

National College of Ireland
Project Submission Sheet –
2022/2023

Student Name: Vedant Thaksen Gavhane
Student ID: X21182663
Programme: MSCCYB1_JAN23B_I **Year:** 2023
Module: Secure Programming for Application Development
Lecturer: Eugene McLaughlin
Submission Due Date: 14/08/2023
Project Title: CA_2 2023
Word Count: 5465 excluding references and appendices.

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Vedant
Date: 14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year.
Any project/assignment not submitted will be marked as a fail.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstract

The goal of this project is to create a safe application in Java which stores data in a database maintained by MySQL. In order to find and fix potential security flaws in the program and database, the work includes providing security testing approaches into practice. The initial focus will be on setting optimal procedures for protecting data, access control, handling sessions, and the identification of users in place. Functionality like password encryption, data encryption in the database, input sanitization from users, detecting and mitigating SQL injection, and cross-site scripting threats will be included in the application. It is important to keep comprehensive security strategies to secure Java applications. A security application will be improving the common security breaches by using SAST and DAST methodologies. The ultimate goal is to create an application that is reliable and secure that is designed for enterprise deployments.

Introduction

In today's world, Java applications are used everywhere, including websites and commercial tools. However, despite all the benefits they offer, they can also be vulnerable to cyber threats that could put your data at risk. The purpose of this project is to build a secure Java application that stores data in a MySQL database. However, it's not just about securing the application, well we have to make sure it is working properly so users cannot be worried about their data. We'll use different ways to check and test the library application to make sure no one can break in easily. This paper proposed to focus on robust protection measures to ensure the integrity, confidentiality, and availability of library resources and user data.

As libraries transition from traditional cataloguing to dynamic, interconnected systems, they become potential targets for cyber threats and data breaches. These dangers vary from intrusive attacks that may hamper library services to illegal access to private customer information and intellectual property theft. A comprehensive security measures are important to make sure that library applications continue to be trusted and reliable despite all of these difficulties. The methods for securing user authentication and access control, preventing data breaches through encryption and IDS, and careful observation of the application activities are included in our study. We also cover the growing issue of inter-library data exchange security, which is essential to current library systems.

This paper intends to provide library administrators, software developers, and security practitioners with practical insights and instructions to implement effective security measures by analyzing the difficulties of protecting library applications. The security for library apps appears as an essential component in assuring the continuing growth of knowledge distribution and protection in this digital era where online literacy and information access thrive.

1. State of the art in the latest programming paradigms

One further way to think about a paradigm is as a means for working through a challenge or carrying out an activity. A programming paradigm is a method refers to systematic approach that use a particular programming language. Initially, we could say that it's a strategy for problem-solving that use some of the tools and techniques that are available to us based on a certain methodology. There are many recognised programming languages, but in order to be carried out, each of these languages needs to adhere to a certain technique or plans, and the term

“paradigm” refers to this methodology or approach. To conclude there are being many varieties of programming languages, there are also a large number of paradigms to choose from to satisfy any need. The following is an example on them.

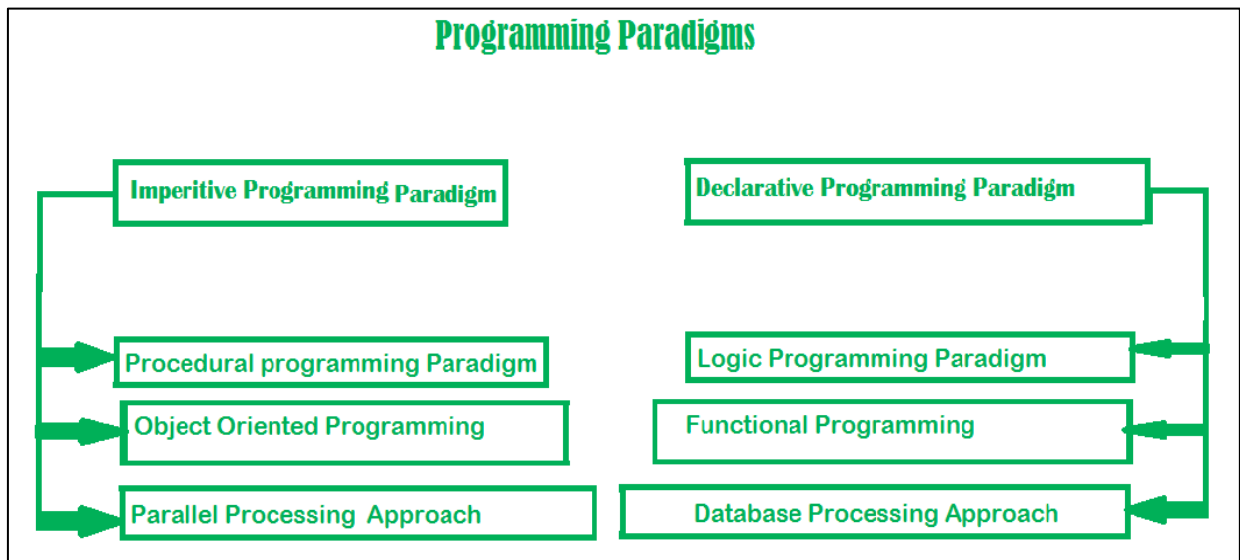


Fig 1. A programming paradigms chart[1]

- I. Imperative programming paradigms:** This was one of those earliest programming paradigms still in use today. It has a strong relationship with machine architecture. Its architecture is based upon Von Neumann’s design. It modifies the state of the programme using assignment statements. It executes sequential tasks by changing its state. The primary focus will be on achieving the objective. In imperative programming, each action is encrypted, so that code itself indicates how the issue is going to be addressed. Thus, imperative programming does not rely on any specified models at any point in the process. But it cannot resolve complex problems also parallel programming is not supported.

There are three main types of imperative programming languages: procedural programming, object-oriented programming (OOP), and parallel processing. Here are the following methods:

- **Procedural programming paradigm:** This theory focuses on the way in which things are done in respect to the machine model they have been built on. Also, there is no difference between a procedural and imperative paradigm approaches. Because of its code-reuse capabilities, it was a lifesaver when it was first implemented.
- **Object oriented programming:** The programme is constructed using a set of classes and objects that are designed to obtain communication, and its primary function is to facilitate such communication. Objects, which are considered to be the most fundamental unit, which are the starting point for every kind of computation that may be conducted. The focus is primarily on the data instead of the technique. It has ability to manage every form of real-life difficulties that exists in the current context.
- **Parallel processing approach:** The execution of programme instructions in parallel refers to the method of breaking those instructions into parts and sending them to more than one processor. A parallel processing system is one that contains a large number of processors, the purpose of which is to speed up the execution of progamme by distributing its tasks among many processors. It appears as though this strategy is one of

“divide and conquer”. Some instances include NESL, which is the earliest ones, and the fact that C/C++ also supports due to specific library functions.

II. Declarative programming paradigm: There are three distinct sections: logic/functional/database. Declarative programming is a method of creating programmes in the field of computer science that avoids discussing the control flow of the underlying processor. It often examines programmes as if they were logical theories. The process of creating parallel code could be simplified. In other words, what the programme actually does is emphasised over how it is being done. It simply states that desired outcome should be achieved. This is the only real difference among the imperative (how to do) and declarative (what to do) approaches to programming languages. As we dig further, we are going to see the logic, functional, and database levels.

- **Logic programming paradigms:** This is often called the abstract form of the computation. It could resolve puzzles, series, and other logical problems. In logic programming, it provides a knowledge base that we already know, and when we send the machine a question and the knowledge base give a result. Typical programming languages don't have the idea for a knowledge base, yet when we're using artificial intelligence and machine learning, we've got models like the perception model that use the same process. In logical programming, the primary focus is on the task and the knowledge base.
- **Functional programming paradigms:** The functional programming paradigms is derived from mathematical principles and does not belong to any specific programming language. The basic concept underlying these paradigms involves the sequential execution of mathematical operations. The primary paradigm for abstraction in this context is the utilisation of functions, which are designed to perform specific computations rather than manipulate data structures. The relationship between data and functions is characterised by a loose association. The function conceals its implementation. The substitution of functions with their respective values does not change the program's semantics. The paradigm of using languages such as Perl and JavaScript are mostly observed.
- **Database/Data driven programming paradigm:** This programming style is founded around the manipulation and transmission of data. Programme statements are determined by data instead of being explicitly programmed with a predetermined sequence of steps. The database programme serves as the central component of a business information system, facilitating essential operations such as file creation, data entry, updating, querying, and reporting. Numerous programming languages have been primarily designed for the purpose of database application. One illustrative example is SQL. The application of this technique is commonly observed in the context of processing structured data streams, where it fulfils the purpose of filtering, transforming, aggregating (e.g., generating statistics). Or invoking other programmes. The concept covers a broad range of practical uses. [1]

Java security versions

Version 20 is the most recent release of Oracle Java SE, while versions 17,11, and 8 belong to the LTS versions for which Oracle Customers were eligible for Oracle Premier support. Oracle will forever provide free open updates to Java 8 for both development and private

use. To every user, especially commercial and industrial use, Oracle will continue to distribute public Java 17 LTS upgrades at no additional cost till September 2024.

The first version of java was **1.0** it released on January 23, 1996, after that **JDK 1.1** is released with lot of changes were made to the AWT event model, serialization, inner classes, and JIT (Just In Time) compiler proposed by Microsoft. [2]

Then **J2SE (1.2)** released on December 8, 1998, they developed 1520 classes in 59 packages. The major improvements were added like Swing API is added as a part of the main classes, strictfp keyword is also added. [2]

The third **J2SE (1.3)** version released on May 8, 2000. They added JNDI (Java Naming and Directory Interface) in some of the main libraries, included HotSpot JVM for J2SE 1.2 JVM version, and added synthetic proxy classes to the JDK.

Then **J2SE (1.4)** released on February 6, 2002 and introduced several security measures like code and classloader improvements, code signing, Java security manager, policy file improvements, etc.

Then **J2SE (1.5)** released on September 2004 and introduced several security measures like pluggable authentication modules integration, security API improvements, changed older cryptographic algorithms to more secured. [2]

The **SE 6**, which came out in December 2006, made security better by specific permission to code and resources, better Security Manager rules, supporting to elliptic curve cryptography algorithms and **SE 7** released in July 2011, **Java 7** introduced features like secure processing API for XML, enhanced cryptographic algorithms, support for Server Name Indication (SNI) for virtual hosting security.

The **Java 8** discovered on March 18, 2014 some security features proposed are improved TLS 1.2, secured Random Number Generation (java.security.Secure.Random), the java.util.Base64 class was added to Java 8 to help with base64 encoding and decoding, which is often used for identity and data security.

Java version **9, 10, and 11** didn't make that such changes but they aimed to improve module system (jigsaw), process API updates, Local variable Type Inference, HTTP client API provides more secure way to interact with HTTP objects, TLS 1.3 improved security.

After that **JDK 12** improved all cryptographic algorithms and developed Key agreement for post-quantum cryptography. **JDK 13** didn't bring any such security updates in their library.

JDK 14 added Edwards-curve digital signature algorithm to enhance cryptographic objects.

JDK 15 improved ZGC (Garbage Collector) for memory management. **JDK 16** proposed Unix-domain socket channels allowing systems on the same host talk to each other securely, better support for code signatures on macOS. **JDK 17** developed a foreign function and memory API. **JDK 18** fixes normal security bugs and foreign functions. **JDK 19** JEP 405 makes it possible for record patterns to directly identify the parts of the record. This extends the syntax matching possibilities of instance of operations and switch expression. **JDK 20** improved all existing security features like record pattern, pattern matching for the switch cases, foreign function and memory API, vector API, virtual threads, etc. [3]

Security aspects of MySQL:

Security is an essential component of any IT infrastructure, particularly in the field of databases. A single failure in this domain can lead to service interruptions, delays,

Complaints from clients, and in the most severe cases, disastrous results. MySQL provides a number of security features such as restricted access, database auditing, and data encryption, the elements and add-ons that make MySQL secure, the basic flow of security rules, provide organizations-level security support. The management of permissions in MySQL is often considered to be a fundamental aspect of its overall security framework. The careful allocation of privileges with a systematic manner provides a fundamental component of access control.

The Grant and revoke commands, as their names indicate, have the ability to give or remove access to people. The appropriate allocation of privileges is considered to be essential for ensuring the security of all system supported by MySQL. Another factor highly associated with access control & secure passwords refers for the protecting of MySQL users itself. Many of the suggestions made in this domain might seem basic nevertheless when integrated with the previous components, they collectively offer an effective security framework for MySQL. In order to enhance the privacy of login credentials in MySQL, it can be advisable to prevent users from transmitting passwords in plain text while instead choose SSL for transmission of the file. Additionally, it should be avoided operating MySQL as the root user, because it can possess the access to the private files that may lead to made changes unauthorizedly. Lastly, it is important to use strong password policy. [4]

2. The current state of security testing techniques and methodologies

Java security technology contains a wide collection of application programming interfaces (APIs), utilities, and instances of widely employed security methods, processes, and protocols. The java security application programming interfaces encompass a diverse array of domains, encompassing cryptography, hashing and salting, secure protocols, authentication, and access control. Java security functionality offers developers an extensive framework for creating applications with robust security measures. Additionally, it provides consumers and admins with a range of tools to effectively administer programmes in a secure manner.

The components of java security are organized in the following structure:

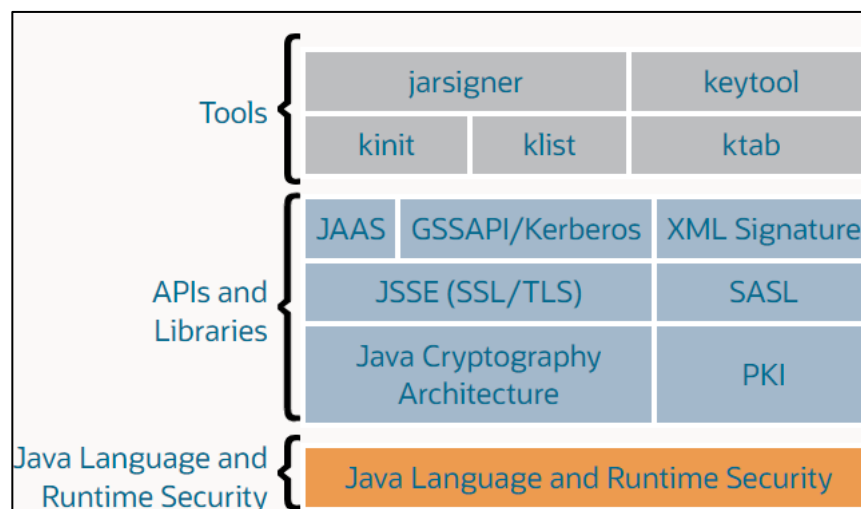


Fig 2. Java security component chart[5]

Application security testing refers to the several methodologies used by organizations to identify and mitigate vulnerabilities which are present in their software applications. Application security testing, commonly known as AppSec testing and AST, contains the comprehensive examination, analysis, and documentation of the security architecture of a software application while it progresses during the lifecycle of software development (SDLC). The concept of application security (AppSec) testing is proactively mitigating software threats before to deployment, followed by immediately detecting vulnerabilities during the production phase. This approach aims to enhance the integrity of the source code and protect applications against potential internal and external attacks.[6]

- **Penetration Testing:** A penetration test, sometimes referred to as a pen test, represents a simulated cyber attack carried out on a system with the purpose of identifying potential vulnerabilities that could be exploited. In the field of application security, penetration testing is often used as a means to enhance the effectiveness of a web application firewall (WAF). Penetration testing covers the intentional attempt to penetrate various application systems, which includes but is not limited to application protocol interfaces (APIs) and frontend/backend servers. The primary objective of this process is to identify threats, like unsanitized inputs which are vulnerable to code injection attacks. [7]
- **Fuzz testing:** Fuzz testing, often known for fuzzing, is an approach to software testing that involves the automated injection of incorrect, flawed, and improper data to an entire system. The purpose of this method is to identify software problems as well as vulnerabilities. A fuzzing application is used to introduce such inputs to the system and thereafter observe for occurrences of exceptions, which might include system failure or unintentional disclosure of information. [8]
- **Vulnerability Scanning:** A vulnerability analysis checks your computer networks for weaknesses in security that hacker could use to take over your devices or obtain your information.
- **Security Scanning:** Security scanning is a way of checking a system, network, program, or additional security product for weakness and possible potential risks. The objective of a security scan is to find flaws that attacker could use to make the device or data less private, less trustworthy, or barely available.
- **Static code analysis:** Static code analysis, static analysis, as well as analysis of source code are all terms that are employed frequently. Static code analysis looks for flaws in the source code result in security breaches. Certainly, this may be done by looking at the original code through manually. But applying tools that do things automatically are much better. Static analysis is often used to make sure that code rules like MISRA are followed. Its also often used to meet business standards like ISO 26262. [9]

The term **SAST** is referred to as **Static application security testing** which examines the application source code to detect known security vulnerabilities. It includes SQL injection, buffer overflows, XML external entity attacks, also different OWASP Top 10 security risks. The SAST technique indicates developers to start testing their applications as soon as possible, before running any of its useful parts. This method detects security flaws in the source code of an app in less time and keeps them from being ignored until later in the development process. This reduces the time it takes to make a program and makes it better. The SAST tool contains the capability of continuously monitoring source code

vulnerabilities by identifying and flagging difficult coding patterns that violates established software development security standards. Additionally, this tool has the capability to automate the process of examining your program code for various vulnerabilities, according to widely recognized security industry standards such as OWASP Top 10 and also SANS Top 25. [10]

Dynamic application security (DAST) The software programs are continuously scanned in real-time using popular vulnerability libraries such as the OWASP Top 10 and SANS/CWE 25. This scanning process aims to identify security issues or exposed vulnerabilities. The Dynamic Application Security Testing technique involves doing closed box analysis to simulate the perspective from an external attacker. The assumption is made that the tester lacks knowledge of the internal working of the program. Runtime program analysis flaws that static application security testing is unable to discover. This includes vulnerabilities that show up exclusively during the execution of the program.

Due to the requirement of a fully functional application, it is advisable to allocate DAST testing to the later stage in the application development procedure. In order to effectively evaluate the program, testers must engage in various interactions such as providing inputs, verifying outputs, and simulating common user behaviours. These tests serve the purpose of verifying the application's resistance against web assaults, specifically cross-scripting (XSS) and SQL injection. [10]

There are various recommended strategies for enhancing the security of the Software Development Life Cycle (SDLC), which include:

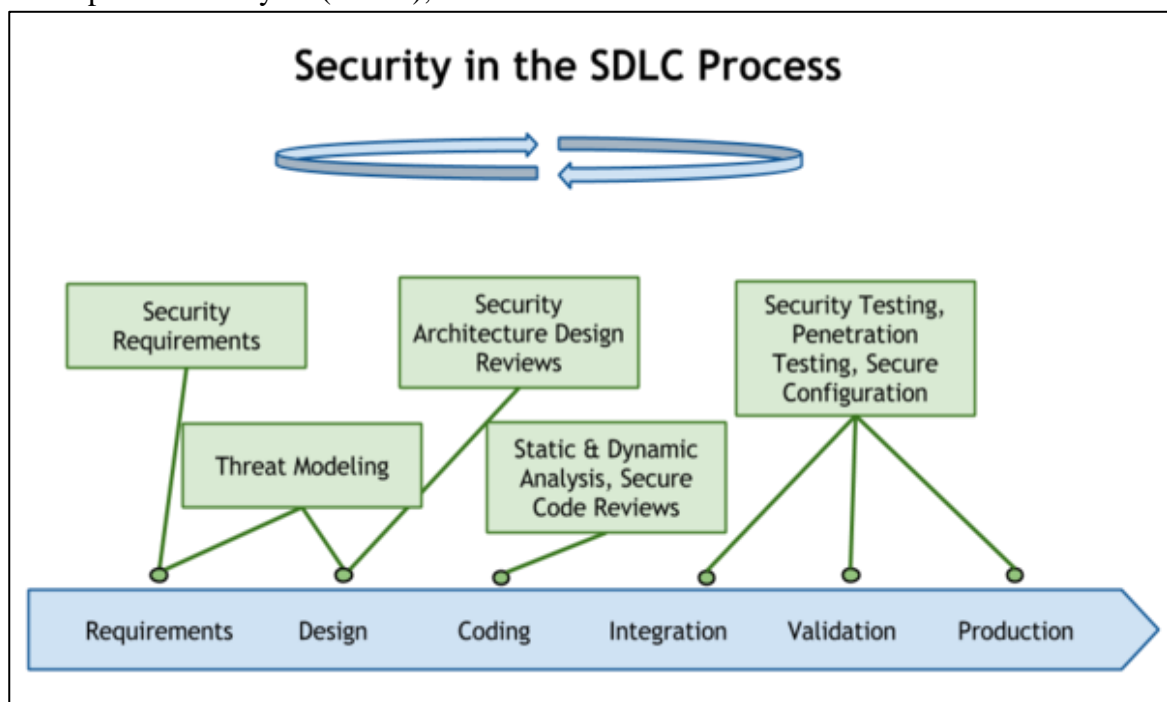


Fig 3. SDLC Process[11]

- a) **Requirements:** The initial stage of the Software Development Life Cycle is commonly referred to by the term "Requirement" phase, during which the project's goals and limitations are specified. During this stage, OWASP proposes the implementation of security criteria for an application.

- b) **Design:** During the design phase, OWASP suggests including security considerations into the design process. The process under consideration is commonly referred to as threat modelling which consists of an analysis of a software application's privacy to identify and address potential threats, resulting in the development of a comprehensive security plan.
- c) **Development:** During the stage of development, OWASP advises developers to stick to "Secure Coding Standards." To ensure compliance with these standards, it is recommended that organizations create awareness among developers on the Secure Coding practices. This is crucial as developers sometimes do not follow the provided guidelines.
- d) **Testing:** During the testing stage, OWASP suggests conducting a penetration test that includes an examination of the infrastructure. This is done to validate whether the issues identified during the phase of design and development were actually effectively addressed. During the stage of development, OWASP suggests that developers adhere to "Secure Coding Standards". To ensure compliance with these standards, it is recommended that organizations raise awareness among developers on secure coding practices. This is crucial because developers often tend to neglect the provided guidelines.
- e) **Deployment:** The application you developed passes from development to improve production during deployment. OWASP suggests a penetration test, such as network testing, during testing to make sure that design and development issues were fixed. OWASP suggests developers to adopt "secure coding standards" during development, but the organization needs to educate developers about secure coding since they typically ignore recommendations.

3. Application security testing

This section of the research project outlines the comprehensive security testing conducted on The Library Management System. The testing process includes the combination by various parts of the testing, in which the static, dynamic, and functional techniques are used to identify any conceptual issues or method errors inside the program. To execute this project, we have used Apache Netbeans IDE to make some changes into the source code and for storing data we used MySQL to store data in encrypted format also it is necessary to make connection with the source code. The existing dataset has numerous vulnerabilities we have to identify those vulnerabilities by using some automated tools and by doing manual code review after threat identification we need to fix some major vulnerabilities like SQL injection, input sanitization, password hashing, and some syntactical errors. By using Java latest secure libraries, we can prevent those threats and provide a clean code for the application.

Sonar Lint:

SonarLint is a code analysis tool which is considered essential in the field of software development. It serves a critical function in the identification and resolution of issues within software development projects. SonarLint is effectively included into widely used Integrated Development Environment (IDEs) like Eclipse, IntelliJ, IDEA, and Apache Netbeans. This tool functions in real-time to promptly furnish developers with immediate evaluations regarding the quality of their code. The importance of good code quality is important in

current software development processes. This pertains to mitigation of problems that may adversely affect users, the prevention of security vulnerabilities not being exposed to the public, and the facilitation of code maintenance.

The utilization of Static Code Analysis is of utmost importance in this context. SonarLint offers a comprehensive and seamlessly integrated user interface. Whenever a problem is identified, it is instantly reported and further clarified within the text. A targeted perspective additionally provides a comprehensive understanding of all matters contained within the document. SonarLint provides alerts regarding issues detected in code, categorizing them into three distinct types: bugs, vulnerabilities, and code maintainability. This facilitates an instant awareness of the associated risks and offers an accurate learning experience through the provision of a comprehensive explanation of the rules. [12]

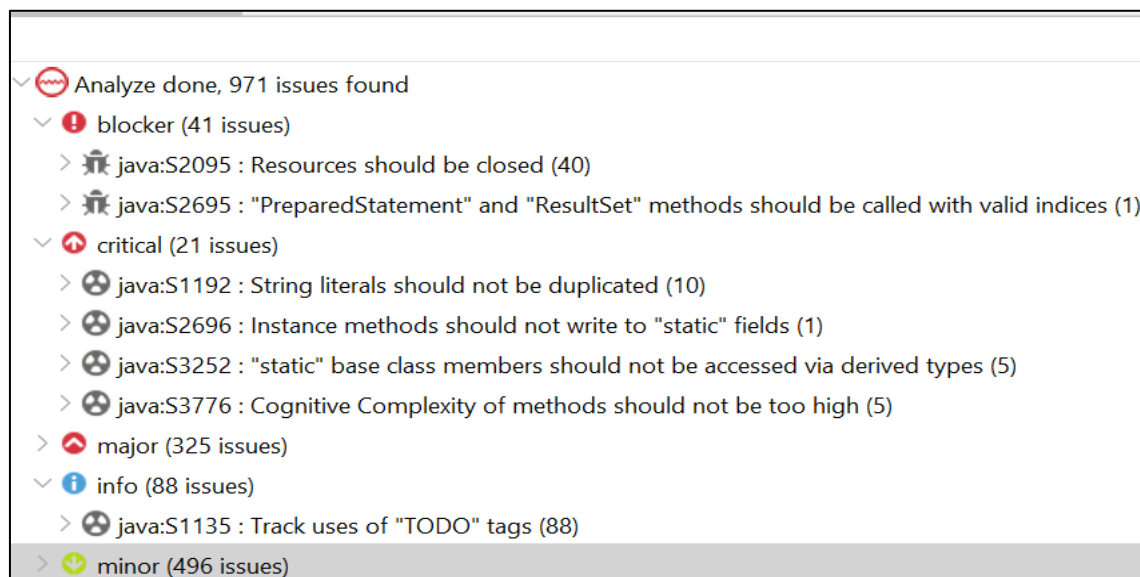


Fig 4. SonarLint testing of Library Management Application

VCG (Visual Code Grepper):

Visual Code Grepper is a free and open-source analysis software which focuses on evaluating the visual aspects in a source code to identify OWASP top 10 security flaws. This method has significance in identifying potential threats inside the source code and providing an extensive view on the locations of vulnerabilities along with issues. The implementation of this method is characterized by its simplicity; however, it proves to be quite valuable in the identification of areas of interest. It should be useful because it offers a few characteristics. It also does some more complicated checks and has a setup file over every language that lets you add any incorrect functions or other text you want to look for. It looks for phrases in comments that could be signs of broken code, and it gives statistics and a pie chart (for the whole source code and for each file) that show the amounts of code, whitespace, comment style, and bad coding.

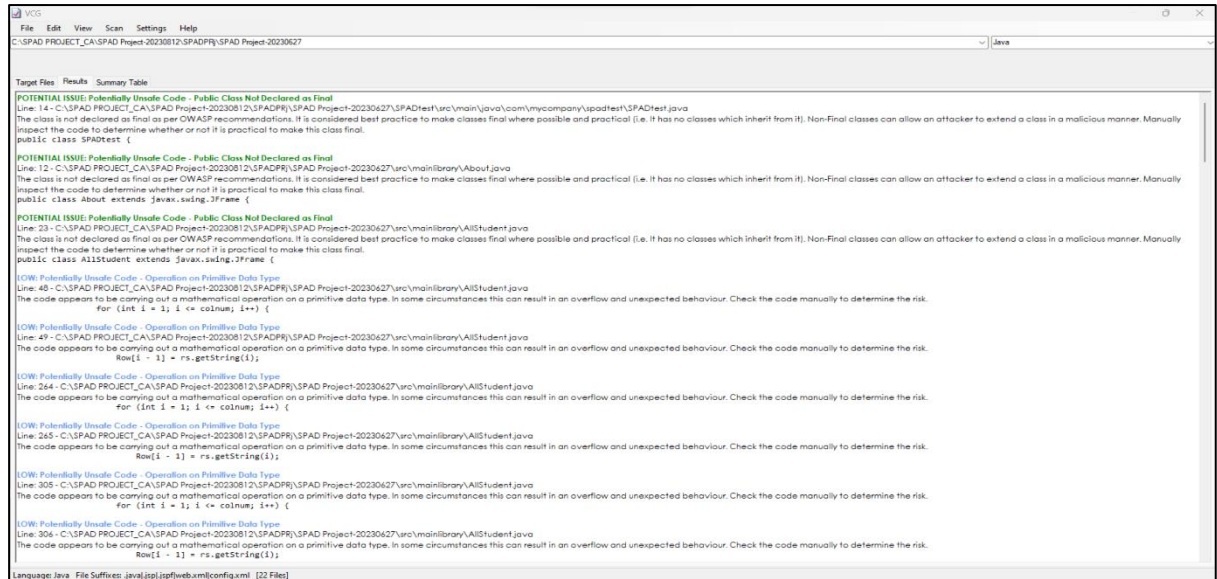


Fig 5. Security checkup on library management system

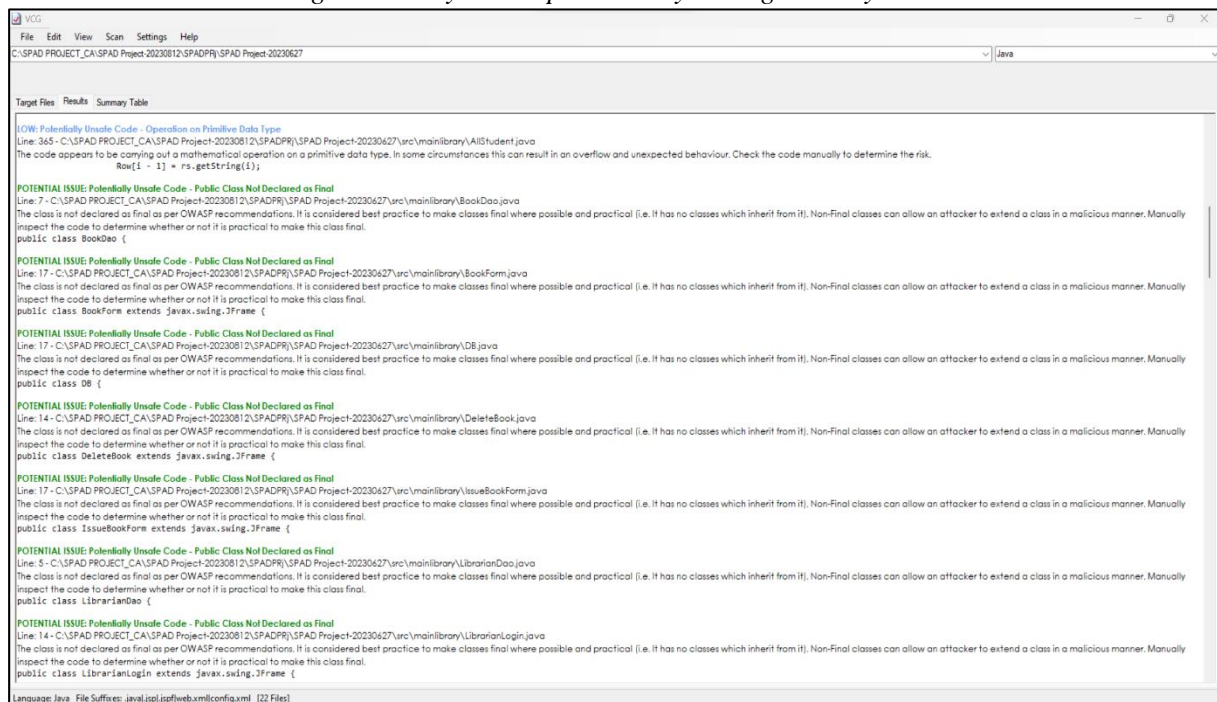


Fig 6. Potential threats on library management system

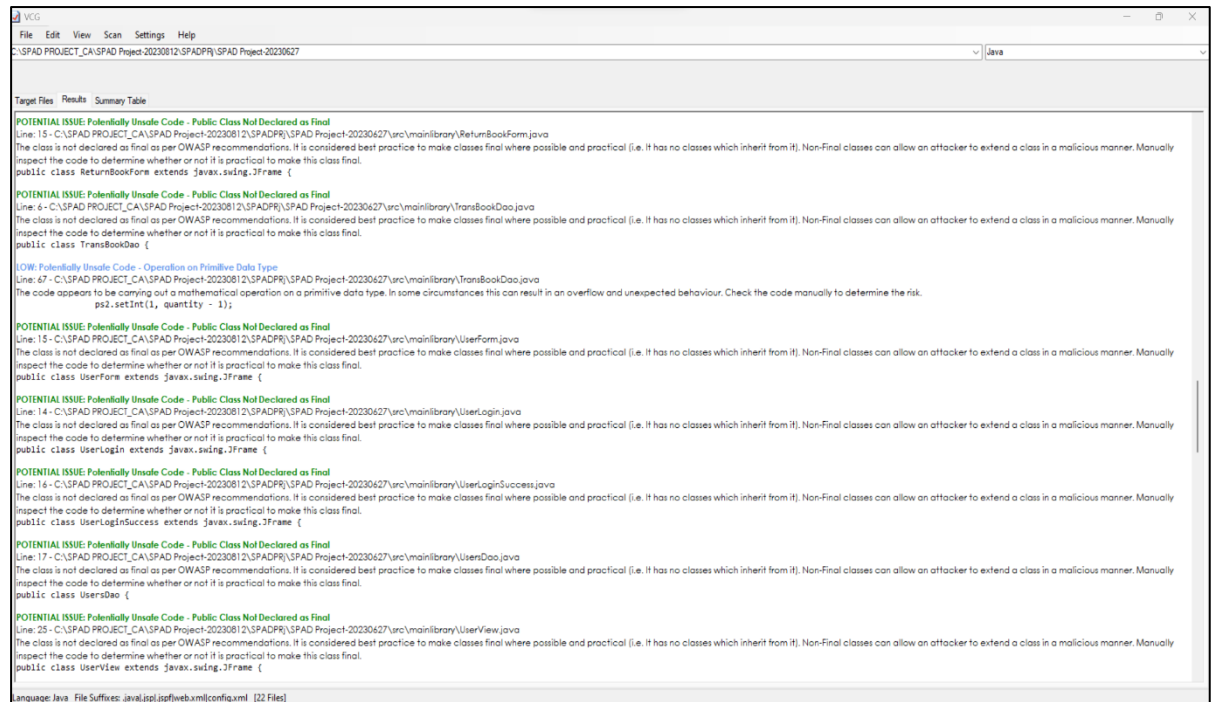


Fig 7. Scanning process by VCG

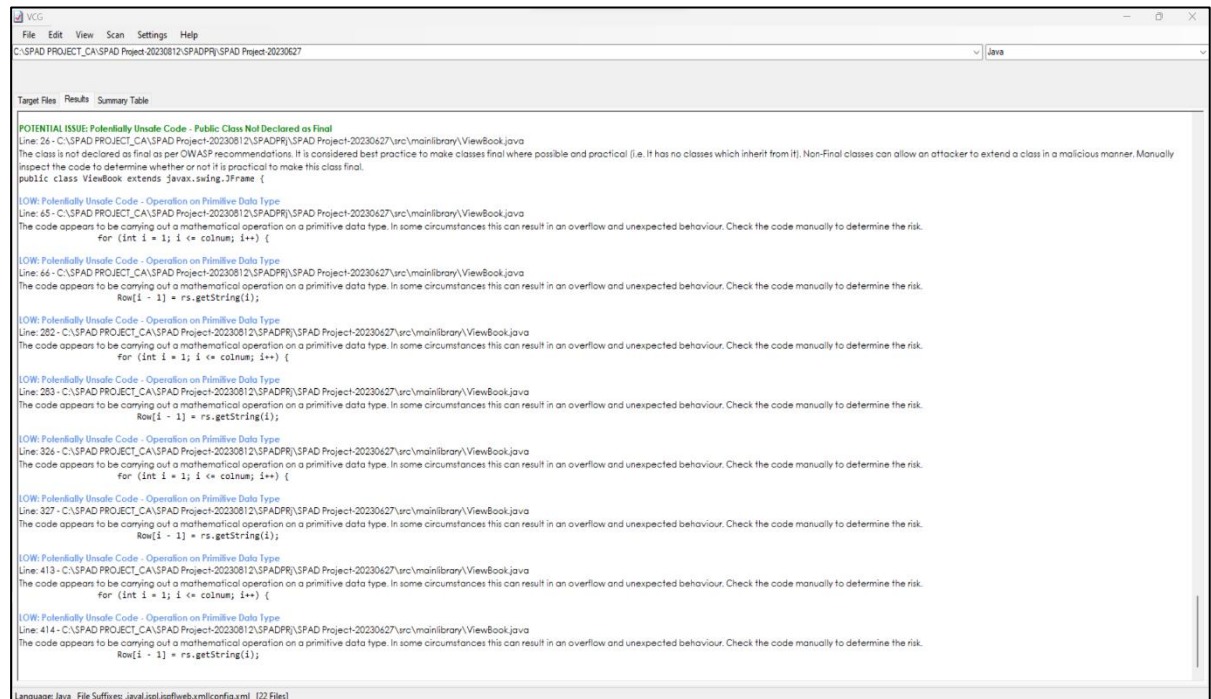


Fig 8. Low potential threats by VCG

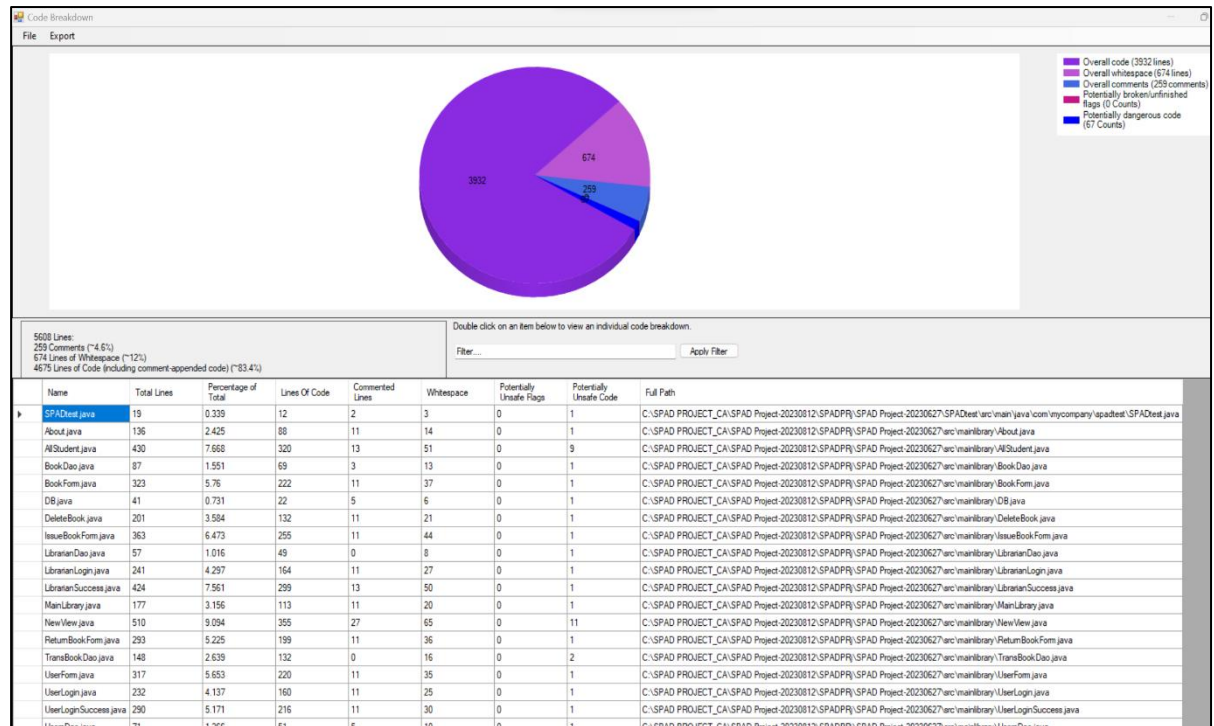


Fig 9. Pie chart of overall project vulnerability

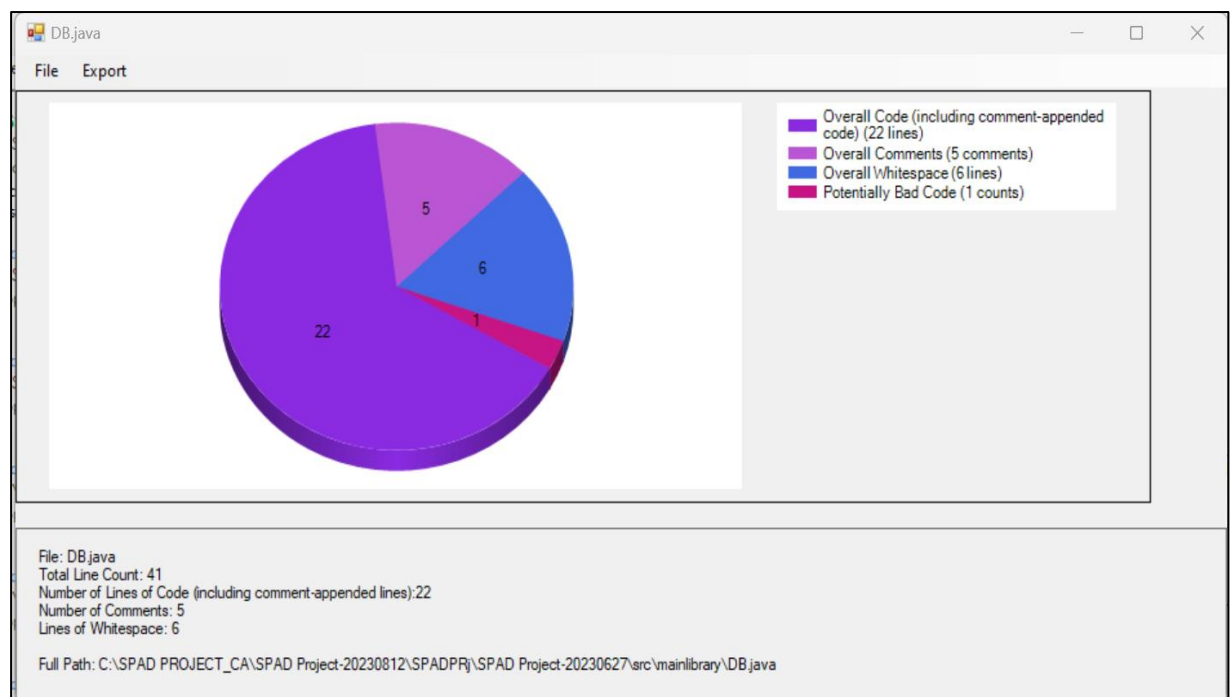


Fig 10. Individual threat description by VCG

Codacy:

Codacy is a software tool that uses automation to do inspections and access the standard of code with respect to each modification and patch request across a wide range of coding languages. It provides feedback on the effects of each committed or patch request, as well as identifies issues related to coding style, complying to appropriate practices, security concerns,

and various other aspects. The system monitors modifications in the distribution of code, code repetition and code complexities. Effectively addressing technical problems involves the optimization of code review processes to minimize the time spent by developers.

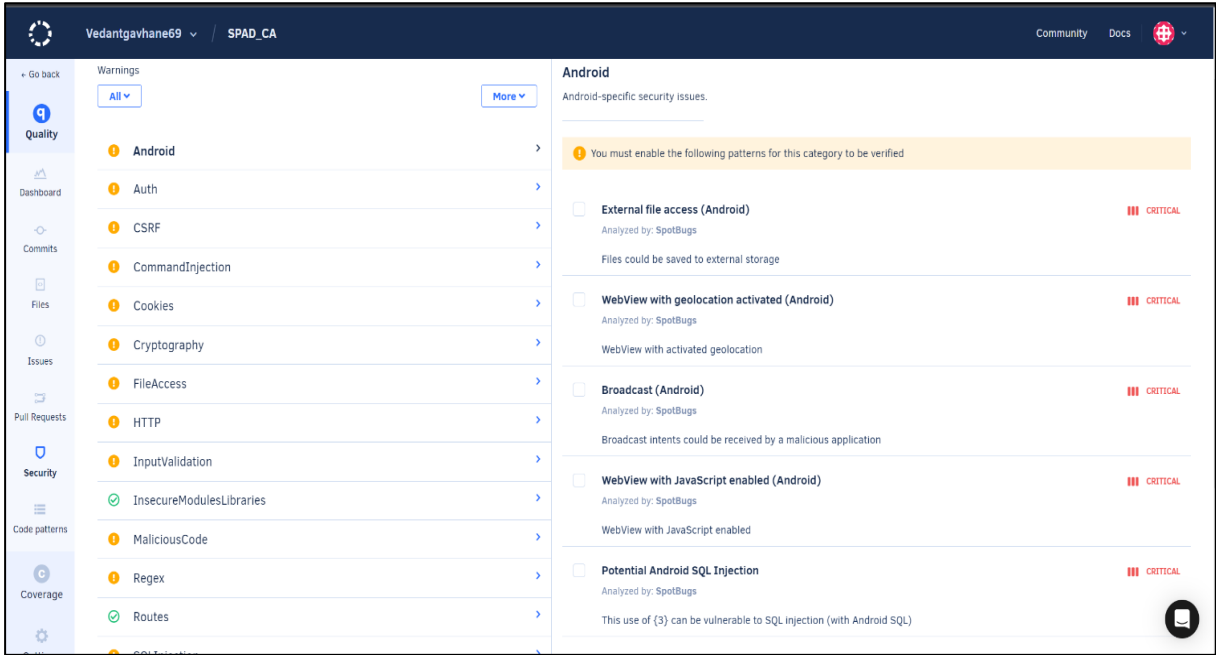


Fig 11. Codacy scanning results.

The screenshot shows the 'Files' section of the Codacy interface. The sidebar is the same as in Fig 11. The main area is titled 'Files' with a dropdown set to 'master'. Below this is a search bar and a message 'Missing some files?'. A table displays the results of the scan, with columns for Grade, Filename, Issues, Complexity, and Duplication. The table lists several files, including build.xml, nbproject/build-impl.xml, nbproject/project.xml, SPADtest/pom.xml, SPADtest/src/main/java/com/mycompany/spadtest/SPADtest.java, SQL/exportdb.sql, src/mainlibrary/About.java, and src/mainlibrary/AllStudent.java. The files are graded from A (green) to F (red) based on their vulnerability score.

Fig 12. Providing each and every file vulnerabilities.

Grade	Filename	Issues	Complexity
C	src / mainlibrary / BookDao.java	13	2
D	src / mainlibrary / BookForm.java	89	7
D	src / mainlibrary / DB.java	8	2
D	src / mainlibrary / DeleteBook.java	53	7
D	src / mainlibrary / IssueBookForm.java	90	7
B	src / mainlibrary / LibrarianDao.java	6	2
D	src / mainlibrary / LibrarianLogin.java	59	7
D	src / mainlibrary / LibrarianSuccess.java	100	8

Fig 13. Critical and low threat identification

Security testing using manual code review.

The practice of manual code review is an essential part of the software development process. During this activity, IT professionals carefully study source code in order to locate flaws, vulnerabilities, and potential areas for improvement. The expertise and analytical integrity of reviewers are put to use in this process, which ultimately results in improved code quality. It assists in the detection of bugs, security risks, and design flaws many automated tools can overlook.

The process of manually reviewing code encourages collaboration among team members, which in turn promotes the sharing of information and maintains consistent coding standards. Reviewers not only assure the dependability and performance of the software by providing insights, suggestions, and best practices, but they additionally encourage learning and growth within the development team. This is because reviewers ensure the growth of the program. In general, the process of manually reviewing code is an essential component of the software development lifecycle that contributes to the production of code that is secure, efficient, and easy to maintain.

Manual code review findings:

Name	Total lines	Potentially unsafe code	Proposed solution
DB.java	15	The presence of a hard-coded password found. Public static string user = "root";	Use encryption and provide a secure password
	20	useServerPrepStmts is configured with a false value and risk to the SQL injection threat.	The false value is needed to be set as a true on the server side.
	25-28	Improper input validation getConnection() method	Fix the input validation for user inputs

	30	SQL injection vulnerability in the <code>Class.forName("com.mysql.jdbc.driver");</code>	Use parameterized queries or prepared statements and fix input validation
	34	Failing to properly terminate the database connection might result in data leakage	Properly terminate the database connection by using <code>con.close()</code> method
AllStudent.java	23	Non-Final classes can allow an attacker to extend a class in a malicious manner class About extends <code>javax.swing.JFrame</code>	Mark the class as final also use composition instead of inheritance
	48-49, 264-265, 305-306	The code appears to be carrying out a mathematical operation on a primitive data type. It can cause overflow and unexpected behaviour for <code>(int i = 1; i <= colnum; i++)</code>	To avoid this issue, use a larger data type, exception handling to handle the overflow condition.
BookDao.java/ BookForm.java	7-17	It has no classes which inherit from it. public class BookDao	By using 'final' keyword it is explicitly stated the BookDao class is not designed to be extended.
DeleteBook.java	14	It has no classes which inherit from it. Public class DeleteBook extends <code>javax.swing.JFrame</code>	To avoid classes to inherit from 'DeleteBook' mark it as a 'final' which secure other classes from extending it.
LibrarianDao.java	5	It has no classes which inherit from it. Public class LibrarianDao	Use composition between 'LibrarianDao' class and other class and mark it as a 'final'.
New view.java	26	Non-Final classes can allow an attacker to extend a class in a malicious manner. Public class NewView extends <code>javax.swing.JFrame</code>	The class can't be created from outside due to its private constructor. Instead, you return a NewView instance from static factory method.
IssueBookForm.java	360	Perhaps 'jLabel7' could be replaced by a local variable <code>private javax.swing.JLabel jLabel7;</code>	Fields with single-method scopes do not depend on the contained object to offer them to other methods. They may work better as local variables in certain approaches.
About.java	89	Reports parameters of methods and constructors that have not referenced them in the method body. Unused formal parameter.	Avoid unused method parameters such as 'evt'.
MainLibrary.java	119	Reports parameters of methods and constructors that have not	Avoid unused method parameters such as 'evt'.

		referenced them in the method body. Unused formal parameter.	
ReturnBookForm.java	291	Some fields have limited scopes restricted to individual methods do not depend on the contained objects for their provision to other methods.	Perhaps 'jLabel8' could be replaced by a local variable.
UserLoginSuccess.java	263	The system recognizes instances in which a local variable is declared and assigned but remains unused.	Avoid unused local variables such as 'status'
ViewBook.java	247	Configurable naming conventions for method declarations. This rule reports method declarations which do not match the regex that applies to their specific kind.	The instances method name 'SearchFieldActionPerformed' dchanged to '[a-z][a-zA-Z0-0]*'

Table No. 1 Manual code review process

4. Security testing performed by using Penetration Testing of Library:

The process of penetration testing facilitates the comprehensive evaluation of risks by examining both potential vulnerabilities and capabilities related to the security of a given system.

1. SQL Injection vulnerability found on User and librarian login page as follows:

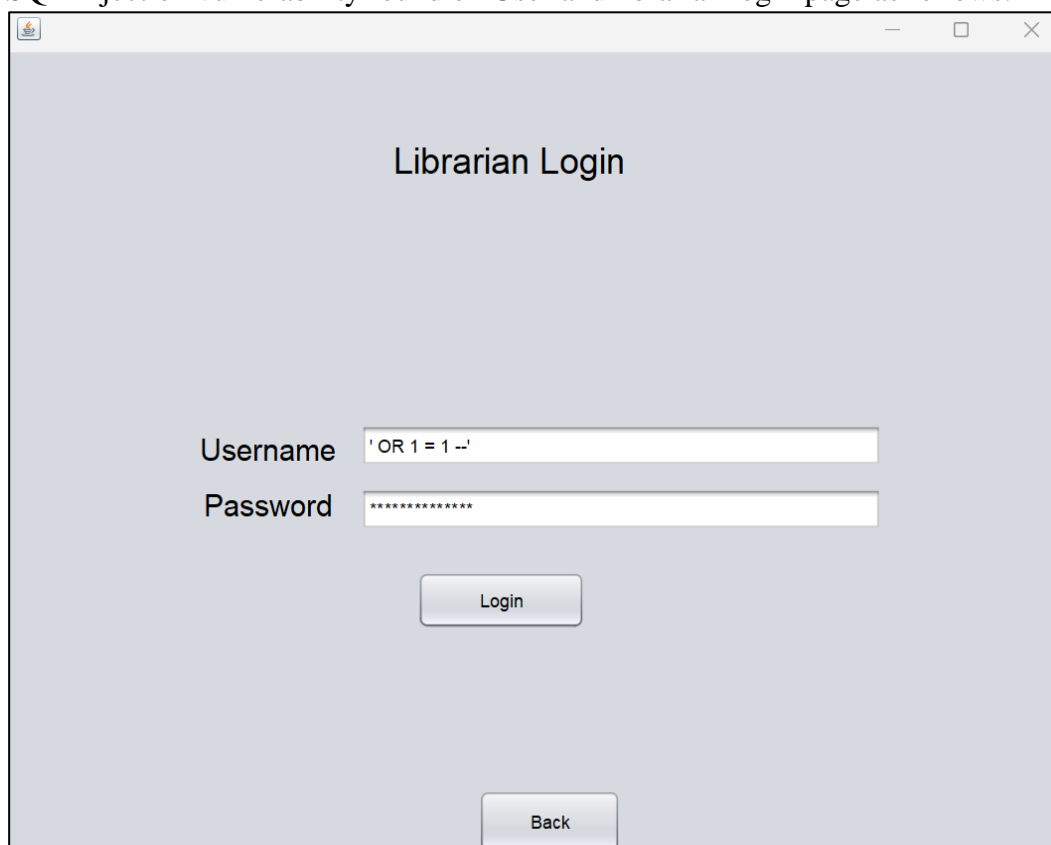


Fig 14. SQL Injection vulnerability on Librarian login page

A screenshot of a web application interface for a librarian. The page has a light gray background. On the left side, there are four input fields: 'Name' with the value 'Leloush Britannia', 'Library ID' with the value '2', 'Email' with the value 'leloush.zero@bitannia.com', and 'Contact No.' which is empty. Below these fields are two buttons: 'Add Book' and 'View Book'. On the right side, there is a vertical stack of buttons: 'View Issued Books', 'Issue Book', 'Return Book', 'Delete Book', 'Add Student', and 'View Students'. At the bottom center, there is a 'LogOut' button. The window has a standard title bar with minimize, maximize, and close buttons.

Fig 15. Successfully login via SQL Injection vulnerability on Librarian login page

A screenshot of a web application interface for a user login. The page has a light gray background. At the top center, the text 'User Login' is displayed. Below it, there are two input fields: 'Username' with the value 'OR 1 = 1 --' and 'Password' with the value '****'. Below these fields is a 'Login' button. At the bottom center, there is a 'Back' button. The window has a standard title bar with minimize, maximize, and close buttons.

Fig 16. SQL Injection vulnerability on User login page

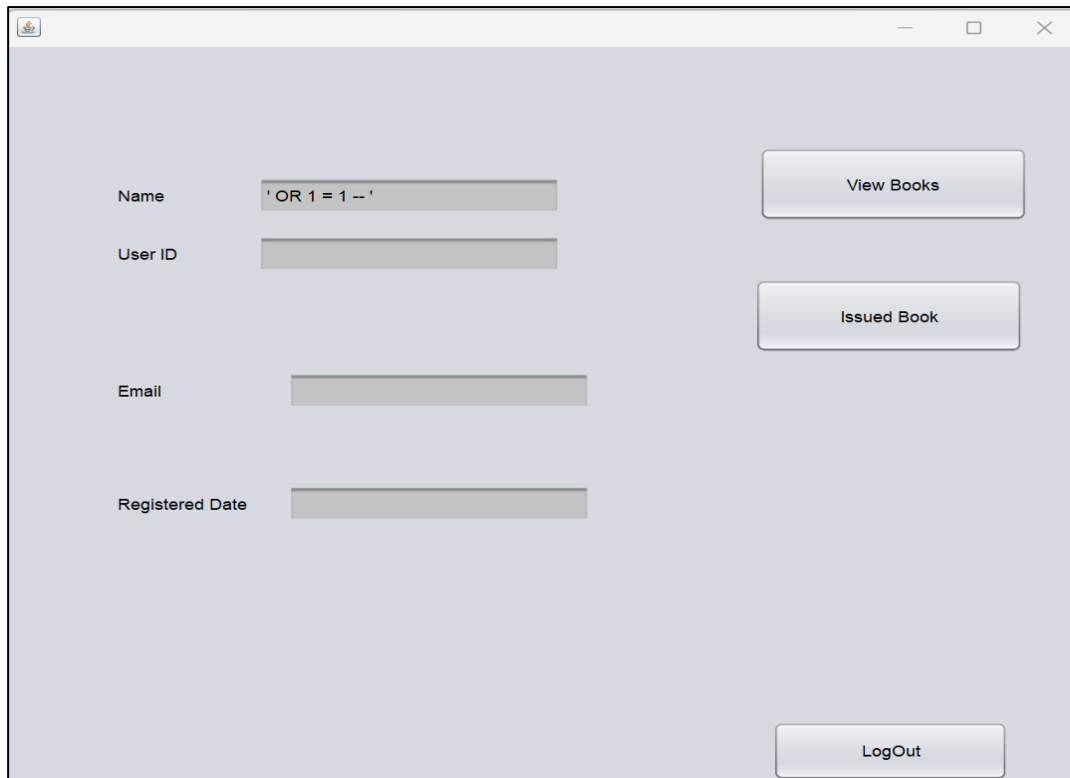


Fig. 17 Successfully login via SQL Injection vulnerability on User login page

2. Hardcoded password vulnerability found in a DB.java form.

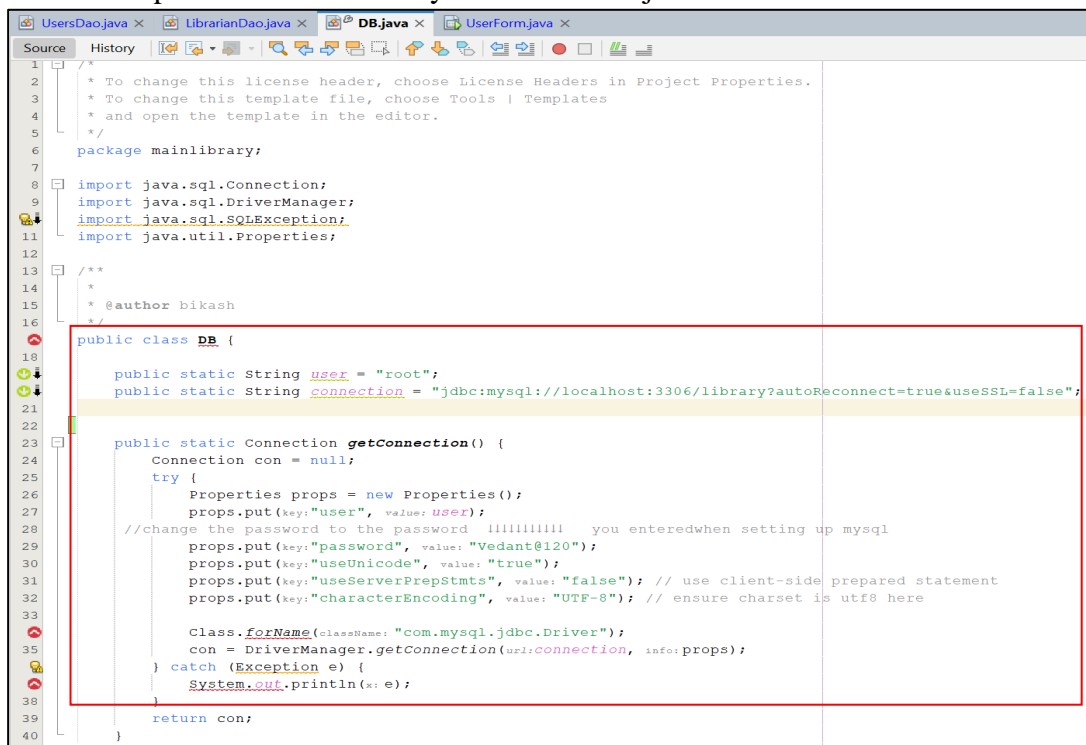


Fig 18. Hardcoded password seen in DB.java.

3. There is a problem with registering the date it accepts the old dd-mm-yy.

The screenshot shows a web application window with a form for issuing a book. The form contains the following fields and values:

- Book ID: 13
- User ID: 4
- Issue Date: 13 - 8 - 2023
- Return Date: 28 - 8 - 2000 (highlighted with a red box)

Below the form are two buttons: "Issue" and "Back". A modal dialog box titled "Book Issued!" is open, displaying a red exclamation mark icon and the text "The Book is Issued!". The dialog has an "OK" button.

Fig 19. Accepting older date formats

4. Improper input validation found in login page.

The screenshot shows a web application window with a form for adding a new user. The form contains the following fields and values:

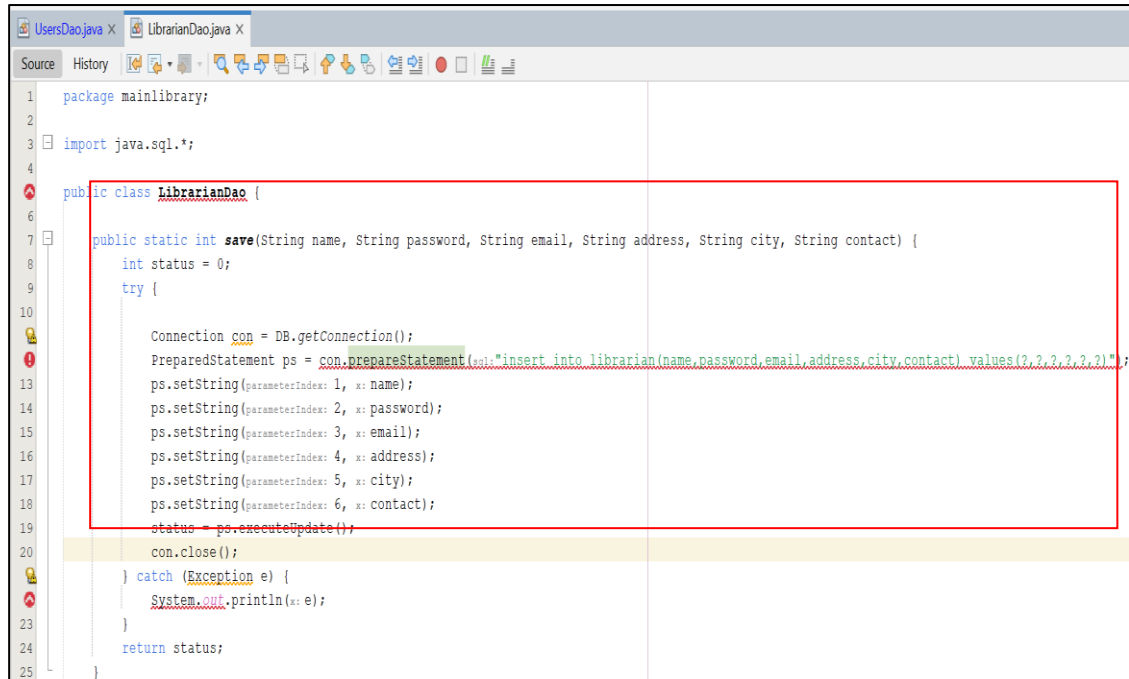
- User Name: @reyna (highlighted with a red box)
- Password: ****
- Position: asda
- Email: asda (highlighted with a red box)
- Academic Program: adas
- Year: 1234

Below the form are two buttons: "ADD U" and "Back". A modal dialog box titled "Adding New User!" is open, displaying a red exclamation mark icon and the text "User is Added Successfully!". The dialog has an "OK" button.

Fig 20. Accepting improper input validation found in login page.

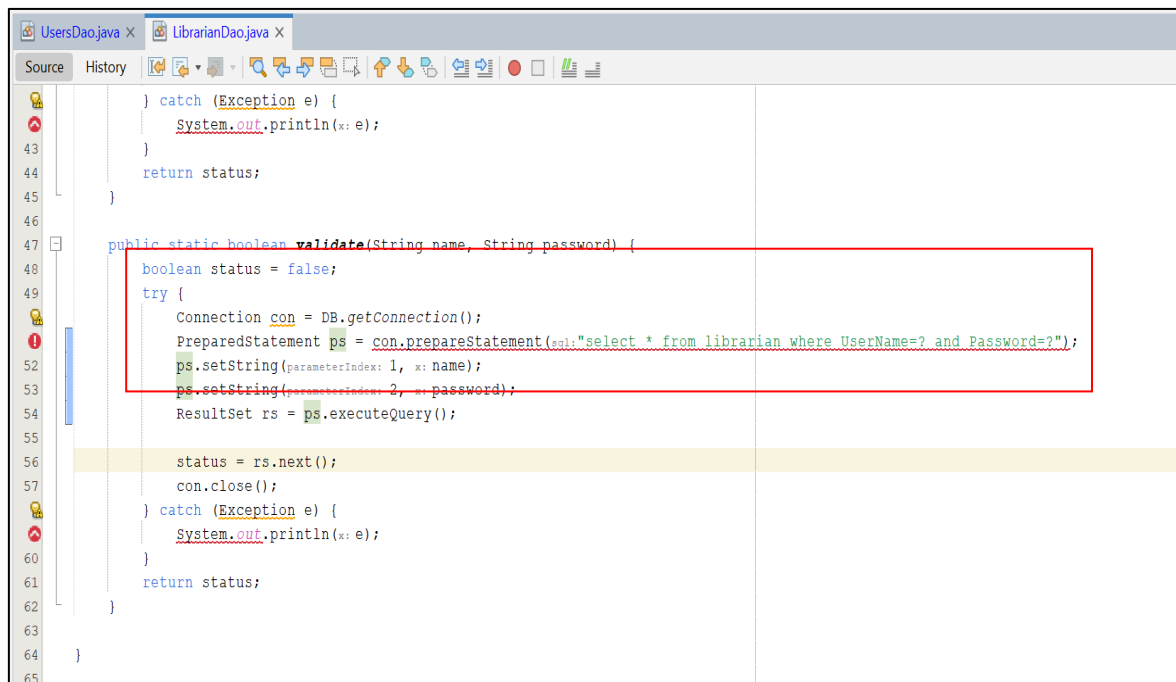
4.1) Proposed Solution to the issues we found in penetration testing:

- 1) Vulnerable code to SQL Injection on LibrarianDao.java. It accepts the string format and saves in the database.



```
1 package mainlibrary;
2
3 import java.sql.*;
4
5 public class LibrarianDao {
6
7     public static int save(String name, String password, String email, String address, String city, String contact) {
8         int status = 0;
9         try {
10
11             Connection con = DB.getConnection();
12             PreparedStatement ps = con.prepareStatement("insert into librarian(name,password,email,address,city,contact) values(?,?,?,?,?,?)");
13             ps.setString(parameterIndex: 1, x: name);
14             ps.setString(parameterIndex: 2, x: password);
15             ps.setString(parameterIndex: 3, x: email);
16             ps.setString(parameterIndex: 4, x: address);
17             ps.setString(parameterIndex: 5, x: city);
18             ps.setString(parameterIndex: 6, x: contact);
19             status = ps.executeUpdate();
20             con.close();
21         } catch (Exception e) {
22             System.out.println(x: e);
23         }
24         return status;
25     }
26 }
```

Fig 21. Vulnerable source code of SQL Injection on LibrarianDao.java



```
43     } catch (Exception e) {
44         System.out.println(x: e);
45     }
46     return status;
47 }
48
49 public static boolean validate(String name, String password) {
50     boolean status = false;
51     try {
52         Connection con = DB.getConnection();
53         PreparedStatement ps = con.prepareStatement("select * from librarian where UserName=? and Password=?");
54         ps.setString(parameterIndex: 1, x: name);
55         ps.setString(parameterIndex: 2, x: password);
56         ResultSet rs = ps.executeQuery();
57
58         status = rs.next();
59         con.close();
60     } catch (Exception e) {
61         System.out.println(x: e);
62     }
63     return status;
64 }
65 }
```

Fig 22. Used a PreparedStatement to prevent SQL injection vulnerability.

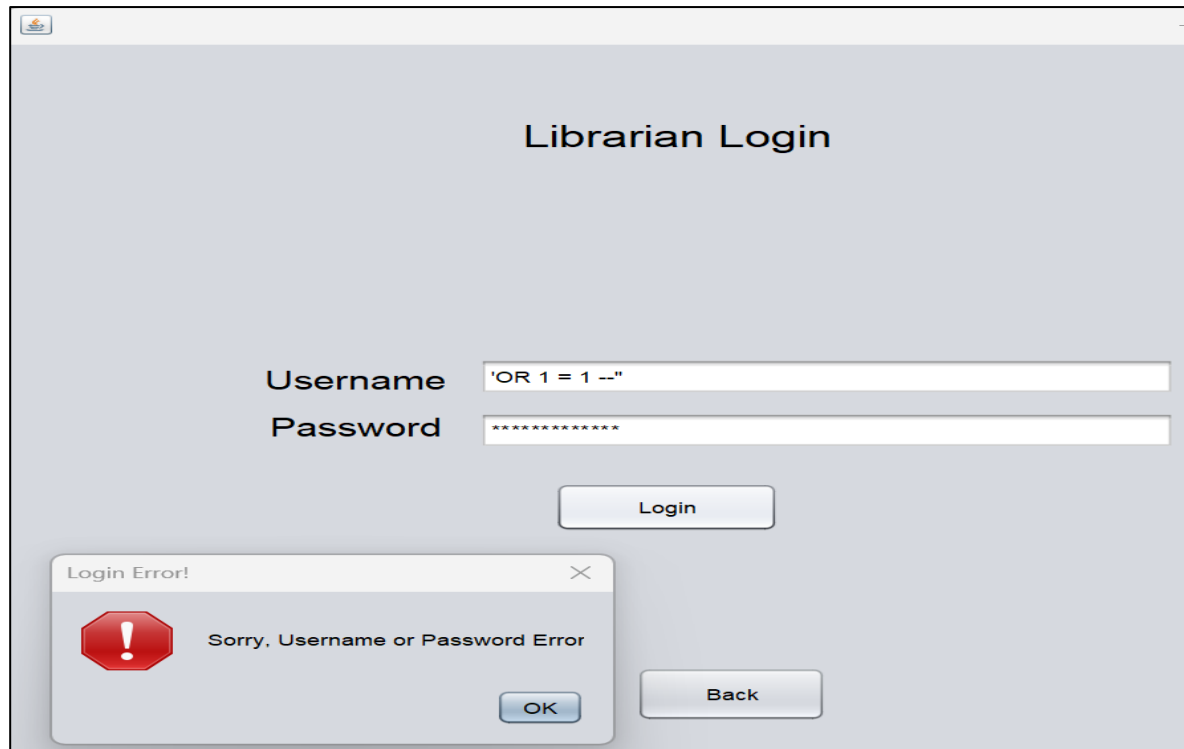


Fig 23. Successfully prevented SQL injection vulnerability on librarian login page

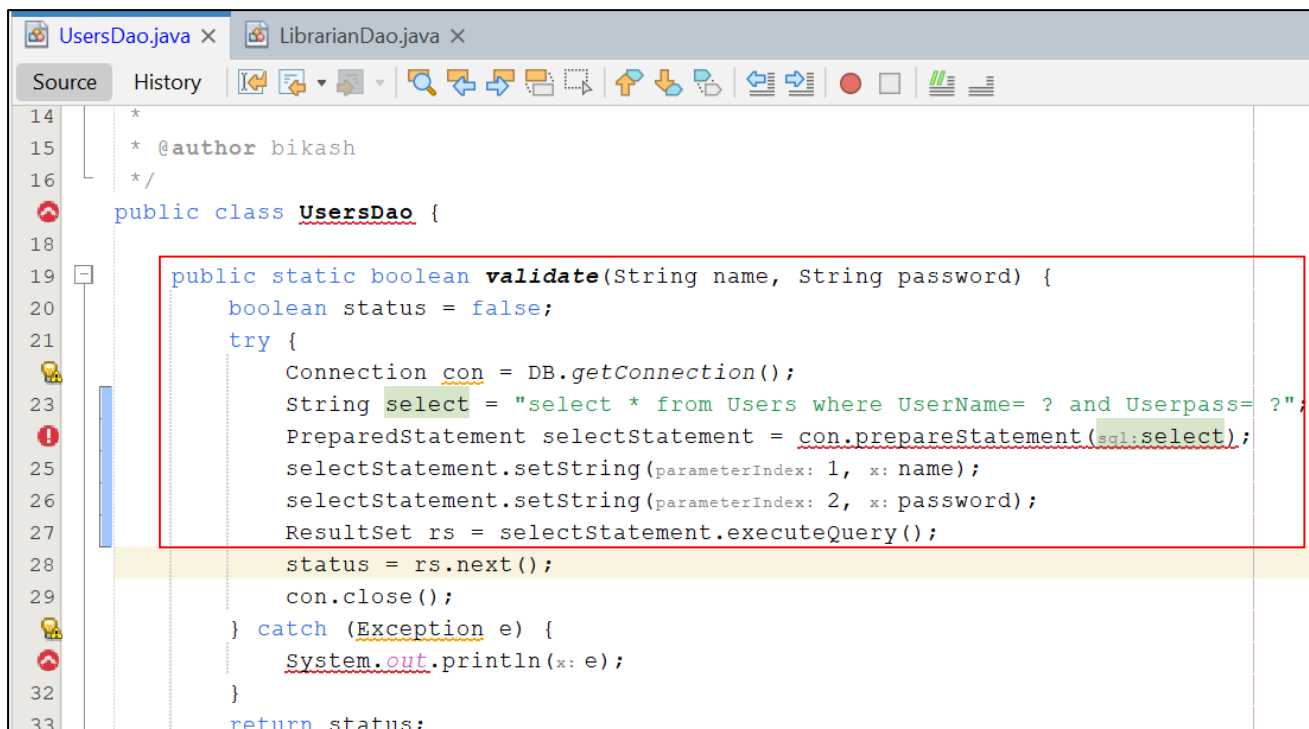


Fig 24. Fixed SQL injection vulnerability to the User login page

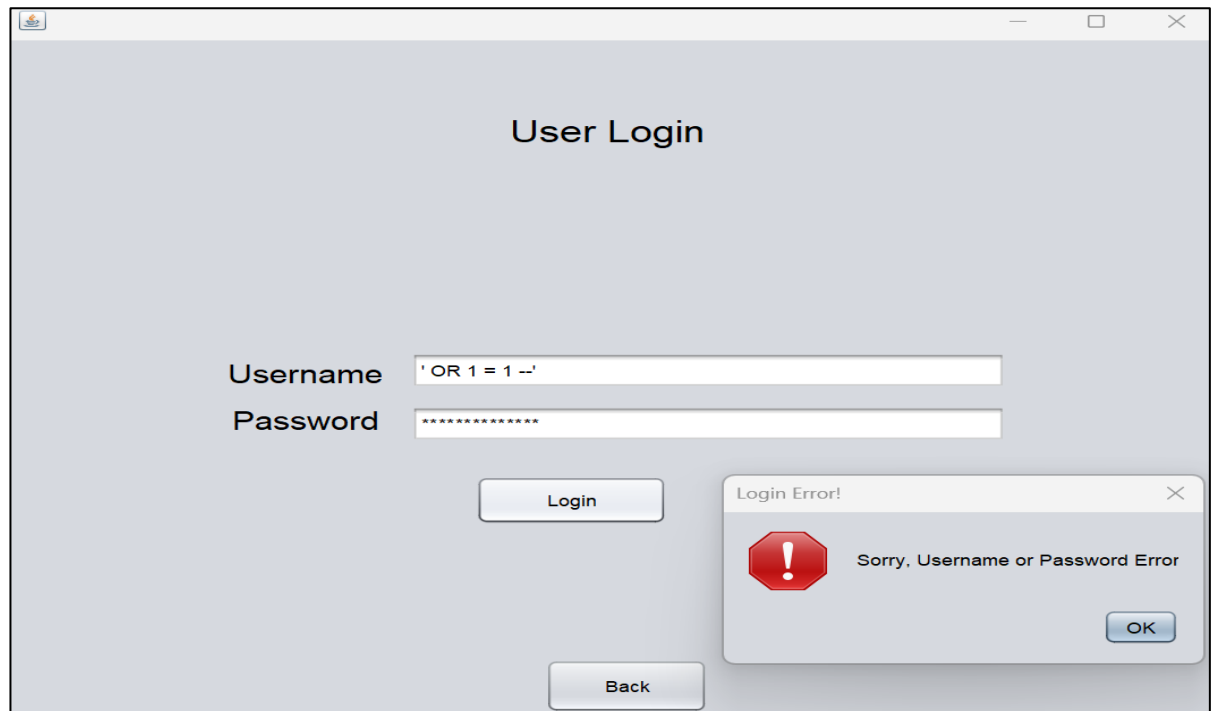


Fig 25. After executing the SQL command on user login page, it prevented from threat.

2) The password is storing in a plaintext is found in the database.

To prevent this vulnerability, we have used the predefined MySQL methods to store our password in an encrypted format. For this the md5 function we used to store plaintext password in the encrypted text, and we assign total 35 char length to the password.

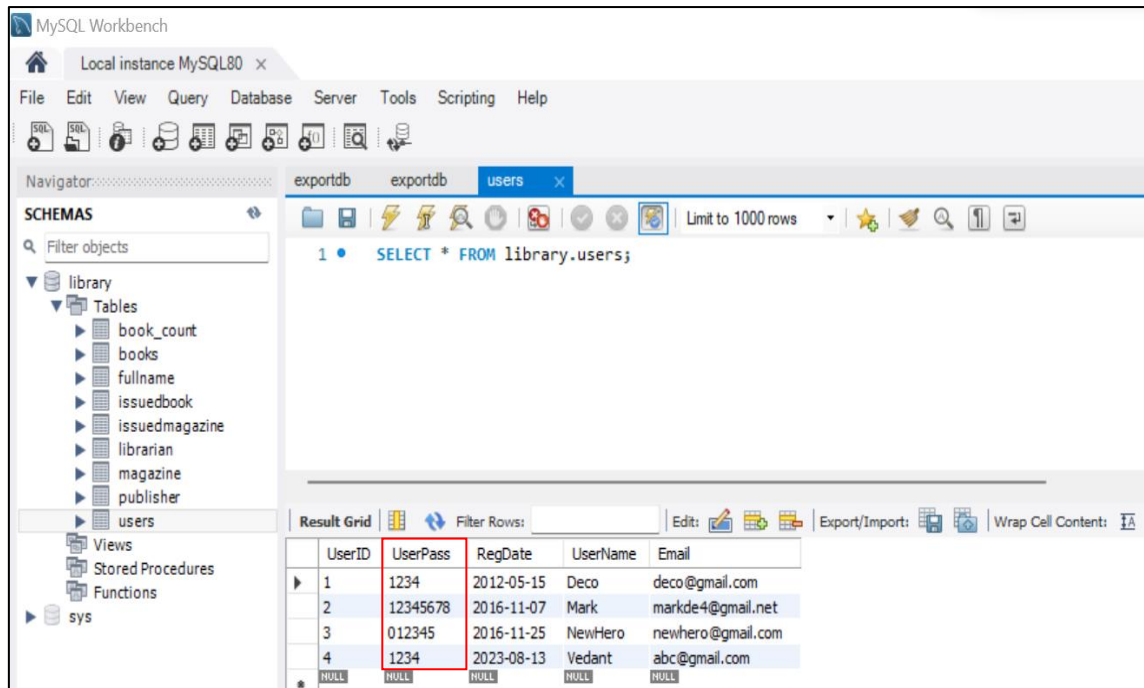


Fig 26. Storing a plaintext password in DB.

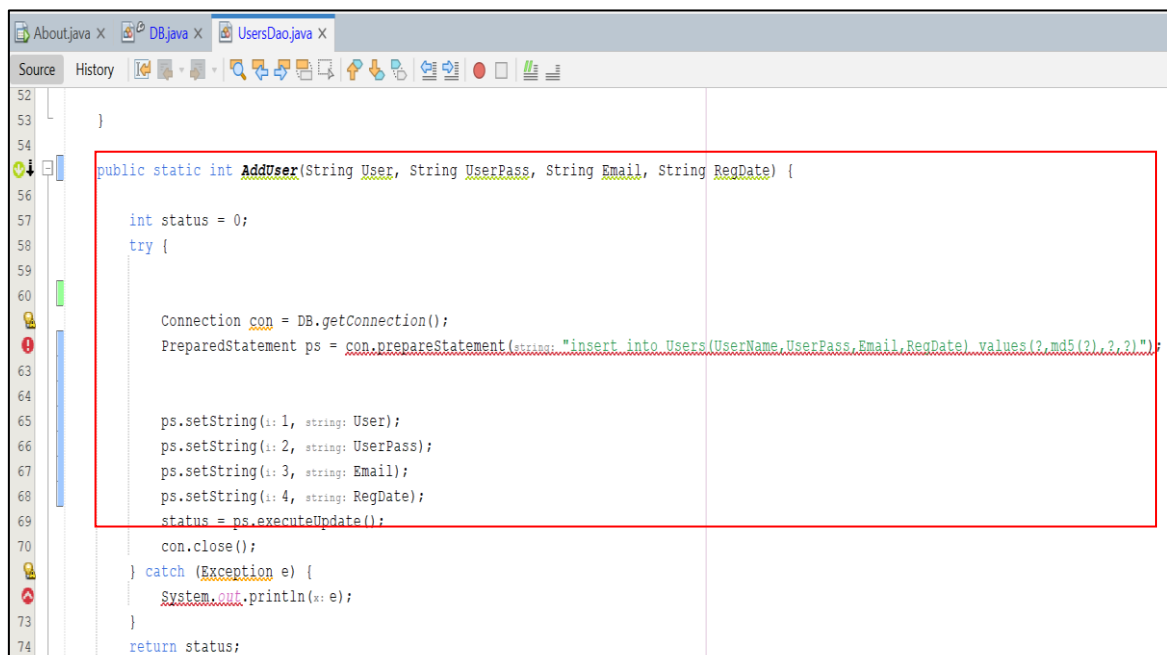


Fig 27. Fixed plaintext password using md5 method.

	UserID	userpass	RegDate	UserName	Email
▶	1	1234	2012-05-15	Deco	deco@gmail.com
	2	12345678	2016-11-07	Mark	markde4@gmail.net
	3	012345	2016-11-25	NewHero	newhero@gmail.com
	11	9ddd8e6a7395d0b2a43301f47f615612	2023-08-14	vedant	ved@gmail.com
	12	b0aa651c991deca550252ed6cbba99ba	2023-08-14	Harsh	harsh@gmail.com
	13	13a4a200512058e39ca485c306372692	2023-08-14	Viraj	vij@gmail.com
	14	23734cd52ad4a4fb877d8a1e26e5df5f	2023-08-14	@reyna	asda
	15	5e841196ef7b42e9f77bb830309f4455	2023-08-14	vikrant	vikrant@gmail.com
*	NULL	NULL	NULL	NULL	NULL

Fig 28. Encrypted password in DB

- 3) We found a hardcoded password in the DB.java form the password should not be directly accessible in the source code. To fix that issue we created a normal txt file in the mainlibrary file. In that file we set our user id and password and that file is directly executed in DB.java file by using getproperty function.

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package mainlibrary;
7
8  import java.sql.Connection;
9  import java.sql.DriverManager;
10 import java.sql.SQLException;
11 import java.util.Properties;
12
13 /**
14 *
15 * @author bikash
16 */
17 public class DB {
18
19     public static String user = "root";
20     public static String connection = "jdbc:mysql://localhost:3306/library?autoReconnect=true&useSSL=false";
21
22
23     public static Connection getConnection() {
24         Connection con = null;
25         try {
26             Properties props = new Properties();
27             props.put(key:"user", value: user);
28             //change the password to the password !!!!!!!! you entered when setting up mysql
29             props.put(key:"password", value: "Vedant@120");
30             props.put(key:"useUnicode", value: "true");
31             props.put(key:"useServerPrepStmts", value: "false"); // use client-side prepared statement
32             props.put(key:"characterEncoding", value: "UTF-8"); // ensure charset is utf8 here
33
34             Class.forName(className: "com.mysql.jdbc.Driver");
35             con = DriverManager.getConnection(url:connection, info:props);
36         } catch (Exception e) {
37             System.out.println(x: e);
38         }
39         return con;
40     }
41 }

```

Fig 29. Hardcoded password present in the source code of DB.java

```

public class DB {
    public static String user = "root";
    public static String connection = "jdbc:mysql://localhost:3306/library?autoReconnect=true&useSSL=false";

    public static Connection getConnection() {
        Connection con = null;
        try {
            Properties pr = new Properties();
            FileInputStream input = new FileInputStream("src\\mainlibrary\\cred.properties");
            pr.load(input);
            input.close();

            // Modified Code
            String user = pr.getProperty(key:"user");
            String password = pr.getProperty(key:"password");

            pr.put(key:"useUnicode", value: "true");
            pr.put(key:"useServerPrepStmts", value: "false");
            pr.put(key:"characterEncoding", value: "UTF-8");

            Class.forName(className: "com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url:connection, info:pr);
        } catch (Exception e) {
            System.out.println(e);
        }
        return con;
    }
}

```

Fig 30. Fixed hardcoded source code file with getproperty function.

4) There is an improper input validation also found on the user login page:

The username and password are stored into a normal plaintext. To fix this vulnerability the username should be provided in a lowercase and the password is stored with some special characters like @, #, / [_ -]. For the user name we set [a-z]+\$ this format and for the password ^(?=.*[0-9])(?=.*[a-z])(?=.*[A-and (?=.*[@#\$%^&+=]) and (?=\\S+\$).{8,20}\$, for the email we set [A-Za-z0-9+_-] +@.(+)\$ these pattern will provide a robust security and every user will be authenticate.

```

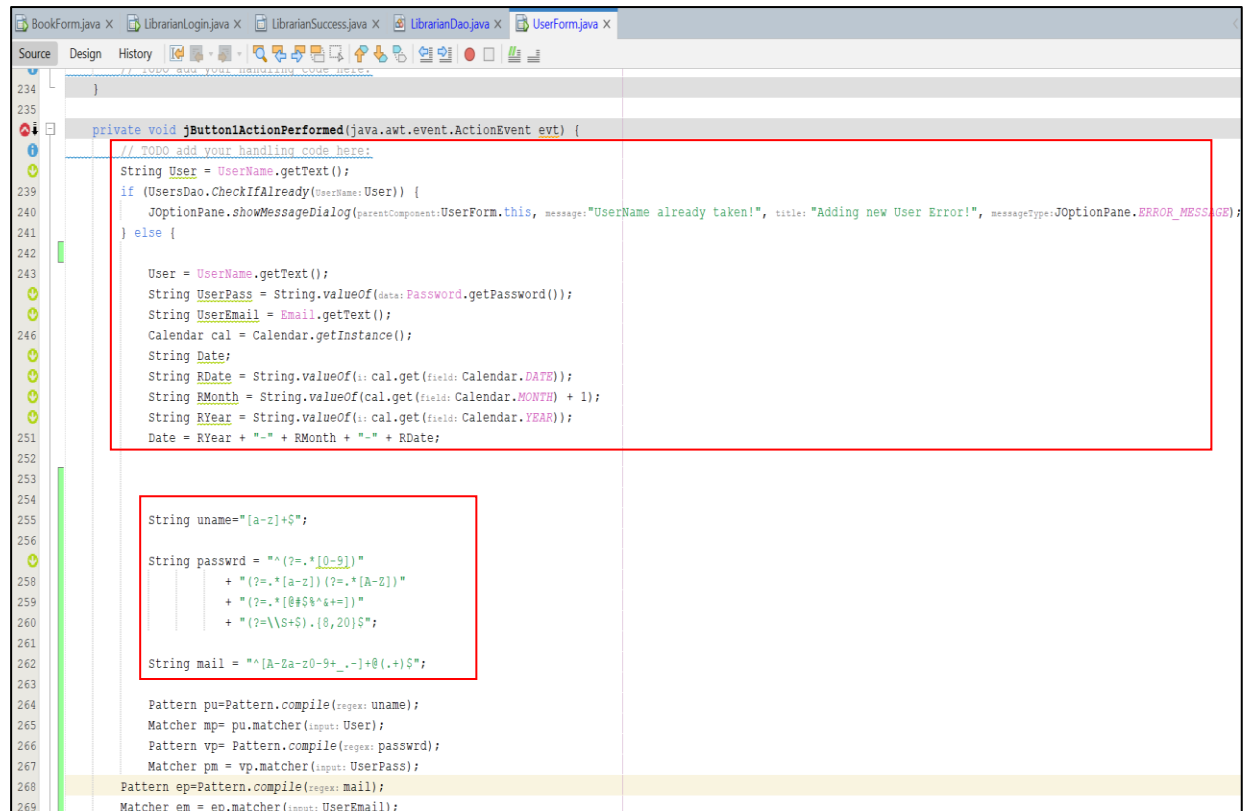
private void EmailActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String User = UserName.getText();
    if (UsersDao.CheckIfAlready(UserName: User)) {
        JOptionPane.showMessageDialog(parentComponent: UserForm.this, message: "UserName already taken!", title: "Adding new User Error!",
    } else {
        User = UserName.getText();
        String UserPass = String.valueOf(data: Password.getPassword());
        String UserEmail = Email.getText();
        Calendar cal = Calendar.getInstance();
        String Date;
        String RDate = String.valueOf(cal.get(field: Calendar.DATE));
        String RMonth = String.valueOf(cal.get(field: Calendar.MONTH) + 1);
        String RYear = String.valueOf(cal.get(field: Calendar.YEAR));
        Date = RYear + "-" + RMonth + "-" + RDate;

        if (UsersDao.AddUser(User, UserPass, UserEmail, Date) != 0) {
            JOptionPane.showMessageDialog(parentComponent: UserForm.this, message: "User is Added Successfully!", title: "Adding New User!");
            UserName.setText("");
            Password.setText("");
            Email.setText("");
            Position.setText("");
            Program.setText("");
            Year.setText("");
        } else {
            JOptionPane.showMessageDialog(parentComponent: UserForm.this, message: "Unable to add new User", title: "Adding new User Error!");
        }
    }
}

```

Fig 31. Improper Input sanitization source code



```
234 }
235
236 private void jButtonActionPerformed(java.awt.event.ActionEvent evt) {
237     // TODO add your handling code here:
238     String User = UserName.getText();
239     if (UsersDao.CheckIfAlready(Username:User)) {
240         JOptionPane.showMessageDialog(parentComponent:UserForm.this, message:"UserName already taken!", title:"Adding new User Error!", messageType:JOptionPane.ERROR_MESSAGE);
241     } else {
242
243         User = UserName.getText();
244         String UserPass = String.valueOf(data: Password.getPassword());
245         String UserEmail = Email.getText();
246         Calendar cal = Calendar.getInstance();
247         String Date;
248         String RDate = String.valueOf(cal.get(field: Calendar.DATE));
249         String RMonth = String.valueOf(cal.get(field: Calendar.MONTH) + 1);
250         String RYear = String.valueOf(cal.get(field: Calendar.YEAR));
251         Date = RYear + "-" + RMonth + "-" + RDate;
252
253         String uname="[a-z]+$";
254
255         String passwd = "(?=.*[0-9])"
256             + "(?=.*[a-z])(?=.*[A-Z])"
257             + "(?=.*[!@#$%^&*+=])"
258             + "(?=\S+$).{8,20}$";
259
260         String mail = "[A-Za-z0-9+_.-]@(.+)$";
261
262         Pattern pu=Pattern.compile(regex: uname);
263         Matcher mp= pu.matcher(input: User);
264         Pattern vp= Pattern.compile(regex: passwd);
265         Matcher pm = vp.matcher(input: UserPass);
266         Pattern ep=Pattern.compile(regex: mail);
267         Matcher em = ep.matcher(input: UserEmail);
```

Fig 32. Fixed input sanitization with proper formats

Conclusion

The following is an in-depth review that establishes connections between the different sections of the report and clarified how they are interconnected. The study begins with a review of the current innovations in programming paradigms and the field of Java security. The given information offers a contextual understanding of imperative, declarative, and object-oriented methodologies that are essential to the development of Java applications. The study also covered the progression of security features in various versions of Java. The present condition of application security testing was later analysed. This study examines various methodologies, including penetration testing, static analysis, dynamic analysis, and fuzzing. The discuss also included the integration of security measures within the Software Development Life Cycle (SDLC). This process provided the foundation to perform evaluations. We have done the application security testing by using different tools like SonarLint, VCG, and Codacy. In addition, a manual code review was conducted in order to identify potential vulnerabilities such as SQL injection, hardcoded passwords, improper input validation, and plaintext password stored in DB. In order to fix those issues, multiple fixes were suggested by using the most up-to-date Java libraries and security features that were previously proposed. This improvement will increase the overall security approach of the programme.

References

- [1] "Introduction of programming paradigms," GeeksforGeeks, <https://www.geeksforgeeks.org/introduction-of-programming-paradigms/> (accessed Aug. 14, 2023).
- [2] "Java version history," Wikipedia, https://en.wikipedia.org/wiki/Java_version_history (accessed Aug. 14, 2023).
- [3] "JDK 20 Release Notes," JDK 20 release notes, important changes, and information, <https://www.oracle.com/java/technologies/javase/20-relnote-issues.html> (accessed Aug. 14, 2023).
- [4] L. Vileikis, "Security in mysql: Part one," Simple Talk, <https://www.red-gate.com/simple-talk/databases/mysql/security-in-mysql-part-one/> (accessed Aug. 14, 2023).
- [5] "Java SE Technologies - Security," Java SE Security, <https://www.oracle.com/java/technologies/javase/javase-tech-security.html> (accessed Aug. 14, 2023).
- [6] "Application security testing tools: Web app security testing," Application Security Testing Tools | Web App Security Testing, <https://www.contrastsecurity.com/glossary/application-security-testing> (accessed Aug. 14, 2023).
- [7] "What is penetration testing: Step-by-step process & methods: Imperva," Learning Center, <https://www.imperva.com/learn/application-security/penetration-testing/> (accessed Aug. 14, 2023).
- [8] "What is Fuzz Testing and how does it work?," Synopsys, <https://www.synopsys.com/glossary/what-is-fuzz-testing.html> (accessed Aug. 14, 2023).
- [9] R. Bellairs, "What is static analysis? Static Code Analysis Overview," Perforce Software, <https://www.perforce.com/blog/sca/what-static-analysis> (accessed Aug. 14, 2023).
- [10] J. Schmitt, "Sast vs dast: What they are and when to use them," CircleCI, <https://circleci.com/blog/sast-vs-dast-when-to-use-them/> (accessed Aug. 14, 2023).
- [11] "Secure application development and modern defenses," Miscellaneous Ramblings of An Ethical Hacker, <https://www.rafaybaloch.com/2017/06/secure-application-development-and.html> (accessed Aug. 14, 2023).
- [12] C. Guindon, "SonarLint - fix issues before they exist," The Eclipse Foundation, https://www.eclipse.org/community/eclipse_newsletter/2019/march/sonarlint.php (accessed Aug. 14, 2023).