

***Submitted to: Dr. Safaa Hriez***

***Submitted by: Hamza AL-Risheq***

***Secure Coding***

***HealthClinicPro***

Table of Contents

[1. Security Principles 1](#_Toc156802196)

[1.1. Secure the Weakest Link 1](#_Toc156802197)

[1.2. Fail Securely 2](#_Toc156802198)

[2. Quality Principles 2](#_Toc156802199)

[2.1. Flexibility and Reusability 2](#_Toc156802200)

[2.2. Understandability 3](#_Toc156802201)

[3. Use Case and Misuse Case Diagram for HealthcarePro Application 4](#_Toc156802202)

[4. Data Flow Diagram for HealthcarePro Application 8](#_Toc156802203)

[7. Type of Testing methods Used in HealthcarePro Application 8](#_Toc156802204)

[8. Type of Security Testing methods Used in HealthcarePro Application 10](#_Toc156802205)

[9. HealthcarePro Application Test Cases 12](#_Toc156802206)

[9.1. Employee Login Test Cases 12](#_Toc156802207)

[9.2. Patient Login Test Cases 13](#_Toc156802208)

[9.3. Enter New Medical Report Test Cases 15](#_Toc156802209)

[12. Security-Based Testing Techniques 17](#_Toc156802210)

[11. Code Review Analysis 17](#_Toc156802211)

[11. Code Versioning 17](#_Toc156802212)

[Screen Shots: 17](#_Toc156802213)

[GitHub Link: 17](#_Toc156802214)

[References 18](#_Toc156802215)

# 1. Security Principles

## 1.1. Secure the Weakest Link

The "Secure the Weakest Link" logic emphasizes the critical effort of identifying and securing an application's most vulnerable aspects. This principle emphasizes the importance of addressing possible weaknesses throughout the whole software development lifecycle, from inspiration to maintenance, in the same way that the strength of a chain is defined by its weakest link. This viewpoint calls into question the idea that attackers will always choose the most lucrative targets, emphasizing the importance of focusing security efforts on vulnerabilities viewed as more accessible. Using this strategy necessitates comprehensive risk studies in which firms identify and prioritize potential hazards based on their severity. Prioritizing the mitigation of the weakest link ensures a proactive and effective defense against possible threats by proactively focusing security actions towards the most significant flaws. Addressing the weakest link in the IT security chain is critical for businesses looking to strengthen their systems and effectively combat possible cyber threats (M.W., 2021).

Upon thorough analysis, the login page emerges as a potentially vulnerable component within my application for the following reasons:

* Primary User Interaction Point: The login page serves as the first point of contact for users, making it an appealing target for attackers seeking illegal access.
* Weaknesses of the Authentication System: In the absence of strong fortification, user authentication systems on the login page are vulnerable to common exploits, including password attacks, brute force attempts, and credential stuffing.
* User-Induced Security Risks: Users may unwittingly undermine security integrity by using weak passwords or falling victim to phishing attacks. These user-related flaws amplify the login page's overall susceptibility.

A complete set of security measures has been developed to offer powerful protection against potential threats in reinforcing the login system within my application. Among the primary security practices used are:

* Hashing.
* Strong password Policy.
* Avoid similarity to username, first name, middle name, or last name.
* Add salt to the password before hashing.
* Use strong hashing algorithm.
* Encrypt salt. (Use strong encryption algorithm)
* Store secret key that used for encrypting salt indifferent database.
* Limitation for trying the password.
* Use strong keys for encrypting and hashing.
* Two-Authentication.
* Use OTP (One-Time-Password).

## 1.2. Fail Securely

Fail Securely is a secure coding principle that highlights the importance of designing software systems to respond to errors and unexpected failures in a secure manner. In essence, it advocates for a controlled and secure response to system difficulties, ensuring that the program maintains a consistent and predictable response even in the face of failures, decreasing the potential for security vulnerabilities and unauthorized access.

Benefits of Fail Securely:

* Consistency in Response: Failing securely ensures that error responses align with security goals, providing a consistent and predictable reaction to unexpected events.
* Controlled Deviations: The principle enables developers to exercise precise control over deviations from intended security procedures, minimizing the impact of unexpected errors.
* Enhanced Resilience: By addressing unforeseen circumstances in a measured manner, failing securely contributes to the system's overall resilience, reducing the potential for security vulnerabilities and unauthorized access.

# 2. Quality Principles

## 2.1. Flexibility and Reusability

Flexibility in programming refers to the ability of code to adapt to changes without requiring a total rebuild. It entails avoiding inflexible coding standards, adopting modular structures, and employing reusable components. This method assures that the software is adaptable, scalable, and capable of effortlessly supporting new requirements or user needs.

Reusability, on the other hand, is creating code in a way that ensures parts or modules can be used in several situations. Reusable code, rather than writing new code for comparable tasks, increases productivity, collaboration, and easier maintenance. A good example is designing a single React component that can be used on numerous pages or within separate components. (Alves, 2023)

Benefits of Flexibility and Reusability:

* Savings in Time and Effort: Flexibility enables quick response to changes, while reusability reduces the need to recreate code for comparable processes, reducing unnecessary effort and time.
* Increased Efficiency: By adopting flexible and reusable code methodologies, developers may work more effectively. Not only does this boost individual productivity, but it also improves collaboration within development teams.
* Simplified Maintenance: Flexibility ensures that software changes or updates can be made without rewriting the entire code. Reusability simplifies maintenance by allowing changes to be done in a single location while ensuring consistency across reused components.

## 2.2. Understandability

Understandability in programming refers to the clarity and accessibility of code for developers. Clear code reduces the learning curve for new team members, fostering effective cooperation and project longevity. Using descriptive variable names, comprehensive documentation, and consistent coding conventions improves code understandability, allowing current and future developers to easily interpret, modify, and extend the code, encouraging a more efficient and collaborative working environment.

Benefits of Understandability:

* Scalability: As projects grow in size, having code that is simple to understand becomes increasingly important. It enables seamless scalability, making it simpler to extend and adapt the codebase to meet new features or required changes.
* Knowledge Transfer: Understandable code facilitates knowledge transfer within a development team. It ensures that many team members can work on different areas of the codebase without experiencing insurmountable obstacles.
* Enhanced Debugging: When code is simple to comprehend, debugging gets easier. Developers can find and repair defects more quickly, resulting in speedier problem resolution.

# 3. Use Case and Misuse Case Diagram for HealthcarePro Application

A diagram of a network

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | Login | **Use case number** | 1 |
| **Primary actors** | Doctor, Patient, and Register | **Secondary Actors** | None |
| **Description** | Allow the doctors, patients, and registrars to into their accounts. | | |
| **Pre-condition** | Doctors, patients, and registrars have a valid username and password. | | |
| **Post-condition** | The system displays the relevant homepage. | | |
| **Main flow** | 1. The customer enters his username and password. 2. The system triggers verify password and username use case. 3. The user's access privileges are determined by their designated role (Doctor, Patient, or Registrar 4. The system displays the customer’s homepage. | | |
| **Alternative flow** | 1.a. Surpasses the maximum login attempts.   1. The system locks the user account. 2. The system displays a message that the user exceeds the limit of attempts. 3. The use case ends.   1.b. Enters malicious inputs.   1. Sanitize all the inputs. 2. Procedures are used. 3. The use case resumes from step 2.   2.a. Attempts to log in with unauthorized credentials   1. The system will send a code to your phone number for you to enter. 2. The use case resumes at main flow step 3.   2.b. Invalid username or password.   1. The system triggers show an error message use case. 2. The system prompts for the username and password. 3. The use case resumes at main flow step 1. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | Register Patient | **Use case number** | 2 |
| **Primary actors** | Register | **Secondary Actors** | None |
| **Description** | Allow the registrars to enter the patient’s information. | | |
| **Pre-condition** | The registrar is authenticated. | | |
| **Post-condition** | The patient's information is successfully recorded in the system. | | |
| **Main flow** | 1. The registrar selects the option to register a new patient. 2. The system prompts the registrar to enter patient information (First name, Middle name, last name, age, contact details, username, and password). 3. The registrar submits the entered information. 4. The system validates and records the patient's information. | | |
| **Alternative flow** | 1a. There are missing or invalid entries.   1. The system prompts the registrar to correct them or enter the missing information. 2. The use case resumes at main flow step 2.   3a. In case of a system error.   1. The registrar is prompted to try again 2. The use case ends. | | |

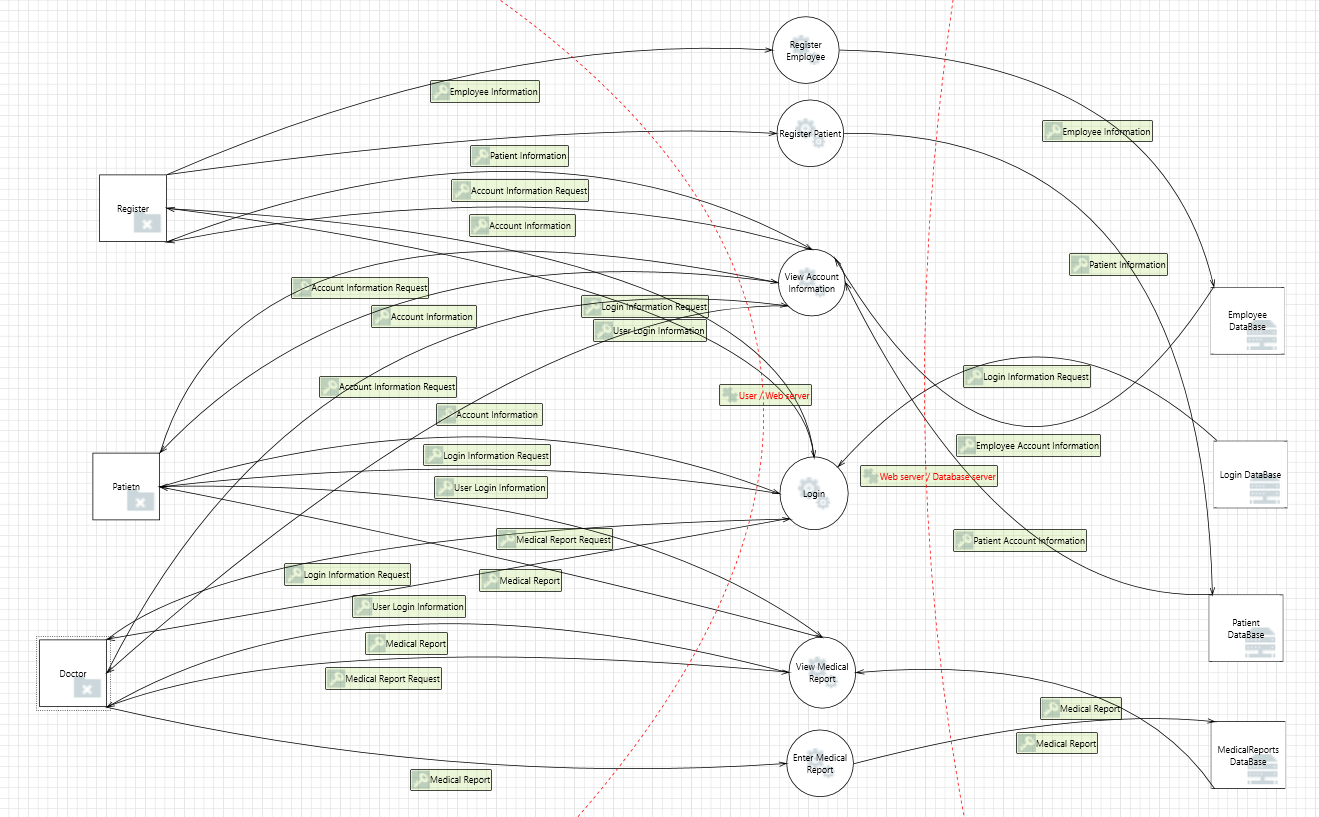
|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | Register Employee | **Use case number** | 3 |
| **Primary actors** | Register | **Secondary Actors** | None |
| **Description** | Allow the registrars to enter the employee’s information. | | |
| **Pre-condition** | The registrar is authenticated. | | |
| **Post-condition** | The employee’s information is successfully recorded in the system. | | |
| **Main flow** | 1. The registrar selects the option to register a new employee. 2. The system prompts the registrar to enter employee information (First name, Middle name, last name, age, contact details, username, and password). 3. The registrar submits the entered information. 4. The system validates and records the employee’s information. | | |
| **Alternative flow** | 1a. There are missing or invalid entries.   1. The system prompts the registrar to correct them or enter the missing information. 2. The use case resumes at main flow step 2.   3a. In case of a system error.   1. The registrar is prompted to try again 2. The use case ends. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | View Information | **Use case number** | 4 |
| **Primary actors** | Doctor, Patient, and Register | **Secondary Actors** | None |
| **Description** | Allow doctors, patients, and registrars to view their account information within the system. | | |
| **Pre-condition** | The user (Doctor, Patient, or Registrar) is authenticated. | | |
| **Post-condition** | The requested information is displayed to the user. | | |
| **Main flow** | 1. User logs into the system. 2. User navigates to the "View Information" page. 3. The system retrieves and displays the requested information. | | |
| **Alternative flow** | None | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | View Medical Report | **Use case number** | 5 |
| **Primary actors** | Patient, and Doctor | **Secondary Actors** | None |
| **Description** | Allow doctors, and patients to view the medical report within the system. | | |
| **Pre-condition** | The user (Doctor, and Patient) is authenticated. | | |
| **Post-condition** | The requested medical report is displayed to the user. | | |
| **Main flow** | 1. User (Doctor or Patient) selects the option to view medical reports. 2. If the user is a doctor, then the system presents a list of available medical reports. 3. If the user is a patient, then the system presents her/his own available medical reports. 4. The system retrieves and displays the selected medical report to the user. | | |
| **Alternative flow** | None | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case name** | Enter Medical Report | **Use case number** | 6 |
| **Primary actors** | Doctor | **Secondary Actors** | None |
| **Description** | Allow doctors to enter medical report for the patients. | | |
| **Pre-condition** | The Doctor is authenticated. | | |
| **Post-condition** | The requested medical report is successfully entered into the system | | |
| **Main flow** | 1. The doctor selects the option to enter a medical report for a specific patient. 2. The system prompts the doctor to input relevant medical information (diagnosis and treatment plan). 3. The doctor submits the entered medical report. 4. The system validates and stores the medical report in the patient's record. 5. The system displays the updated medical report to the doctor. | | |
| **Alternative flow** | 1.a. Enters malicious inputs.   1. Sanitize all the inputs. 2. Procedures are used. 3. The use case resumes from step 4. | | |

# 4. Data Flow Diagram for HealthcarePro Application



# 7. Type of Testing methods Used in HealthcarePro Application

Testing the satisfaction of a HealthcarePro application's requirements entails a variety of testing methods to ensure that the software meets the specified requirements and performs as expected.

Here are some testing methods commonly used for the HealthcarePro application:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Testing Type* | *Testing Methods* | *Definition* | *Benefits* | *Tools* |
| Functional Testing | Integration Testing | Individual software modules or components are joined and tested as a group during the methodical testing procedure known as integration testing. The primary objective is to detect defects in the interactions between integrated components and ensure the proper functioning of the integrated system. (Awati, 2022) | * Guarantees the smooth operation of linked units. * Adapts to changing requirements effectively. * Identifies defects missed in unit testing. * Enhances overall system reliability. * Integration testing helps address common issues that arise from the combination of different modules or components. * Integration testing detects and resolves issues early in the development process, reducing the likelihood of encountering problems in later stages or in a production environment. | * JUnit * TestNG * Postman * Manual   Used tool: Manual |
| Functional Testing | Smoke Testing | Smoke testing is an important software testing method used to rapidly confirm that an application is functioning as intended. It seeks to identify significant problems that might prevent additional testing. Before conducting more thorough testing, smoke tests verify that application builds are stable. Smoke testing assesses major operations before going into more depth. Smoke testing is a crucial step in the development process since it helps to prevent serious software flaws, which are discovered after every build and before release. (Biswas, n.d.) | * Smoke testing finds significant errors early in the development cycle, enabling developers to fix important problems before they affect end users or spread to later stages. * Smoke tests are simple to conduct and can be executed frequently without overtaxing servers and resources. * By identifying issues early on, smoke testing spares development teams from laborious manual code searches. This efficiency, especially when automated, results in considerable time and cost savings since it enables quick and reliable testing, which facilitates a more seamless development process. | * Selenium * Jenkins * Manual   Used tool: Manual |
| Functional Testing | Unit testing | Unit testing is a type of automated software testing in which individual units or components of a software program are evaluated using specialized tools or frameworks. In contrast to manual testing by developers, unit tests are performed by dedicated software, such as a unit testing framework or tool. This automated approach enables the creation of repeatable and continuous tests. (Oppermann, 2023) | * Through the regular running of unit tests, developers gain immediate feedback on code quality. * Unit tests force developers to work on code rather than just write it. In other words, after receiving feedback from the unit test, the developer must constantly rethink their own process and optimize the produced code. * Unit tests allow for extensive test coverage. * With speed and accuracy, it is possible to undertake automated, high-quality testing of the entire software unit by unit. | * JUnit * pytest * NUnit * Manual   Used tool: Junit |
| Non- Functional Testing | Performance Testing | Performance testing is a non-functional testing method that evaluates a software application's speed, responsiveness, scalability, and stability under various conditions. The major purpose is to evaluate how effectively the system operates in terms of responsiveness and speed under varying loads and stresses. Performance testing, as opposed to functional testing, investigates the overall system behavior and its ability to manage increasing degrees of user interactions and data. (Belcher, 2023) | * Increases confidence in the software's overall quality by confirming its performance under realistic scenarios. * Assures development teams, stakeholders, and end users of the application's dependability and efficiency. * Identifies and solves performance issues, providing quick response times and excellent user experiences. * Improves client satisfaction by providing applications that meet or exceed user expectations. * Ensures that your software stands out in a crowded market where user experience is critical. | * Apache JMeter * LoadRunner * Gatling * Manual   Used tool: Manual |
| Maintenance Testing | Regression testing | Regression testing is a type of software maintenance testing used to ensure that a recent program or code update has not negatively impacted current features. (Hamilton, 2024) | * Regression testing methodically determines whether new features, improvements, or code modifications have introduced new bugs or problems. * Makes ensuring that problems that have been found and fixed in the past don't come up again when new modifications are made. * creates a reliable, systematic testing procedure that gives users trust in the operation and dependability of the product. | * Selenium * JUnit/TestNG * TestComplete * Manual   Used tool: Junit and Manual |

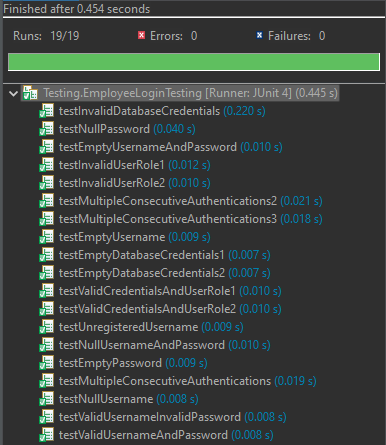
# 8. Type of Security Testing methods Used in HealthcarePro Application

|  |  |  |  |
| --- | --- | --- | --- |
| *Testing Methods* | *Definition* | *Benefits* | *Used Tools* |
| Static Analysis Security Testing (SAST) | SAST is a proactive software testing technique that identifies potential security vulnerabilities in programs by examining their source code without executing them. It is a white box testing method that examines the code thoroughly to find vulnerabilities. (SAST: basic beginner's guide, 2022) | * Early Detection: Identifies software vulnerabilities early in the development process. * Fixes that are less expensive: Allows for simpler and less expensive security fixes. * White-Box Advantage: Access to source code allows for in-depth study. * Resource Optimization: Examines the full codebase efficiently, optimizing security resources. * Automation and speed: Scans millions of lines of code in seconds, outperforming manual reviews. * Results with High Confidence: Systematic analysis ensures high-confidence vulnerability identification. * Integration: An essential component of application security practices that aligns with business objectives. | * SonarQube. * SonarCloud. * Veracod. * Codacy. * Checkmarx. * Microsoft DevSkim. * PMD. |
| Dynamic Analysis Security Testing (DAST) | DAST is a method of app security testing that assesses a running application without knowing its internal structure. It is carried out as "black box" testing, in which the application's responses to simulated attacks are observed, replicating real-world circumstances in order to uncover vulnerabilities and assess susceptibility to malicious attacks. DAST adds to Static Analysis Security Testing (SAST) by concentrating on runtime security problems. (What is Dynamic Application Security Testing (DAST) and how does it work?, n.d.)  DAST types:   * Fuzz Testing:   Fuzz testing exposes security vulnerabilities by bombarding a software application with diverse and unexpected data inputs.   * Reliability Testing:   Reliability testing assesses a software application's stability and consistency, identifying potential points of failure in dynamic scenarios.   * Resiliency Testing:   Resiliency testing determines an application's capacity to survive and recover from security threats or disruptive events, hence increasing overall robustness.  **And I used the Fuzz testing type.** | * Real-World Vulnerability Assessment: To effectively identify vulnerabilities, DAST comprehensively evaluates programs that are already running while replicating real-world circumstances. * Rapid Issue Detection: DAST rapidly identifies possible security flaws by simulating attacks and evaluating defenses. * Smooth DevSecOps Integration: DAST makes it easier to integrate DevSecOps, and it offers insightful feedback for effective tool integration with SecOps and DevOps. * Risk Mitigation: DAST finds and fixes possible exploits used in actual assaults on apps, which is essential for lowering business risk. * Effective Remediation: By providing high-quality vulnerability assessment reports and facilitating quick issue resolution, DAST speeds up the remediation process. * Adaptability to Change: DAST enables teams to keep ahead of new threats and shifts in adversaries' knowledge by adapting to changing application landscapes. * Empowered Quick Corrections: When DAST is used correctly, teams are given the authority to make quick corrections without interfering with daily operations. | * Tenable.io. * Netsparker. * AppScan (IBM Security AppScan). * OWASP ZAP. * Nessus. * StackHawk.   Used tools: Jazzer. |

# 9. HealthcarePro Application Test Cases

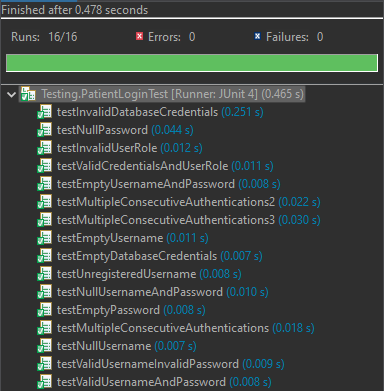
## 9.1. Employee Login Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *ID* | *What to test* | *Description (inputs)* | *Expected* | *Actual* | *Status* |
| 1 | Unregistered Username | "Hamza", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for unregistered username | Authentication failed for unregistered username | Done |
| 2 | Valid Username and Password | "HO8", "H##43USt", "Employee", "HEU445#$@we" | Authentication should succeed for registered username and valid credentials | Authentication succeeded for registered username and valid credentials | Done |
| 3 | Valid Username, Invalid Password | "ho8", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for registered username and invalid password | Authentication failed for registered username and invalid password | Done |
| 4 | Test Empty Username | "", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for empty username | Authentication failed for empty username | Done |
| 5 | Test Empty Password | "HO8", "", "Employee", "HEU445#$@we" | Authentication should fail for empty password | Authentication failed for empty password | Done |
| 6 | Empty Username and Password | "", "", "Employee", "HEU445#$@we" | Authentication should fail for empty username and password | Authentication failed for empty username and password | Done |
| 7 | Test Null Username | null, "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for null username | Authentication failed for null username | Done |
| 8 | Test Null Password | "HO8", null, "Employee", "HEU445#$@we" | Authentication should fail for null password | Authentication failed fail for null password | Done |
| 9 | Null Username and Password | null, null, "Employee", "HEU445#$@we" | Authentication should fail for null username and password | Authentication failed for null username and password | Done |
| 10 | Invalid Database Credentials | "HO8", "H##43USt", "InvalidUser", "InvalidPass" | Authentication should fail for invalid database credentials | Authentication failed for invalid database credentials | Done |
| 11 | Valid Credentials and the returned result for register | "HO8", "H##43USt", "Employee", "HEU445#$@we" | User role should be "Register" | User role is “Register" | Done |
| 12 | Valid Credentials and the returned result for doctor | "S378", "H##43USt", "Employee", "HEU445#$@we" | User role should be "Doctor" | User role is "Doctor" | Done |
| 13 | Invalid UserRole with doctor username | "S378", "H##43USt", "Employee", "HEU445#$@we" | User role should not be "Manager" | User role is "Doctor" | Done |
| 14 | Invalid UserRole with register username | "HO8", "H##43USt", "Employee", "HEU445#$@we" | User role should not be "Manager" | User role is "Register" | Done |
| 15 | Multiple Consecutive Authentications | "HO8", "H##43USt", "Employee", "HEU445#$@we" and "S378", "H##43USt", "Employee", "HEU445#$@we" | Both authentications should succeed | Both authentications succeeded | Done |
| 16 | Multiple Consecutive Authentications | "HO8", "H##43USt", "Employee", "HEU445#$@we" and "ooo", "H##43USt", "Employee", "HEU445#$@we" | First authentication should succeed, second should fail | First authentication succeeded, second failed | Done |
| 17 | Multiple Consecutive Authentications | "eee", "H##43USt", "Employee", "HEU445#$@we" and "ooo", "H##43USt", "Employee", "HEU445#$@we" | Both authentications should fail | Both authentications failed | Done |
| 18 | Empty Database Credentials 1 | "HO8", "H##43USt", "", "" | Authentication should fail for empty database credentials | Authentication failed for empty database credentials | Done |
| 19 | Empty Database Credentials 2 | "S378", "H##43USt", "", "" | Authentication should fail for empty database credentials | Authentication failed for empty database credentials | Done |



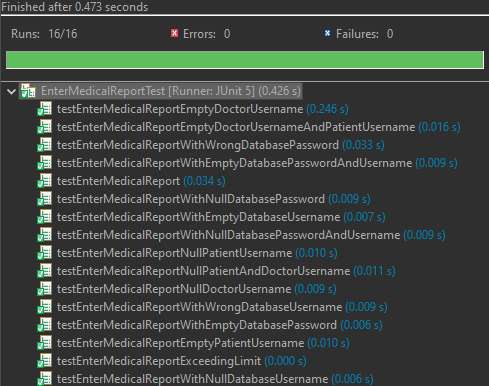
## 9.2. Patient Login Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *ID* | *What to test* | *Description (inputs)* | *Expected* | *Actual* | *Status* |
| 1 | Unregistered Username | "Hamza", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for unregistered username | Authentication failed for unregistered username | Done |
| 2 | Valid Username and Password | "HHH", "H##43USt", "Employee", "HEU445#$@we" | Authentication should succeed for registered username and valid credentials | Authentication succeeded for registered username and valid credentials | Done |
| 3 | Valid Username, Invalid Password | "ho8", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for registered username and invalid password | Authentication failed for registered username and invalid password | Done |
| 4 | Test Empty Username | "", "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for empty username | Authentication failed for empty username | Done |
| 5 | Test Empty Password | "HHH", "", "Employee", "HEU445#$@we" | Authentication should fail for empty password | Authentication failed for empty password | Done |
| 6 | Empty Username and Password | "", "", "Employee", "HEU445#$@we" | Authentication should fail for empty username and password | Authentication failed for empty username and password | Done |
| 7 | Test Null Username | null, "H##43USt", "Employee", "HEU445#$@we" | Authentication should fail for null username | Authentication failed for null username | Done |
| 8 | Test Null Password | "HHH", null, "Employee", "HEU445#$@we" | Authentication should fail for null password | Authentication failed fail for null password | Done |
| 9 | Null Username and Password | null, null, "Employee", "HEU445#$@we" | Authentication should fail for null username and password | Authentication failed for null username and password | Done |
| 10 | Invalid Database Credentials | "HHH", "H##43USt", "InvalidUser", "InvalidPass" | Authentication should fail for invalid database credentials | Authentication failed for invalid database credentials | Done |
| 11 | Valid Credentials and User Role | "HO8", "H##43USt", "Employee", "HEU445#$@we" | User role should be "Patient" | User role is “Patient" | Done |
| 12 | Valid Credentials and Invalid User Role | "HHH", "H##43USt", "Patient", "PEN#$82" | User role should not be "Doctor" | User role is “Patient" | Done |
| 13 | Empty Database Credentials | "HHH", "H##43USt", "", "" | Authentication should fail for empty database credentials | Authentication failed for empty database credentials | Done |
| 14 | Multiple Consecutive Authentications | "HHH", "H##43USt", "Patient", "PEN#$82" and "AAA", "A##43USt", "Patient", "PEN#$82" | First authentication should succeed, second should fail | First authentication succeeded, second failed | Done |
| 15 | Multiple Consecutive Authentications | "HHH", "H##43USt", "Patient", "PEN#$82" and "ooo", "H##43USt", "Patient", "PEN#$82" | Both authentications should succeed | Both authentications succeeded | Done |
| 16 | Multiple Consecutive Authentications | "eee", "H##43USt", "Employee", "HEU445#$@we" and "ooo", "H##43USt", "Employee", "HEU445#$@we" | Both authentications should fail | Both authentications failed | Done |

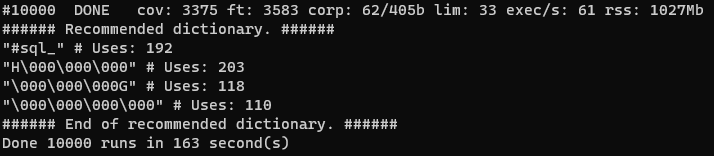


## 9.3. Enter New Medical Report Test Cases

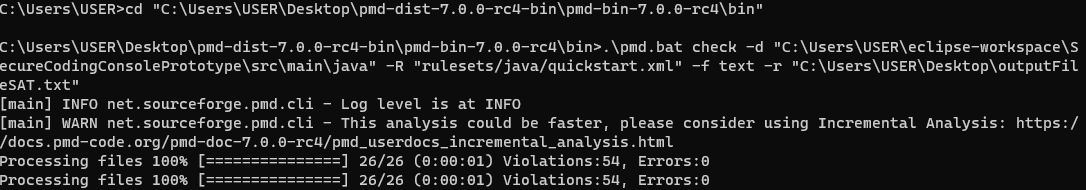
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *ID* | *What to test* | *Description (inputs)* | *Expected* | *Actual* | *Status* |
| 1 | Enter Medical Report | "HHH", "S378", "Valid", "Prescription", "Doctor", "S@#m378H" | Successful addition of a valid medical report | Successfully added. | Done |
| 2 | Enter Medical Report with Empty Patient Username | "", "S378", "Valid", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 3 | Enter Medical Report with Empty Doctor Username | "HHH", "", "Valid", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 4 | Enter Medical Report with Empty Doctor and Patient Username | "", "", "Valid ", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 5 | Enter Medical Report with Null Doctor Username | "HHH", null, "Valid", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 6 | Enter Medical Report with Null Patient Username | null, "S378", "Valid", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 7 | Enter Medical Report with Null Patient and Doctor Username | null, null, "Valid", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 8 | Enter Medical Report Exceeding Limit | "HHH", "S378", "Very long medical report……", "Prescription", "Doctor", "S@#m378H" | Unsuccessful addition of an invalid medical report | Unsuccessful addition of an invalid medical report | Done |
| 9 | Enter Medical Report with Wrong Database Username | "HHH", "S378", "Valid", "Prescription", "Employee", "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 10 | Enter Medical Report with Wrong Database Password | "HHH", "S378", "Valid", "Prescription", "Doctor", "S@#m333378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 11 | Enter Medical Report with Null Database Password | "HHH", "S378", "Valid", "Prescription", "Doctor", null | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 12 | Enter Medical Report with Null Database Username | "HHH", "S378", "Valid", "Prescription", null, "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 13 | Enter Medical Report with Null Database Username and Password | "HHH", "S378", "Valid", "Prescription", null, null | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 14 | Enter Medical Report with Empty Database Username | "HHH", "S378", "Valid", "Prescription", null, "S@#m378H" | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 15 | Enter Medical Report with Empty Database Password | "HHH", "S378", "Valid", "Prescription", "Doctor ", null | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |
| 16 | Enter Medical Report with Empty Database Username AND Password | "HHH", "S378", "Valid", "Prescription", null, null | Unsuccessful addition of a valid medical report | Unsuccessful addition of a valid medical report | Done |

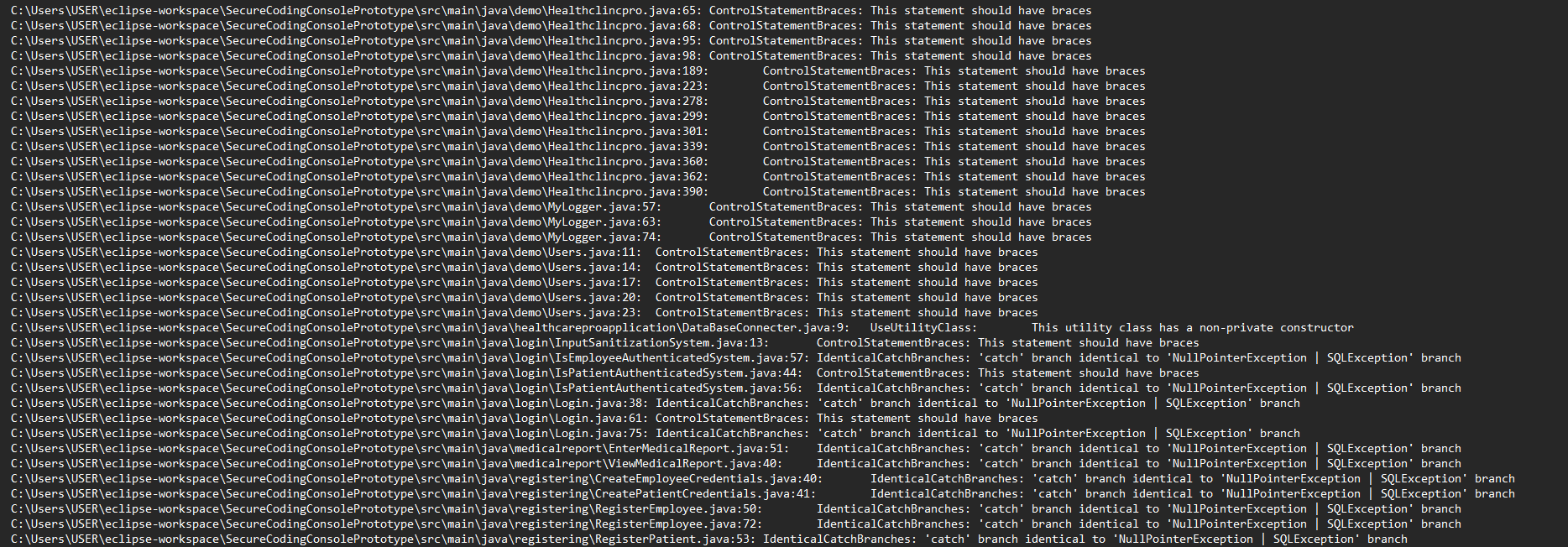


# 12. Security-Based Testing Techniques



# 11. Code Review Analysis



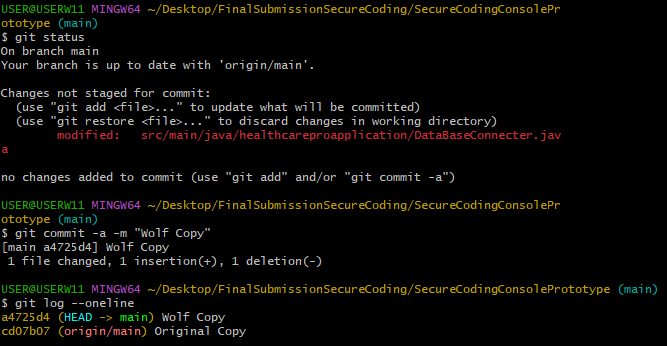


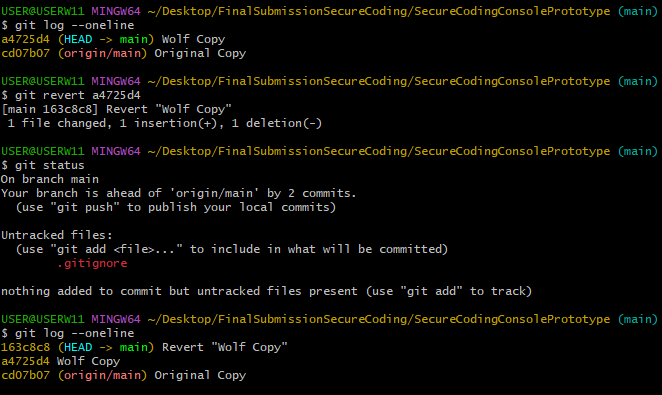
A black screen with many squares

Description automatically generated

# 12. Code Versioning

## Screen Shots:





## GitHub Link:

<https://github.com/Hamzx1/Healthclinicpro>

# References

Alves, J. (2023, March 14). *Abstraction, reusability and flexibility in software design*. Retrieved from Dev: https://dev.to/alvesjessica/abstraction-reusability-and-flexibility-in-software-design-3k7g

Awati, R. (2022, January 13). *What is integration testing (I&T)?* Retrieved from TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/integration-testing#:~:text=Integration%20testing%20%2D%2D%20also%20known,tested%20as%20a%20combined%20entity.

Belcher, D. (2023, May 2). *What Is Performance Testing? The Key to Delivering Fast and Scalable Software*. Retrieved from mabl: https://www.mabl.com/articles/what-is-performance-testing

Biswas, S. (n.d.). *What Is Smoke Testing: Examples & Best Practices*. Retrieved from lambdatest: https://www.lambdatest.com/learning-hub/smoke-testing

Hamilton, T. (2024, January 3). *What is Regression Testing?* Retrieved from GURU99: https://www.guru99.com/regression-testing.html#why-regression-testing

M.W., S. (2021, March 26). Retrieved from TechGenix: https://techgenix.com/weakest-link-principle-of-it-security/?\_\_cf\_chl\_tk=pyYPrBUtmB.ygkw2nSAF.f0y3gxfMdmDFD0.G9KfUvc-1703314647-0-gaNycGzNFCU

Oppermann, A. (2023, July 24). *What Is Unit Testing?* Retrieved from Built In: https://builtin.com/software-engineering-perspectives/unit-testing

*SAST: basic beginner's guide*. (2022, April 26). Retrieved from Sonar: https://www.sonarsource.com/learn/sast/

*What is Dynamic Application Security Testing (DAST) and how does it work?* (n.d.). Retrieved from SYNOPSYS: https://www.synopsys.com/glossary/what-is-dast.html