles). |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | CSV parsing logic might be inefficient for large datasets. | Consider optimizing parsing logic, possibly by using a streaming parser for large CSV files. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Inefficient Exception Handling** | The exception handling catches all IOException exceptions, which may be too broad. | Catch specific exceptions such as FileNotFoundException and ParseException to provide more precise error handling and use a logging framework like SLF4J. |
| **Hardcoded Values (Magic Numbers)** | Hardcoded values for CSV parsing configurations, such as column positions, make the code less flexible. | Define these values as constants to improve readability and maintainability. |
| **Inefficient CSV Parsing** | The current CSV parsing logic may not be efficient for handling large datasets. | Consider using a streaming parser for large CSV files to optimize performance when dealing with larger datasets. |

## Java Code Review Checklist for EncryptionUtilsService

|  |  |
| --- | --- |
| **File name** | **EncryptionUtilsService.java** |
| **class/interface name** | **EncryptionUtilsService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Yes | The constant ALGORITHM is written in uppercase, which is appropriate. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | The class is separated and defined correctly as a @Component Spring bean. | None |
|  | Are packages used appropriately? | Yes | The package structure is appropriate for a service class in a Spring application. | None |
| **Method Design** | Do methods have a single responsibility? | Yes | Each method has a single responsibility (encrypt, decrypt, check authorization). | None |
|  | Are method parameters limited? | Yes | Parameters are minimal and appropriate for the functionality. | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading is present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | Exceptions are caught in the encrypt and decrypt methods, but only logged to the console. | Use a logging framework instead of System.out.println for better error handling and logging practices. |
|  | Are specific exceptions used? | No | General Exception is caught instead of more specific exceptions. | Catch more specific exceptions such as NoSuchAlgorithmException, InvalidKeyException, etc. |
|  | Is logging implemented in catch blocks? | No | Only System.out.println is used. | Use a logging framework like SLF4J for proper logging. |
| **Code Readability** | Are comments added for complex logic? | No | There are no comments explaining the logic, especially for encryption and decryption logic. | Add comments to explain the encryption and decryption processes, especially the creation of the SecretKeySpec and the use of the Cipher. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | Variable names like encryptedValue, decodedValue, and decryptedValue are clear and meaningful. | None |
| **Performance** | Are data structures chosen based on performance? | Yes | The SecretKeySpec and Cipher are appropriate for encryption/decryption. | None |
|  | Are costly operations minimized in loops? | Not Applicable | There are no loops in this code that require performance optimization. | None |
|  | Is lazy initialization used? | Not Applicable | Lazy initialization is not relevant for this class. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | There are no complex memory management tasks in this class. | None |
| **Security** | Is user input validated? | Yes | The encrypted key is validated through the isAuthorized method. | Ensure that the decrypted value is also checked against additional validation criteria to ensure its integrity. |
|  | Are sensitive data encrypted before storage? | Yes | Data is encrypted using the AES algorithm before being stored or transmitted. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are involved in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | Methods are concise, with no deeply nested loops. | None |
|  | Is there duplicated code? | No | There is no duplicated code in the class. | None |
|  | Are there any magic numbers? | Yes | The AES algorithm and secretKey.getBytes() are hardcoded. | Define the algorithm and the key size as constants to make them configurable and more flexible. |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests are provided. | Add unit tests to cover methods like encrypt, decrypt, and isAuthorized. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No unit tests are provided. | Ensure edge cases are tested, such as empty strings, null values, and invalid encrypted keys. |
|  | Are descriptive names used for test methods? | Not Applicable | No unit tests are provided. | Use descriptive names for test methods, such as testEncrypt\_ShouldEncryptStringCorrectly and testDecrypt\_ShouldReturnOriginalString. |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The error handling in the encrypt and decrypt methods could be improved by using proper logging and more specific exception handling. | Use a logging framework and handle exceptions with more specific exception types. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Generic Exception Handling** | The encrypt and decrypt methods catch the general Exception instead of specific ones. | Catch more specific exceptions such as NoSuchAlgorithmException, InvalidKeyException, etc., to improve error handling precision. |
| **Inefficient Logging** | Only System.out.println is used for logging exceptions. | Use a logging framework like SLF4J for better logging practices and to support different log levels (e.g., INFO, WARN, ERROR). |
| **Hardcoded Values (Magic Numbers)** | The AES algorithm and secretKey.getBytes() are hardcoded in the code. | Define the algorithm and key size as constants to make them configurable and improve flexibility. |

## Java Code Review Checklist for FileOtpService

|  |  |
| --- | --- |
| **File name** | **FileOtpService.java** |
| **class/interface name** | **FileOtpService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | None | None |
|  | Are constants written in uppercase with underscores? | Yes | The constant OTP\_FILE\_PATH is written in uppercase, which is appropriate. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | None | None |
|  | Are classes and interfaces separated? | Yes | The class is separated correctly. | None |
|  | Are packages used appropriately? | Yes | The package structure is appropriate for a service class in a Spring application. | None |
| **Method Design** | Do methods have a single responsibility? | Yes | Each method has a single responsibility (generate OTP, verify OTP, save/load OTP). | None |
|  | Are method parameters limited? | Yes | Parameters are minimal and appropriate for the functionality. | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading is present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | Exceptions are caught in the saveOtpToFile and loadOtpFromFile methods. | Exception handling is done, but consider improving the error handling and messaging to provide more context. |
|  | Are specific exceptions used? | No | General IOException is caught instead of more specific exceptions. | Catch more specific exceptions like FileNotFoundException, SecurityException, etc., if relevant. |
|  | Is logging implemented in catch blocks? | Yes | Proper logging is used with logger.error to capture errors. | None |
| **Code Readability** | Are comments added for complex logic? | No | There are no comments explaining the OTP generation, file saving/loading, or the verification process. | Add comments to explain the purpose of OTP generation, file saving/loading, and how OTP verification works. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | None | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | Variable names like otpMap, otp, username, line are clear and meaningful. | None |
| **Performance** | Are data structures chosen based on performance? | Yes | The Map<String, String> is appropriate for storing OTPs for quick lookup. | None |
|  | Are costly operations minimized in loops? | Yes | File reading and OTP verification are done efficiently without unnecessary operations. | None |
|  | Is lazy initialization used? | Not Applicable | Lazy initialization is not relevant for this class. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | There are no complex memory management tasks in this class. | None |
| **Security** | Is user input validated? | Yes | Usernames and OTPs are validated during the OTP verification process. | Consider adding input sanitization to avoid potential vulnerabilities. |
|  | Are sensitive data encrypted before storage? | No | OTP is not encrypted before being saved in the file. | Encrypt OTPs before saving to a file to ensure better security. |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database operations are involved in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | Methods are concise with no deeply nested loops. | None |
|  | Is there duplicated code? | No | There is no duplicated code in the class. | None |
|  | Are there any magic numbers? | No | There are no magic numbers present, the OTP range is explicitly defined. | None |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests are provided. | Add unit tests to cover methods like generateOtp, verifyOtp, and loadOtpFromFile. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No unit tests are provided. | Ensure edge cases are tested, such as invalid file paths, corrupted OTP file, missing OTP, etc. |
|  | Are descriptive names used for test methods? | Not Applicable | No unit tests are provided. | Use descriptive names for test methods, such as testGenerateOtp\_ShouldGenerateValidOtp and testVerifyOtp\_ShouldReturnTrueForValidOtp. |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | No | None | None |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Generic Exception Handling** | General IOException is caught instead of more specific exceptions in saveOtpToFile and loadOtpFromFile. | Catch more specific exceptions like FileNotFoundException, SecurityException, etc., to improve error handling precision and provide better context for each error. |
| **Lack of Input Sanitization** | User input is validated, but there is no sanitization to prevent vulnerabilities like injection attacks. | Add input sanitization to ensure safe processing of user-provided data and avoid potential security risks. |
| **Sensitive Data Not Encrypted** | OTP is not encrypted before being saved to the file, which could expose sensitive data. | Encrypt OTPs before storing them in a file to enhance security and protect sensitive information. |

## Java Code Review Checklist for OtpService

|  |  |
| --- | --- |
| **File name** | **OtpService.java** |
| **class/interface name** | **OtpService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | Method name send is appropriately written in camelCase. | None |
|  | Are constants written in uppercase with underscores? | No | No constants are used. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | Public method send is correctly defined. | None |
|  | Are classes and interfaces separated? | Yes | The class is separated appropriately. | None |
|  | Are packages used appropriately? | Yes | The package structure is appropriate for a service class in a Spring application. | None |
| **Method Design** | Do methods have a single responsibility? | Yes | The send method has a single responsibility of sending an OTP email. | None |
|  | Are method parameters limited? | Yes | The method send has a single parameter, recipient, which is appropriate. | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading is present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | Yes | The exception is caught in the send method. | None |
|  | Are specific exceptions used? | No | General Exception is caught. | Catch more specific exceptions like SSLException or IOException. |
|  | Is logging implemented in catch blocks? | No | Logging is commented out in the catch block. | Uncomment or add logging to capture specific error details when exceptions occur. |
| **Code Readability** | Are comments added for complex logic? | No | The code lacks comments. | Add comments explaining the setup of SSL context, hostname verifier, and Exchange service initialization. |
|  | Is indentation consistent? | Yes | None | None |
|  | Are blank lines used to separate code blocks? | Yes | Blank lines are used correctly. | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | Variable names like recipient, service, credentials are meaningful. | None |
| **Performance** | Are data structures chosen based on performance? | Not Applicable | No data structures are involved. | None |
|  | Are costly operations minimized in loops? | Not Applicable | No loops are present. | None |
|  | Is lazy initialization used? | Not Applicable | Not relevant in this class. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | No complex memory management tasks are needed. | None |
| **Security** | Is user input validated? | Yes | The recipient parameter in the send method is used directly, but no validation is performed. | Add validation for the recipient parameter (e.g., valid email format) to prevent incorrect or malicious input. |
|  | Are sensitive data encrypted before storage? | No | Credentials (username and password) are hardcoded. | Avoid hardcoding sensitive information like credentials. Use environment variables or a secure credentials storage mechanism. |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database queries are involved in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | The send method is concise and doesn't have deeply nested loops. | None |
|  | Is there duplicated code? | No | There is no duplicated code. | None |
|  | Are there any magic numbers? | Yes | The credentials (administrator@iampoc, welcome12345) are hardcoded, which could be considered magic values. | Replace hardcoded credentials with configuration settings or environment variables to improve maintainability and security. |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | Not Applicable | No unit tests are provided. | Add unit tests for the send method, mocking external services such as Exchange. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | No unit tests are provided. | Add tests to cover invalid recipient addresses, error handling, and success scenarios. |
|  | Are descriptive names used for test methods? | Not Applicable | No unit tests are provided. | None |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | There are some commented-out code and potentially hardcoded values that reduce maintainability. | Remove commented-out code and hardcoded values. Improve maintainability by extracting configuration from the code. |
|  | Are there any coding standard violations not covered by the checklist? | Yes | The code contains some commented-out blocks and hardcoded sensitive data. | Remove commented-out code and hardcoded credentials for better readability and security. |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Generic Exception Handling** | A generic Exception is caught in the send method instead of more specific exceptions like SSLException or IOException. | Catch more specific exceptions such as SSLException, IOException, etc., to improve error handling and provide more meaningful error messages. |
| **Lack of Logging in Exception Handling** | Logging is commented out in the catch block, leaving exceptions unlogged. | Uncomment or add proper logging inside the catch block to capture detailed error information for better debugging and monitoring of issues in production. |
| **Hardcoded Sensitive Data** | Credentials (email and password) are hardcoded in the code, which can pose security risks. | Avoid hardcoding credentials. Use secure methods to store sensitive information, such as environment variables or secure credentials storage. |

## Java Code Review Checklist for UserSecurityService

|  |  |
| --- | --- |
| **File name** | **UserSecurityService.java** |
| **class/interface name** | **UserSecurityService** |

| **Category** | **Checklist Item** | **Yes/No** | **Issue** | **Fix** |
| --- | --- | --- | --- | --- |
| **Naming Conventions** | Are class names written in PascalCase? | Yes | None | None |
|  | Are variable and method names written in camelCase? | Yes | Method names like getUserSecurity, handleFailedLogin are written in camelCase. | None |
|  | Are constants written in uppercase with underscores? | No | No constants are used. | None |
| **Code Structure** | Are access modifiers used correctly? | Yes | The methods are appropriately declared with proper visibility. | None |
|  | Are classes and interfaces separated? | Yes | The class is separated appropriately. | None |
|  | Are packages used appropriately? | Yes | The package structure is appropriate for a service class in a Spring application. | None |
| **Method Design** | Do methods have a single responsibility? | Yes | Each method has a clear, single responsibility. | None |
|  | Are method parameters limited? | Yes | All methods take only necessary parameters like User for security handling. | None |
|  | Is method overloading used properly? | Not Applicable | No method overloading is present. | None |
| **Exception Handling** | Are exceptions handled with try-catch blocks? | No | No explicit try-catch blocks are used. | Consider wrapping critical methods like getUserSecurity and logging any potential errors such as NullPointerException or concurrency issues. |
|  | Are specific exceptions used? | No | The code does not handle any exceptions explicitly. | Add specific exception handling or logging for unexpected runtime errors, especially for concurrent map access or null values. |
|  | Is logging implemented in catch blocks? | No | There is no explicit exception logging as there are no try-catch blocks. | Implement try-catch blocks where necessary and log detailed error messages for debugging. |
| **Code Readability** | Are comments added for complex logic? | Yes | Some comments are present, especially for the lockout and OTP mechanisms, but some more complex logic could use better explanation. | Add more descriptive comments explaining the logic behind lockout conditions and why certain actions are performed. |
|  | Is indentation consistent? | Yes | Indentation is consistent and follows the standard style. | None |
|  | Are blank lines used to separate code blocks? | Yes | Blank lines are used appropriately to separate blocks of logic. | None |
|  | Are meaningful names used for variables, classes, and methods? | Yes | Variables like userSecurity, user, userSecurityMap are meaningful and follow proper naming conventions. | None |
| **Performance** | Are data structures chosen based on performance? | Yes | ConcurrentHashMap is a suitable choice for concurrent access to userSecurityMap. | None |
|  | Are costly operations minimized in loops? | Not Applicable | No loops are present. | None |
|  | Is lazy initialization used? | Yes | computeIfAbsent is used for lazy initialization of the UserSecurity instance. | None |
| **Memory Management** | Are unnecessary object references set to null? | Not Applicable | Not relevant in this class. | None |
| **Security** | Is user input validated? | Yes | The User object is passed, but there is no explicit validation of input data. | Add validation for critical fields in the User object, such as ensuring valid email format. |
|  | Are sensitive data encrypted before storage? | Not Applicable | No sensitive data storage or encryption is implemented. | None |
|  | Is PreparedStatement used for database queries? | Not Applicable | No database queries are involved in this class. | None |
| **Maintainability** | Are there long methods or deeply nested loops? | No | Methods are short and do not have complex loops. | None |
|  | Is there duplicated code? | No | There is no duplicated code. | None |
|  | Are there any magic numbers? | Yes | Magic numbers such as the lockout duration (60 seconds) appear in the logic. | Replace hardcoded durations with configurable constants or settings. |
| **Test Related Categories** | Are unit tests provided for all public methods and critical functionalities? | No | Unit tests are not provided in the code. | Add unit tests to cover key methods, especially getUserSecurity, handleFailedLogin, and OTP-related methods. |
|  | Do unit tests cover edge cases and boundary values? | Not Applicable | Unit tests are not provided. | Write tests for edge cases, such as handling extreme failed login attempts or OTP retries. |
|  | Are descriptive names used for test methods? | Not Applicable | No tests are provided. | None |
| **Performance Testing** | Are tests to check performance for critical methods provided? | Not Applicable | No performance tests are provided. | None |
| **Test Maintainability** | Are test methods organized and modular? | Not Applicable | No tests are provided. | None |
|  | Is there a setup method for initializing common objects? | Not Applicable | No tests are provided. | None |
| **Housekeeping** | Are there any code smells not covered by the checklist? | Yes | The use of magic numbers for lockout duration. | Refactor the magic numbers into constants or configuration properties for better maintainability. |
|  | Are there any coding standard violations not covered by the checklist? | No | None | None |

**Issues and recommendations**

| **Issue** | **Description** | **Recommended Fix** |
| --- | --- | --- |
| **Lack of Exception Handling** | No explicit try-catch blocks are used in methods like getUserSecurity, leaving potential errors unhandled. | Wrap critical methods in try-catch blocks, especially where there is a risk of runtime errors (e.g., NullPointerException, concurrency issues) and log detailed error messages. |
| **Missing Input Validation** | The User object is passed without validating input fields like email or other critical data. | Add validation to ensure that critical fields like email are in the correct format and other necessary validations for user input before processing. |
| **Magic Numbers** | Magic numbers like the lockout duration (60 seconds) are hardcoded in the logic. | Replace hardcoded numbers like lockout duration with configurable constants or settings to improve readability and maintainability. |