# Secure Banking Application Project

**Student Number: L00172671**

## Introduction

The purpose of this report is to detail the design and development of the Secure Banking Application (OGB), created as part of my assignment for Secure Programming. The project focuses on securing user data and protecting the system against potential security threats. In this report, I will describe the application features and functionality, outline the technologies and tools used, and explain the secure coding practices implemented during development. Additionally, I will assess the robustness and security of the system by testing key features, highlighting areas for improvement, and providing insights gained from the development process.

## Creating the project

For this project, I decided to use Visual Studio Code (VS Code**)**, version 1.98, as my primary development environment. VS Code is a versatile and lightweight editor that supports multiple programming languages and offers great flexibility through extensions. I ran the project using Java version 23.1.0, executed through the terminal in VS Code. This setup was smooth and straightforward, with no significant issues during installation or configuration. To keep things organized, I created a dedicated folder for the project, with subfolders for documentation and code, ensuring everything was properly structured for easy access and management.

In terms of hardware, I used my personal laptop for most of the development. My laptop has the following specifications:

* **Device Name:** LAPTOP-0EPH4T90
* **Processor:** 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
* **RAM:** 8.00 GB (7.80 GB usable)
* **System Type:** 64-bit operating system, x64-based processor
* **OS:** Windows 11 Home, Version 24H2

I also made use of the ATU machines occasionally while in class.

Additionally, I used GitHub for version control and to back up my work. This allowed me to track changes to my work and add collaborators so that my lecturer can mark and view my work.

## Functionality

The project is a terminal-based banking system designed to run in Visual Studio Code. It allows users to create accounts, log in securely with multi-factor authentication (MFA), check their balances, deposit funds, and make withdrawals. The system ensures data persistence, enforces security measures, and provides a user-friendly experience for seamless banking transactions.

A screen shot of a computer

AI-generated content may be incorrect.

**Figure 1: Main Menu**

When creating an account, the system will prompt the user to enter their name and set a password. The password must be 6-10 characters long and include at least one uppercase letter, one lowercase letter, one number, and one special character. If these requirements are not met, the system will prevent account creation and prompt the user to try again in a loop until a valid password is provided. (‘Java Code Example: check password strength’, no date)

# Validation

**A computer code with text

AI-generated content may be incorrect.**

**Figure 2: Password Validation Code**

Each user account is stored in a relational SQL database called “bank\_system”, which ensures that all account-related information and data are preserved even if the system is shut down or restarted. This persistent storage allows the system to retain user data and ensure continuity across sessions.(*Check the Strength of a Password Using Java*, 2023)

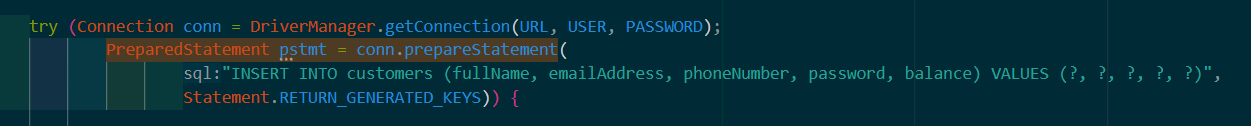
The system is designed to work with SQL Workbench version 8.0.41, the version we were taught to use as part of our course curriculum.

A screenshot of a computer program

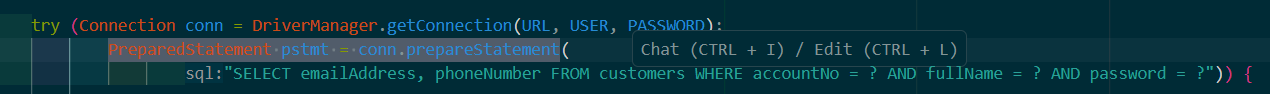
AI-generated content may be incorrect.

**Figure 3: Creating the Database**

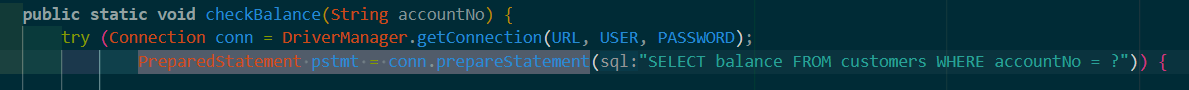
As illustrated in **Figure 3**, the database serves as a secure storage system for member details, including names and account numbers. Users can create accounts, log in with their credentials, and receive two distinct one-time passwords (OTPs) for identity verification. To enhance security, the program implements prepared statements, effectively preventing SQL injection attacks, and enforces strong password validation during account creation. Once authenticated, users gain access to essential banking functionalities, such as checking their balance, making deposits, withdrawing funds, and logging out.

To prevent SQL injection in my system, I used prepared statements, which separate SQL queries from user input by using placeholders (?). This ensures that input values are treated strictly as data rather than executable SQL commands. By binding user inputs safely within parameterizedqueries, the system effectively neutralizes injection attempts, making it secure against unauthorized database manipulation. Examples are shown in **Figure 4-5** below. (*PreparedStatement | JDBC Tutorial for Beginners - YouTube*, 2021)

**Figure 4: Account Creation Protection**

****

**Figure 5: login() Protection**

****

**Figure 6: checkBalance() Protection**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**Figure 7: updateBalance() Protection**

Creating an account involves using the method shown in **Figure 8**. The code utilizes basic Java functions to prompt the user to enter a:

* Full name
* Email address
* Phone Number
* Password
* Initial Deposit

The password is then validated through the method outlined in **Figure 2** to ensure it meets the necessary security requirements. Each newly created account is assigned a unique account number.

A screen shot of a computer program

AI-generated content may be incorrect.

**Figure 8: Account Creation**

# Static Analysis

I have integrated static analysis tools into my Visual Studio Code environment to proactively identify potential errors that might otherwise go unnoticed by my system. As illustrated in the **Figure 9** below, these tools highlight possible bugs in the code without requiring execution. The static analysis tools I have implemented include Snyk (*Snyk Developer Security Platform | AI-powered AppSec Tool | Snyk*, no date) and SonarQube (*SonarQube for Visual Studio 2022 - Visual Studio Marketplace*, no date) , both of which provide valuable insights into code quality and security.A screen shot of a computer program

AI-generated content may be incorrect.

**Figure 9: Static Analysis Tools Highlighting errors**

# Unit Tests

To ensure that individual components of the codebase function correctly in isolation, a dedicated tester class was implemented to facilitate structured and repeatable unit testing. This class acts as a centralized environment where specific methods, functions, or logic blocks where tested independently from the rest of the system. As seen in **Figure 10** below, the unit tests were implemented on most of the major components of the system. (Miecznik, 2023). After some trial and error each of the components of the system passed the Unit Testing phase.

A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 10: Unit Tests**

# Exception Handling

As with any robust system, exception handling plays a critical role in ensuring that unexpected issues do not cause the entire application to crash. Instead of halting execution, exceptions are caught and managed gracefully to maintain a smooth user experience and reliable system behaviour. (*Java Exceptions (Try...Catch)*, no date)

In this project, several specific exceptions are handled strategically across the codebase:

* SQLException – Used to catch errors related to database access
* MessagingException – Handles errors in email or message delivery (Didn’t end up needing this one).
* NoSuchAlgorithmException – Catches cases where a requested cryptographic algorithm is not available in the environment, preventing security features from silently failing.
* InvalidKeySpecException – Manages errors during key generation or conversion, especially in password encryption.

These exception handling mechanisms help log issues and allow fallbacks, all without compromising the stability of the banking system.

A blue background with text

AI-generated content may be incorrect.

**Figure 11: SQLException**



**Figure 12: MessagingException**



**Figure 13: NoSuchAlgorithmException + InvalidKeySpecException**

# Immutable

An access modifier is a keyword in Java that defines who can access certain parts of the code. It helps control the visibility and accessibility of classes, methods, and variables, and is a fundamental part of object-oriented programming for enforcing encapsulation and security within a system.

In this program, the access modifiers used include:

* private
* public
* Getter and setter methods (used to access private data safely)

Fields such as accountNumber, accountHolder, and balance are declared as private, which means they cannot be accessed directly from outside the BankAccount class. This is a core principle of encapsulation, where the internal state of an object is hidden from external classes. Instead of allowing direct access to these fields, the class provides controlled, indirect access through public methods.

As shown in **Figure 14**, marking variables as private ensures that they are protected from unintended or unauthorized modifications. This promotes data integrity and reduces the risk of bugs or inconsistent behavior due to external interference.

To allow interaction with these private fields, the class provides public getter and setter methods, such as getAccountHolder() and setAccountHolder(String accountHolder). These methods act as a controlled interface through which other parts of the program can retrieve or update field values, while still respecting the rules and validations defined within the class.

For example, a setter method can include validation logic (e.g., rejecting invalid names), and a getter provides read-only access without exposing the field directly.

As illustrated in **Figure 15**, this design pattern allows the program to maintain a clean separation between internal data representation and external access behaviour, which is a best practice in designing secure and maintainable object-oriented systems.(*Immutability in JavaScript – Explained with Examples*, 2024)A screenshot of a computer

AI-generated content may be incorrect.

**Figure 14: Example of Access Modifiers**

**A computer screen with text

AI-generated content may be incorrect.**

**Figure 15: Getter/Setter Methods**

# Password Encryption

To ensure secure authentication, passwords must be stored in a protected manner. To achieve this, I will implement hashing along with salting. Hashing transforms a password into an irreversible cryptographic representation, while salting adds a unique, random value to each password before hashing. This process ensures that even identical passwords produce distinct hashes, making it significantly harder for attackers to use precomputed databases, such as rainbow tables, to crack stored credentials. Storing passwords as plain text is widely discouraged due to security vulnerabilities, making these cryptographic techniques essential for safeguarding user credentials. (*Encryption vs. Hashing vs. Salting - What’s the Difference?*, no date) 

**Figure 16: Password Hashing**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Figure 17: Verifying input against hashed password**

# Authentication & Validation

When a registered user attempts to login to the system, they will be sent a OTP to the phone number and email address linked to their account. (Prasad, 2020)

A computer screen shot of text

AI-generated content may be incorrect.

**Figure 18: OTP Input validation**

As shown in **Figure 18**, the login process requires the user to enter the correct one-time password (OTP) sent to their email and phone number. Since each OTP is uniquely generated, this enhances the system’s security by ensuring that access cannot be granted using a reused or predicted code. If the user enters an incorrect OTP in either input field, they will be denied access to the system. (*Password and OTP Generator With Java | Java Project*, 2023) A screenshot of a computer program

AI-generated content may be incorrect.

**Figure 19: OTPService**

As shown in **Figure 19**, the OTPService generates a random six-digit number to serve as a one-time password (OTP). These OTPs are stored in a HashMap, where the key is an identifier that the method can reference when sending the OTP to a user. The service then verifies the OTP entered by the user against the stored OTP associated with that identifier, determining whether to grant or deny access.

This implementation currently generates OTPs solely within the system and does not send them to the user's actual phone number or email. However, I did write additional code that, in theory, integrates Twilio and an email sender to facilitate OTP delivery. Due to challenges in implementation, I opted not to include this functionality in the system. (*Twilio SMS Tutorial For Beginners (FULL 2024 GUIDE)*, 2024)

## Conclusion

This project has provided valuable insights into the development of a secure software system, highlighting the importance of secure coding practices and structured software design. The Secure Banking Application successfully demonstrates key security features, including password validation, encrypted authentication, data persistence, and protection against SQL injection through the use of prepared statements. The incorporation of static analysis tools and exception handling further contributed to the application’s resilience and reliability.

From a security perspective, the system achieved a solid foundation in safeguarding user data. The implementation of password hashing and the introduction of a multi-factor authentication mechanism indicate a proactive approach to preventing unauthorized access. While the OTP verification system was limited to internal generation, the groundwork for future integration with external messaging services has been established.

The project also revealed several areas for future improvement. Implementing full OTP delivery via SMS or email, enhancing encryption for user data beyond passwords, and incorporating more comprehensive input validation are key recommendations for advancing the system’s security posture. Moreover, adopting a layered architectural approach or leveraging secure development frameworks such as Spring Boot could support better scalability and maintainability.

The most significant learning outcome of this project was developing a security-first mindset in software development. The process highlighted the complexity of building secure systems and the importance of anticipating vulnerabilities early in the development lifecycle. Overall, this project reinforced the critical role of secure programming in modern application development and provided a strong foundation for further exploration in cybersecurity and software engineering.

## References

*Check the Strength of a Password Using Java* (2023). Available at: https://www.youtube.com/watch?v=rwp1irWJtTc (Accessed: 9 April 2025).

*Immutability in JavaScript – Explained with Examples* (2024) *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/immutability-in-javascript-with-examples/ (Accessed: 15 April 2025).

‘Java Code Example: check password strength’ (no date). Available at: https://codevisionz.com/lessons/java-check-password-strength/ (Accessed: 4 September 2025).

*Java Exceptions (Try...Catch)* (no date). Available at: https://www.w3schools.com/java/java\_try\_catch.asp (Accessed: 10 April 2025).

Miecznik, R. (2023) *The Importance And Benefits of Unit Testing*, *CodiLime*. Available at: https://codilime.com/blog/unit-testing/ (Accessed: 25 March 2025).

*Password and OTP Generator With Java | Java Project* (2023). Available at: https://www.youtube.com/watch?v=odnJaqLfUhE (Accessed: 9 April 2025).

Prasad, A. (2020) ‘arjunpsd/Basic-Java-OTP’. Available at: https://github.com/arjunpsd/Basic-Java-OTP (Accessed: 9 April 2025).

*PreparedStatement | JDBC Tutorial for Beginners - YouTube* (2021). Available at: https://www.youtube.com/watch?v=T0P-cfsD45o&ab\_channel=JavaCodeJunkie (Accessed: 9 April 2025).

*Snyk Developer Security Platform | AI-powered AppSec Tool | Snyk* (no date). Available at: https://snyk.io/ (Accessed: 10 April 2025).

*SonarQube for Visual Studio 2022 - Visual Studio Marketplace* (no date). Available at: https://marketplace.visualstudio.com/items?itemName=SonarSource.SonarLintforVisualStudio2022 (Accessed: 25 March 2025).

*Twilio SMS Tutorial For Beginners (FULL 2024 GUIDE)* (2024). Available at: https://www.youtube.com/watch?v=QjLtzPBy8DY (Accessed: 9 April 2025).