15	jobskills.interviewgemini.com Internet Source	<1%
16	doczz.net Internet Source	<1%
17	www.bidnet.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Table of Contents

1.0 Risk Analysis & Threat Modeling..... 3
1.1 System Context..... 3
1.2 Trust Boundaries..... 3
1.3 Data Flow Diagram..... 4
1.4 STRIDE Threats Identification..... 6
1.5 Risk Assessment Matrix..... 8
1.6 Mitigation Controls..... 10
2.0 Secure Design & Architecture..... 12
2.1 Architectural Diagram of The Secure Component..... 12
2.2 Secure Design Principles..... 16
2.3 Cryptographic Primitives, Authentication Scheme, Session Management, and Access-Control Model..... 18
2.4 Risk-To-Control Mapping..... 21
3.0 Secure Coding Implementation..... 23
3.1 Git Commit History..... 23
3.2 OWASP ASVS Implementation..... 24
3.3 README Documentation..... 25
4.0 Security Testing and Validation..... 28
4.1 Test Plan..... 28
4.1.1 Test Objectives..... 28
4.1.2 Test Scope..... 28
4.1.3 Test Environment and Tools..... 29
4.1.4 Entry and Exit Criteria..... 29
4.2 Test Implementation..... 30
4.3 Defect Summary..... 38
5.0 Conclusion..... 39
References..... 40
Appendices..... 43
Appendix A: DFD diagram..... 43
Appendix B: Secure Architecture Diagram..... 43
Appendix C: ITS68504_ SSS_Assignment_Website..... 43
Appendix D: TestEvidencePack..... 43

1.0 Risk Analysis & Threat Modeling

The purpose of this section is to identify and analyse risk that lies within the component itself in order to plan out mitigation techniques for high risk threats.

1.1 System Context

SecureFin Sdn Bhd is a fintech startup company that is establishing an online customer portal to streamline customer's online financial decisions. Fintech is an abbreviation of "Finance Technology" where its focus lies on how to improve financial services using technology. Supported by (Siroto, 2024), states that a fintech utilises advancement of technology to ease financial management, speeds up the tedious processes and improve accessibility.

Since fintech is a type of "digital-only bank", it is highly targeted by cybercriminals to access sensitive financial data. Thus, fintech relies heavily on a robust security defense mechanism in order to build customer trust. Cybersecurity problems that are faced by fintech companies are evolving cybersecurity threats, data privacy and customer information protection, fraud detection and prevention, and etc. (Katariya, 2026). But the problem that we are addressing in this document will be balancing seamless UX with strong security.

In any user authentication and authorization component, they are responsible for user registration, user login, password management, role-based access control (RBAC) and authorization decision. In simpler words, this component functions to only allow authenticated users to access and perform actions within the portal. Throughout this documentation, we will be implementing secure SDLC principles to deliver a working prototype and supporting security artifacts.

1.2 Trust Boundaries

A trust boundary is one of the most important components in a security architecture. A trust boundary is a stage in architecture where access control, validation and authority must vary because each actor will be having different privileges and risks (Rajeshkumar, 2026). The trust boundary of the system will be divided into 2 and for each boundary will be given brief justifications as per relevance to the system.

a. External Network

The external network will be the customer's internet provider. This is because the internet is untrusted because anyone could be sending malicious requests, spoof identities, brute-force passwords, etc. so it is important for us to be having this boundary to protect the system from external threats mentioned.

b. SecureFin Internal Network

The internal network will be the whole portal system module. For user authentication and authorization module, it will consist of the authentication server, the database server, the log server, multi-factor authentication server and the system's database itself. As the servers work hand in hand, they are managed and protected by internal security controls such as API and Firewall Gateways.

1.3 Data Flow Diagram

Data Flow Diagram (DFD) is a graphical tool that is used to visualise the system's architecture on how the data moves, displaying each input and outputs of every process and storage points. The following diagrams will demonstrate how data flows through the system for the user authentication & authorization module.

Level 0

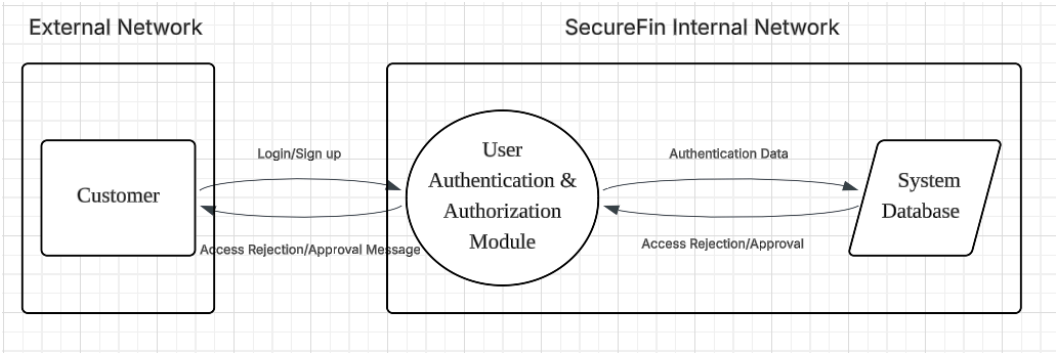


Figure 1: *DFD diagram Level 0*

Level 1

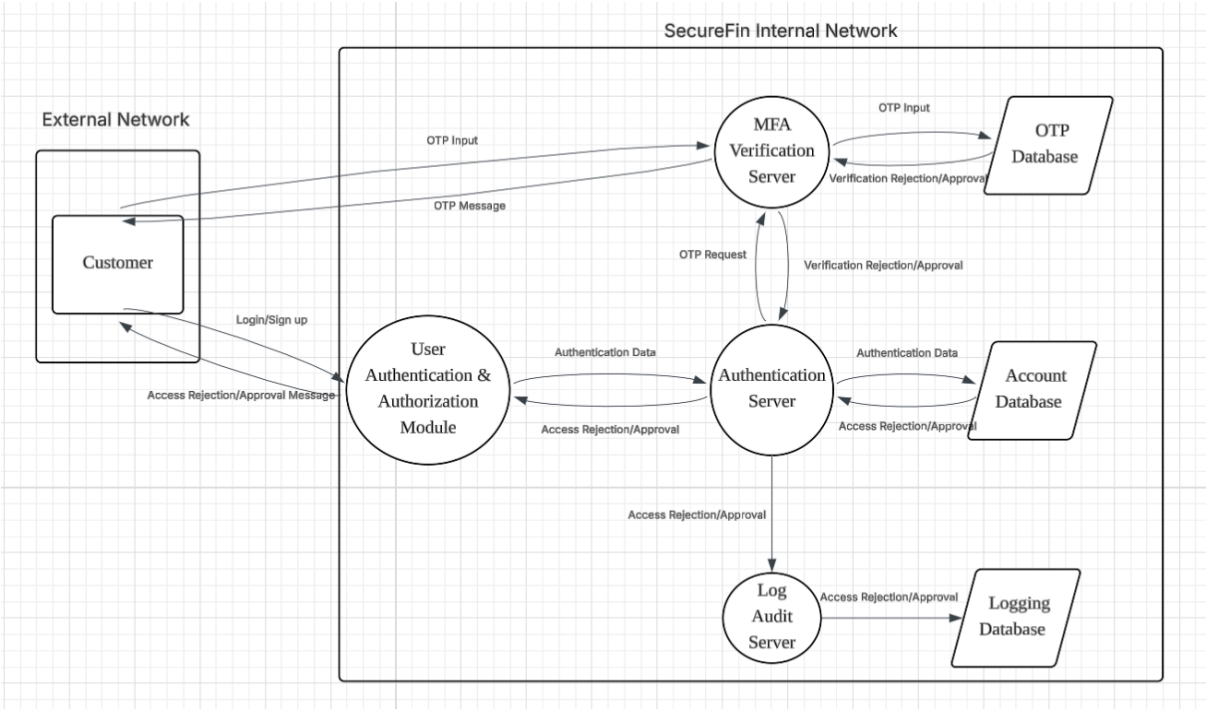


Figure 2: *DFD diagram Level 1*

1.4 STRIDE Threats Identification

In terms of threat modelling we used a STRIDE concept to discover and fix system vulnerabilities. It identifies 6 types of security threats and allows developers to assess the risk of each security risk in every development stage. Therefore in our user authentication & authorization module, here are the threats identified that will likely occur in the module together with its affected asset, threat actor, attack vector and the CIA impact.

Table 1: *Threat Modelling*

Threat	Category	Affected asset	Threat Actor	Attack Vector	CIA Impact
Credential Stuffing / Brute Force Login	Spoofing	User's Account	External Attacker	Automated login attempt using leaked username or password combinations gained from data breaches	Confidentiality
Unauthorized Access	Elevation of privilege	User's account	External Attacker	Exploiting broken authentication or authorization controls to gain access beyond assigned privilege	Confidentiality
Database Leakage	Information Disclosure	User Credentials	External Attacker	Either SQL injection or Database Breach	Confidentiality

Weak Input Validation	Tampering	Application database	External Attacker	Malicious input through Cross-Site Scripting (XSS) and SQL injection	Integrity
Duplicate Account Registration	Spoofing	User’s records	External attacker	Register multiple account using stolen identities, temporary email addresses and fake info	Integrity
Weak Password Selection	Spoofing	User’s Account	External Attacker or legitimate user	Users choosing predictable or commonly used passwords that are easily guessed and vulnerable to dictionary or brute force attack	Confidentiality
Insecure Password Storage	Information disclosure	System’s database and user’s credentials	External Attacker or malicious insider	Through data breach, exposing plaintext or weakly hashed passwords	Confidentiality

1.5 Risk Assessment Matrix

The risk assessment matrix is used to evaluate and prioritize the identified security threats based on their likelihood of occurrence and potential impact on the user authentication & authorization component. Each threat is assigned a risk score by multiplying its likelihood and impact ratings, allowing risks to be classified into different levels ranging from low to very high. This assessment will help to determine which threats require immediate mitigation and should be managed through continuous monitoring.

Table 2: *Risk classification*

Score	Risk Level	Tolerance Decision	Required Action
1-5	Very Low/Low	Acceptable	Monitor periodically and maintain existing controls
6-9	Moderate	Tolerable with control review	Review current controls, monitor regularly, and improve control if needed
10-16	High	Unacceptable without mitigation	Mitigation plan is required before the risk can be accepted
20-25	Very High	Unacceptable/ Immediate action required	Immediate treatment, management escalation and strong security controls are required

Based on the threat listed in Table 1.4.0, the risk score for each threat is as follows. Risk scoring is performed by evaluating each identified threat based on its likelihood of occurence and potential impact on CIA of the User Authentication & Authorization component. The risk score is calculated by multiplying the likelihood rating by the impact rating (Risk Score = Likelihood x Impact). The resulting score is then mapped to the corresponding risk level defined in table 1.5.1. This systematic approach enables the project team to prioritise security threats according to their severity, to ensure that high and very high risk receive immediate attention while lower-risk threats are monitored and managed appropriately.

Table 3: *Risk scoring*

Threat	Likelihood	Impact	Risk Score (Likelihood x Impact)
Credential Stuffing / Brute Force Login	5	4	20 - Very High
Unauthorized Access	3	4	12 - High
Database Leakage	4	5	20 - Very High
Weak Input Validation	4	4	16 - High
Duplicate Account Registration	4	4	16 - High
Weak Password Selection	4	4	16 - High
Insecure Password Storage	3	5	15 - High

1.6 Mitigation Controls

Based on the risk assessment, appropriate mitigation controls are proposed to reduce the likelihood and impact of the identified threats. Each control is selected according to the nature of the threat and follows secure software design principles such as defense in depth, least privilege, and fail-safe defaults. The proposed mitigation will include mechanisms such as multi-factor authentication, parametrized queries, database encryption, etc. This is to strengthen the authentication process and protect sensitive user data.

Table 4: *Mitigation controls*

ID	Threat	Mitigation	Justification
R01	Credential Stuffing / Brute Force Login	Enforce strong password policy, progressive login delay and multi-factor authentication (security questions).	Strong passwords reduce successful credential guessing while additional authentication prevents unauthorized access even if credentials are compromised.
R02	Unauthorized Access	Verify username, password and security question answers before granting access.	Multiple authentication factors ensure only legitimate users can successfully log in.
R03	Database Leakage	Hash passwords with salt (SHA-256), parameterized SQL queries and least-privilege database access.	Password hashing protects stored credentials while parameterized queries prevent SQL injection attacks.
R04	Weak Input Validation	Validate usernames, passwords and user inputs before processing.	Input validation prevents malicious or unexpected input from affecting application behaviour or database operations.
R05	Duplicate Account Registration	Check username uniqueness before account creation.	Prevents account conflicts and maintains database integrity.

R06	Weak Password Selection	Enforce password complexity requirements (uppercase, lowercase, number, special character and minimum length).	Strong passwords increase resistance against dictionary and brute-force attacks.
R07	Insecure Password Storage	Store only salted password hashes instead of plaintext passwords.	Even if the database is compromised, original passwords remain difficult to recover.

2.0 Secure Design & Architecture

This section outlines the secure design and architecture of the User Authentication & Authorization portion of the SecureFin customer portal. The design considers all these threats and risks identified in Q1: Credential Stuffing, Database leakage, Session Hijacking, Login Request Flooding, MFA Bypass, Account Lockout Abuse, Session Token manipulation, & Password reset token prediction. This section is intended to demonstrate how layered, security controls are employed in the proposed architecture in an attempt to minimize these risks but also provide a viable user authentication mechanism *(Datla & Thodupunuri, 2021)*.

A hybrid of prototype-level controls and proposed controls for deployment. The prototype incorporates key secure coding principles including input validation, input sanitization, output encoding, CANAPI's salted-sha256 password hashing, SecureRandom token creation, and using a time-out for 2FA *(Fedlalla & Elshoush, 2023; Sriramya & Karthika, 2015; Prasad et al., 2020)*. The deployment level controls include WAF, API Gateway, TLS 1.3 mTLS, AES-256 database encryption, HSM-based key management, SIEM monitoring and backup recovery. These controls are added to illustrate how to have the component securely deployed in a real fintech set-up *(Kumar et al., 2025)*.

2.1 Architectural Diagram of The Secure Component

Below is the secure layout of the User Authentication and Authorization component. It depicts the flow of end-user requests to the system from the untrusted outside network into the semi-trusted DMZ and ultimately to the trusted services and the highly trusted data layer. This visualization also includes the primary trust boundaries, security controls, and data-protection methods for authentication data, session information, user credentials, and audit logs *(Datla & Thodupunuri, 2021)*.

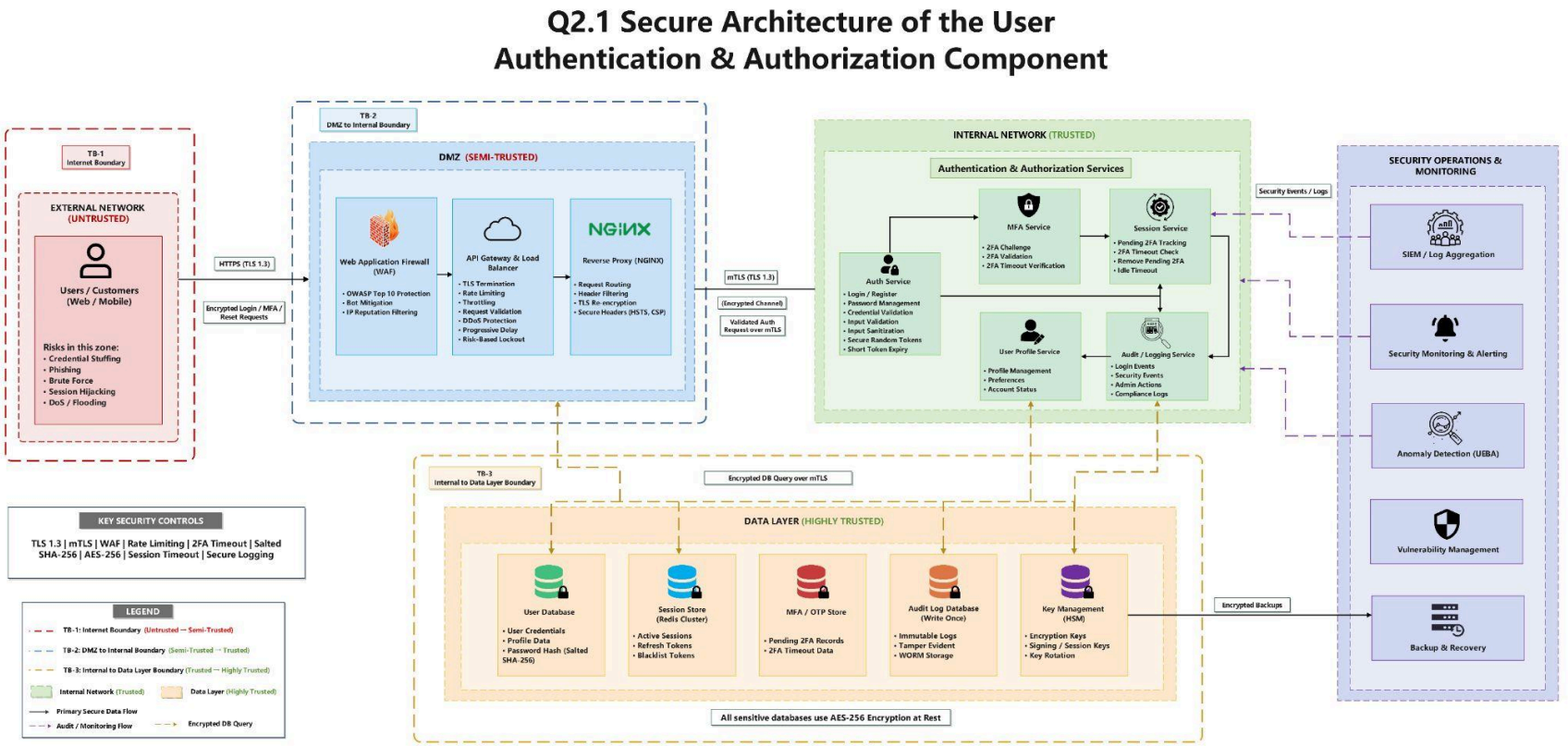


Figure 3:

Secure Architecture Diagram of User Authentication & Authorization Component

The main zones of the architecture are External Network, DMZ, Internal Network, Data Layer and Security Operations and Monitoring. External Network is the public internet portion of the network where users connects to the SecureFin portal from a web browser or mobile device. Since this zone is untrusted, users are not allowed to communicate directly with the internal authentication services. The DMZ helps to protect and control requests that reach outbound sensitive services by providing a buffer zone between the external network and the trusted internal network *(Iskandar et al., 2019).*

The Web Application Firewall, API Gateway and Load Balancer and Reverse Proxy are included in the DMZ. These controls check, cleanse and deliver requests to the inside services. The API Gateway enforces rate limiting, throttling, request validation, progressive delay and risk-based lockout to diminish bruteforce, credential stuffing and login flooding, while the WAF is able to block web attacks, bot traffic and malicious requests *(Iskandar et al., 2019)*. The Reverse Proxy provides secure routing, header filtering, secure headers and TLS re-encryption.

The internal network is where all the Authentication and Authorization Services can be found. Login, registration, password services, authentication, validation of inputs, sanitation of inputs, secure random token generation, and short-lived tokens are managed by the Auth Service *(Fadlalla & Elshoush, 2023)*. Validation and sanitization of inputs are necessary as they minimize any potential risks for invalid or harmful inputs to be accepted by the application *(Prasad et al., 2020).*

The MFA Service can be used to fulfil 2FA challenge, 2FA validation and 2FA timeout verification. This provides an additional layer of identity verification, and helps mitigate the threat of credential compromise *(Kumar et al., 2025)*. The Session Service maintains pending Two Factor Authentication sessions, enforces timeouts, removes stale pending Two Factor Authentication records and provides idle time out. Importance of Timeouts: These decreases the time window for session abuse *(Johnston, 2004).*

The Data Layer is trusted to a very large extent; it holds sensitive information about the system. It's made up of the User Database, Session Store, MFA/OTP Store, Audit Log

Database and Key Management component. User passwords are kept as a salted SHA-256 hash, and as a result, if the Username/Password database is leaked, it will minimise the amount of plaintext passwords that will be exposed *(Sriramya & Karthika, 2015)*. AES-256 encryption security at rest is also proposed for use for sensitive databases as a deployment-level mechanism. The MFA/OTP Store stores any pending 2FA's and timeout data and the Audit Log Database stores all immutable/tamper evident logs. The Key Management component is a representation of the secure handling of encryption and sign/session keys in a production environment.

There are three points on the diagram that are labeled 'trust boundary points'. TB-1 is used to divide the External Network (untrusted) and DMZ (semi-trusted). TB-2 is the security barrier between the DMZ and the secure Internal Network, and can facilitate only properly validated authentication requests, in encrypted form. To protect against database access by unauthorized internal or external services, SB-3 has divided the Internal Network from the highly trusted Data Layer. These boundaries enable network segmentation as well as mitigate the effects of a potential compromise *(Datla & Thodupunuri, 2021)*.

The overall design strategy of defense-in-depth is used, implementing multiple controls on different layers. WAF filtering, rate limiting, progressive delay and risk based lockout help mitigate external threats. To minimize credential- and database-related risks, a password is hashed, input is validated, input is sanitized, database access is encrypted, and access to the database layer is limited. Errors and risks associated with sessions and 2FA are mitigated with timeout handling, secure generation of random tokens and audit logging. For this reason, the architecture is intimately related to the risks highlighted in Q1 and is a structured security baseline for the prototype and the subsequent deployment.

2.2 Secure Design Principles

The proposed architecture / design addresses the risks of Q1 with regard to the User Authentication and Authorization component by using secure design. These principles are put into a holistic application throughout the architecture such as the DMZ, internal authentication services, data layer, and monitoring layer. Table 5 describes the application of each of the principles and the reduction of which risks on the Q1 are supported.

Table 5: *Application of Secure Design Principle in Proposed Architecture*

Secure Design Principle	Definition	Application in Architecture	Mitigated Q1 Risks
Defense in Depth	Using many, stacked layers of security controls in such a way that failure of one security control leaves several security controls to guard the asset.	Multiple layers of protection such as DMZ filtering, WAF, API Gateway controls, TLS/mTLS communication, 2FA timeout handling, salted hash of password based on her SHA-256 algorithm at runtime, AES (256-bit) encryption at rest and security logging.	Credential stuffing, database leaks, session hijacking, login request flooding and MFA bypass.
Least Privilege	Only providing users and services with the minimum amount of permissions to carry out the tasks necessary to them.	Only the component(s) that need it have access to the internal services and the Data Layer. No direct access to internal services or databases is allowed for external users and trusted internal services are the only ones with access to the databases.	Session token manipulation, database leakage.

Fail-Safe Defaults	Architecting systems such that they switch into some safe or secure state (e.g. access denied) in case of an error or exception.	By default, the system fails and rejects any requests that are not successfully validated, that hit rate-limits or that are not successfully two-factored in the time allowed. Sessions pending second factor are removed when they expire and are not allowed to continue if the attempt is not valid.	Account lockout abuse, multi-factor authentication bypass.
Complete Mediation	To make certain that all access request to all objects undergo authoritarian checks, disregarding any security measures.	The WAF is connected in series with a series of other services, including the API Gateway (APi Gateway) and the reverse proxy, to protect any internal services. Access to the database will be indirect and via trusted internal services, with encrypted communications.	Session hijacking, session token manipulation.
Separation of Duties	Sharing of duties among various parts or roles with the aim of not having one party being in control of everything.	The responsibilities of the services are distributed among the Auth Service, MFA Service, Session Service, User Profile Service, Audit/Logging Service, Data Layer and Key Management component. This helps to stop one component controlling the complete authentication process.	Database leakage, password reset token prediction.
Economy Of mechanism	As small and uncomplicated as possible security design to minimize the attack surface and afford mistakes.	There are clear and standard security measures employed including TLS 1.3, mTLS, WAF filtering, input validation, input sanitization, salted hashing of passwords with the SHA-256 algorithm, implementation and handling of 2FA, and secure logging. This helps the design to be understandable and easy to review.	Login request flooding, credential stuffing.

The proposed architecture adopts a secure design in multiple layers instead of using a single control (Table 5). The DMZ works with internal services, data layer and monitoring layer together to mitigate the risks discovered in Q1. This allows for a defense in depth

strategy and facilitates the linkage of authentication, session management, data security and logging features in a single artwork of security.

2.3 Cryptographic Primitives, Authentication Scheme, Session Management, and Access-Control Model

The Authentication and Authorization module applies both prototype-level security controls and proposed deployment-level controls to a user's credentials, authentication requests, data pertaining to sessions and sensitive system information. These mechanisms contribute to the secure architecture depicted in Figure 3 and play a part in mitigating risks as identified in Q1.

- **Cryptographic Primitives**

To secure external communication between users and the SecureFin portal, TLS 1.3 is proposed. mTLS is proposed as a deployment-level control to enable encrypted and authenticated communication between trusted services for internal service-to-service communication. AES-256 encryption-at-rest is used as a production level data-protection mechanism for sensitive databases. In the prototype, users passwords are stored using salted SHA-256 password hashing technique which generates salt before password as well. Additionally, unpredictable tokens are generated using SecureRandom to minimise the risk of an easily predicted reset/verification token. In a real-life environment, the encryption and signing/session keys could be managed using a KMS or HSM solution.

- **Authentication Scheme**

The authentication scheme consists of username and password, password complexity validation. The prototype also includes input validation, input sanitization, and output encoding, decreasing the risk of malicious or questionable input into the system. Users will also have 2FA, or two-factor authentication, in which they'll need to go through an additional verification method before they’re given access. The challenge validation and handling of timeout are part of the 2FA process, enabling sessions to be removed if they go pending for a period of time. Proposed gateway-level controls include rate limiting and progressive delay, as well as risk-based lockout to mitigate brute forcing, credential stuffing and flooding of login requests.

- **Session Management**

The session management design ensures authentication state is controlled and time span of misuse is minimized. The Session Service informs there are sessions remaining for 2FA and verifies the timeout, and clears out the pending 2FA records when they are expired, plus is able to support idle timeout. This is to guarantee that the partially or inactively completed authentication attempts are not held "open" for a long period of time. Short-lived, non-predictable tokens are also issued to limit impacts of token misuses.

- **Access-Control Model**

Implements restricted service access and least privilege access-control model. There is no access for the outer to the internal services or Data Layer directly. Every request will first go through the DMZ controls and then to the internal authentication services. Only trusted internal services are able to access the databases, and the sensitive data stores are isolated within the very trusted Data Layer. This model decreases the risk of unauthorized access, and a compromised component will have a minimal effect on the other components.

Table 6: *Security Mechanism Selected for The Authentication & Authorization module*

Category	Selected Mechanism	Justification
Cryptographic Primitives	TLS 1.3, mTLS, AES-256, salted SHA-256, SecureRandom, KMS/HSM	Secures data both in transit and at rest. In the prototype, the stored passwords are protected with salted SHA-256 and SecureRandom can be used to generate unpredictable tokens. KMS/HSM is included as a key management control proposed to be used at production level.
Authentication Scheme,	Username/Password, Password Complexity, 2FA Challenge, 2FA Timeout, Rate Limiting	Understands tougher user verification and dissuades credential-based attacks like credential stuffing or brute force attempts. 2FA timeout disables unfinished 2FA sessions.

Session Management,	pending 2FA Records, Timeout Check, Remove Expired 2 FA,Idle Timeout	Restricts the time spanning of session or verification misuse, and purges inactive or expired session and/or verification states.
Access-Control Model	Laset Pirvilege, Restricted Service Access, No Direct Database Access	Running on the perimeter, prevents direct access of any user or outside request on internal services or databases. Only services that are in the trusted domain have access to sensitive data.

The results, as displayed in Table 6, prove that the selected security mechanisms can provide the protection of the Authentication and Authorization module presented at various layers. Cryptographic controls encrypt data on the fly, as well as stored passwords and the process of generating tokens, whilst the authentication controls add additional security to the authentication process, handling password validation and the timeout for operating with 2FA. Limited validity of pending verification sessions with session management and restrictive direct access to internal services and databases using access-control model. These controls taken together contribute to supporting the secure architecture and mitigate the key risks identified in Q1.

2.4 Risk-To-Control Mapping

This subsection is intended to relate the threats identified in Q1 with specific design controls proposed in Q2. The intent of this mapping is to indicate that an appropriate architectural or technical control has been implemented for each risk. Input validation, salted SHA-256 password hashing, SecureRandom token creation and 2FA timeout are some of the controls that were added in the prototype. API Gateway, TLS/mTLS, AES-256, HSM, SIEM and other controls – proposed deployment level controls for a real fintech environment. Table 7 illustrates correlation between each of the Q1 threats, STRIDE category, controls selected, and how the control minimizes the threat by reducing threat likelihood and/or minimizing threat impact.

Table 7: Risk-To-Control Mapping for User Authentication and Authorization Component

Threat	Related Q1 STRIDE Category	Design Control from Q2	How the Control Mitigates The Risk
Credential Stuffing	Spoofing	WAF, API Gateway Rate Limiting, Progressive Delay, 2FA Challenge	WAF and rate limiting retard automated logon attempts. Doing it repeatedly makes it more likely to be done correctly, and Progressive Delay decreases the number of times a person has to attempt, and 2FA requires an additional verification step even if the person knows the password.
Session Token Manipulation	Tampering	Session Service, SecureRandom Tokens, Short Token Expiry, Session Timeout	Short expiry or timeout period restricts the period of potential misuse of a token, and SecureRandom lowers the predictability of tokens. Expired pending sessions are cleaned up by the Session Service as well.

Database Leakage	Information Disclosure	AES-256 Encryption at Rest, SHA 256 Password Hashing, Input sanitization, Restricted	Encryption at rest is a proposed deployment control to protect sensitive databases. The passwords are stored as salted hashes in the hash algorithm SHA 256, and data is sanitized on input to minimize access attempts with malicious data.
Session Hijacking	Information Disclosure	TLS 1.3, mTLS, Session Timeout, Idle Timeout	The TLS and mTLS prevents interception of the communication process. The session and idle timeout reduces the risk of the attack on the unutilized or compromised session.
Login Request Flooding	Denial of Service	Web Application Firewall (WAF), API Gateway (Rate Limiting & Throttling)	The WAF and API Gateway detect, and block exceptionally large amounts of traffic originating with a particular IP addresses or botnets before it can load the Authentication Server.
Account Lockout Abuse	Denial of Service	Progressive Delay, Rate Limiting	The system imposes gradual delays instead of locking an account immediately after failed attempts (the attackers use this to create DoS) so as to frustrate the attackers and not lock out the account until the genuine user recovers it.
Multi factor authentication bypass	Spoofing	2FA Challenge, 2FA Validation, 2FA Timeout Verification, Audit Logging	2FA validation ensures that the second method of authentication is performed. The timeout validation helps in stopping the incomplete process of 2FA while audit logging aids in identifying any suspicious attempt.
Password Reset Token Prediction	Elevation of Privilege	SecureRandom Token Generation, Short Token Expiry, Input Validation	Generation of reset tokens through SecureRandom makes it hard to predict the reset token. Expiry helps to minimize time for exploiting the token.

The list of found risks is mitigated by ‘layered security’ controls rather than a single control as depicted in Table 7. WAF filtering, API Gateway controls, rate limiting, plus progressive delay, mitigate external attack, like credential stuffing and login request flooding. Data-related risks are mitigated by salted password hashing, data input sanitization, limiting access to the data layer, and potentially encrypting data on disk. Timeout handling, secure generation of random tokens and audit logging mitigate the risks associated with session or 2FA processing. Thus, the controls in Q2 directly relate to the risks identified in Q1.

3.0 Secure Coding Implementation

Following the system architecture design proposed in the above section, we developed a working source code and practice security coding measures. We applied multiple secure-coding requirements following the OWASP ASVS Version 5.0.0 standard. The source code is then hosted in a Git repository to show meaningful git commits that were carried out while developing the code.

3.1 Git Commit History

A git commit history allows saving of the project iteratively and keeping track of the features and security measures being added into the code. This also gives a chronological record of the features being added.

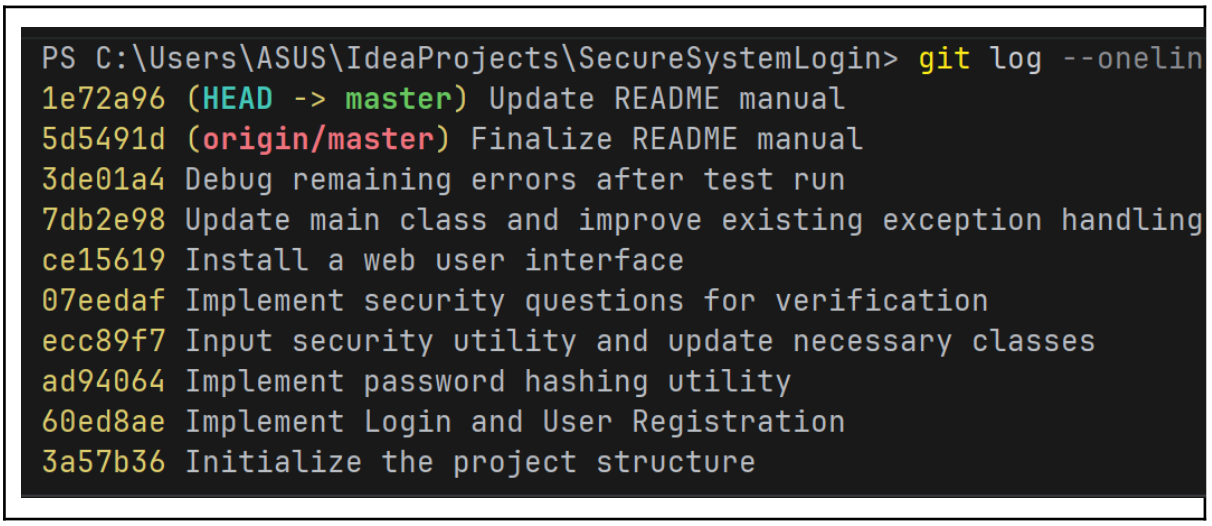


Figure 4: *Commit History in Git*

3.2 OWASP ASVS Implementation

OWASP ASVS allows implementing security measurements and controls to promote secure development for a web application. Applying this standard can boost confidence in the security as well as provide testing guidelines and controls to ensure a secure web application (OWASP, 2024).

Table 8: *Mapping of OWASP ASVS Implementation*

No.	OWASP ASVS ver.5.0.0	Control Name	Implementation in Code
1	**2.2.1**	Input Validation	Validate username input with minimum of 3 characters and maximum of 20 characters.
2	**6.2.1**	Password Security	Passwords must be a minimum of 8 characters including uppercase and lowercase alphabets, numbers and symbols.
3	**1.1.2**	Encoding and Sanitization Architecture	Encode possible malicious input codes to be translated into safe codes or entities.
4	**1.3.3**	Sanitization	Entirely removes parts of input that are malicious.
5	**7.3.1**	Session Timeout	Session is timed out if security questions (2FA) are not answered promptly.
6	**11.4.2**	Hashing and Hash-based functions	Passwords with salt are hashed using algorithms such as SHA-256.
Github repository URL: https://github.com/hnazahar/SecureSystemLogin			

3.3 README Documentation

A README manual provides the basic information of the developed project such as the functions, way of development of the system, list of security implementation and et cetera to be given or read by other users to allow them to test out the program.

```
### **Project Overview**

Our project is a Java-based secure system environment for user
registration and login. The aim of this developed system is to showcase
secure coding practices that should be implemented to prevent
unauthorized access such as input validation, password hashing, session
timeout, access control and exception handling. Our system is displayed
with a User Interface (UI) that runs on a local server when executed. Our
developed system uses in-memory database for simplicity of this project.

### **Features**

1) USER MANAGEMENT
- User registration
- User login
- User logout

2) SECURITY FEATURES
- Input validation
- Password hashing
- Password strength validation
- Input sanitization
- Session timeouts

3) SYSTEM FEATURES
- Exception handling
- Interactive user interface

### **Technologies Used**
- Java (JDK 21)
- HTML/CSS (Frontend/UI)
- IntelliJ IDEA (IDE)
- Git/GitHub

### **Project Structure**

### **Installation**
i) Requirements
- Java JDK 24 or above
- IntelliJ IDEA
- Git

ii) Steps
- Clone the repository --> using git clone <repository-url>
- Open the project in IntelliJ IDEA
- Configure Java SDK
- Run Main.java

### **Usage**
```

```
For Admin:
1. Run application on local server (prompted)
2. Select Log in tab and login as Admin
3. Enter admin's username and password (Username: admin, Password:
Admin@123)
4. Answer Security Questions for authentication (Smith, The Hobbit)
5. To update security questions, re-enter password to confirm change

For Tester (not Admin):
1. Run application on local server (prompted)
2. Select Log in tab and login as Testuser
3. Enter admin's username and password (Username: testuser, Password:
Test@123)
4. Answer Security Questions for authentication (New York, Buddy)
5. To update security questions, re-enter password to confirm change

For user registration:
1. Run application on local server (prompted)
2. Select register tab to register
3. Enter username and password and re-enter password for confirmation
4. Select Security Questions and provide answers for authentication
5. To update security questions, re-enter password to confirm change

### **Security Implementation**
Based on OWASP ASVS version 5.0.0
1)
2.2.1
Input Validation
- Validate username input with minimum of 3 characters and maximum of 20
characters
2)
6.2.1
Password Security
- Password must be minimum 8 characters including uppercase and lowercase
alphabets, numbers and symbols
3)
1.1.2
Encoding and Sanitization Architecture
- Encode possible malicious input codes to be translated into safe codes
or entities
4)
1.3.3
Sanitization
- Entirely removes parts of input that are malicious
5)
7.3.1
Session Timeout
- Session is timed out if security questions (2FA) is not answered
promptly
6)
11.4.2
Hashing and Hash-based functions
- Passwords with salt are hashed using algorithms such as SHA-256


### **Git Development History**
1. Initialize project structure
2. Implement Login and User Registration
3. Implement password hashing utility
```

```
4. Input security utility and update necessary classes
5. Install security questions for further verification
6. Install a website-based user interface
7. Debug remaining errors
8. Write down a README.md manual file

### **Future Improvements**
- Use of a proper database system to store information (not in-memory)
- Use of backend frameworks like Springboot
- Provide more flexibility such as to be able to change username and
password
- Add limits/timeouts for multiple retries to prevent brute force
guessing

### **Contributors**
Section 3, Group 3
```

Figure 5: *README file in repository*

4.0 Security Testing and Validation

This section outlines the security testing and validation performed on the SecureFin Authentication Portal. The testing will verify that the system behaves securely under both normal and adversarial conditions.

4.1 Test Plan

The test plan describes the testing approach for the SecureFin Authentication Portal with Two-Step Verification. The system is tested using Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), functional testing, and manual abuse-case testing to verify both functionality and security before deployment.

4.1.1 Test Objectives

The following objectives guide the testing activities and ensure the authentication module is validated from both functional and security perspectives:

- Verify core system functionality.
- Verify secure user authentication and authorization.
- Identify security vulnerabilities through SAST and DAST.
- Validate user input handling and session management.
- Ensure the application satisfies the defined security requirements.

4.1.2 Test Scope

To keep the testing focused, the scope was defined by identifying the features included in the testing process as well as those that fall outside the project requirements.

Table 9: *In Scope and Out of Scope Testing Items*

In Scope	Out of Scope
1. User Registration and Login 2. Password validation 3. Two-Step Verification 4. Session management 5. Input validation 6. Security testing (SAST & DAST)	1. Performance testing 2. Load testing 3. Database performance optimization 4. Mobile application testing 5. Third-party integration testing

4.1.3 Test Environment and Tools

The authentication module was tested using a combination of development tools, security testing tools, and a controlled testing environment to support both functional and security validation.

Table 10: *Testing Environment and Tools Used*

Category	Details
Operating System	macOS (Apple Silicon)
Browser Used	Google Chrome
Application Environment	Java-based HTTPS web application running on localhost
Database	SQLite
Network Environment	Localhost (127.0.0.1)
Security Configuration	HTTPS enabled with TLS certificate
SAST Tool	SonarQube Cloud
DAST Tool	OWASP ZAP

4.1.4 Entry and Exit Criteria

To ensure the testing process is carried out consistently, clear conditions were established for when testing can begin, when it can be considered complete, and the expected deliverables produced throughout the testing activities.

Table 11: *Testing Entry and Exit Criteria*

Criteria Type	Criteria Description
Entry	• Source code is completed. • Application builds successfully. • Testing tools are installed and configured. • GitHub repository is available for SAST analysis.
Exit	• All planned test cases have been executed. • SAST and DAST analyses are completed. • Identified issues are documented. • Test evidence has been collected.

The deliverables at the end of this testing phase are as follows:

1. Test Cases
2. SonarQube Cloud Report
3. OWASP ZAP Report
4. Test Execution Report
5. Defect Log
6. Test Evidence Pack (.zip)

4.2 Test Implementation

The following test cases were implemented to evaluate the security of the SecureFin Authentication Portal, with each test case traceable to the identified project risks.

Table 12: *Test Cases for SecureFin Authentication Portal*

TC ID	Test Type	Test Case	Risk Addressed
TC01	SAST	Static Source Code Security Analysis	[R03] Database Leakage [R04] Weak Input Validation [R07] Insecure Password Storage
TC02	DAST	Dynamic Web Application Security Assessment	[R02] Unauthorized Access
TC03	Manual (Positive)	Register New User with Valid Credentials	[R05] Duplicate Account Registration
TC04	Manual (Positive)	Authenticate User Using Two-Step Verification	[R01] Credential Stuffing
TC05	Manual (Negative)	Reject SQL Injection Payload During Login	[R03] Database Leakage [R04] Weak Input Validation
TC06	Manual (Negative)	Reject Weak Password During Registration	[R06] Weak Password Selection
TC07	Manual (Negative)	Deny Login with Incorrect Security Question Answer	[R02] Unauthorized Access

The execution details and supporting evidence for each test case are presented below:

Table 13: *Test Case 1 Execution Details and Evidence*

Test Case ID	**TC01**
Test Title	Static Source Code Security Analysis
Test Description	Verify that the application source code does not contain security vulnerabilities or poor coding practices using SonarQube Cloud.
Test Data	Latest version of project source code
Test Steps	1. Upload the latest source code to GitHub. 2. Run SonarQube Cloud analysis. 3. Review the generated security, reliability and maintainability findings.
Expected Result	No Critical or Blocker security issues are detected. Code complies with secure coding practices.
Actual Result	SonarQube Cloud completed the scan. No Critical or Blocker security issues were detected.
Test Status	**Pass**
Evidence	

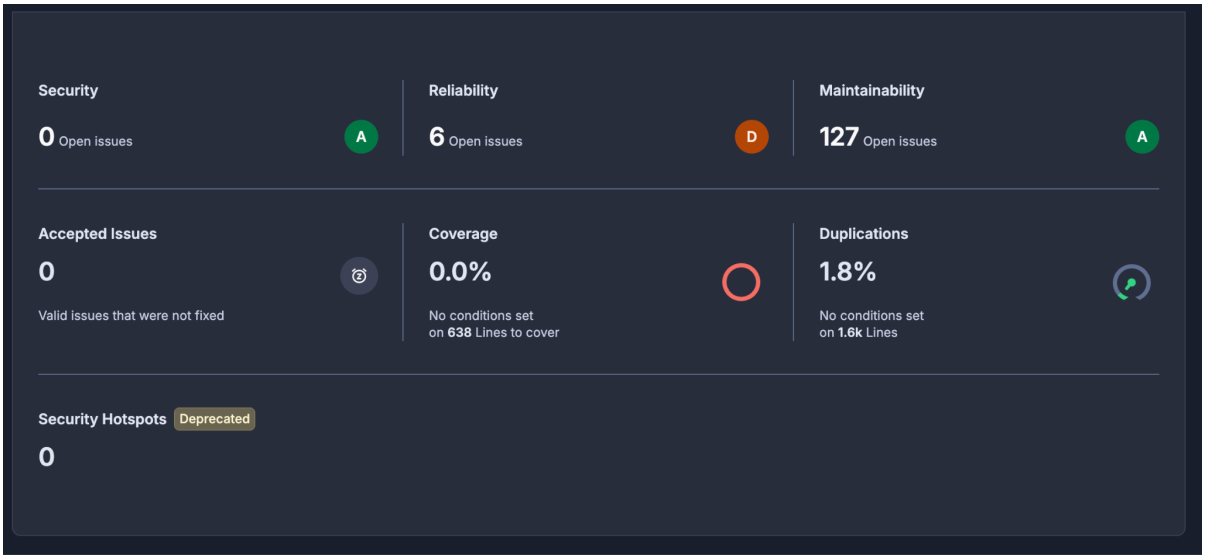


Figure 6: *SonarQube Cloud SAST Summary Report*

Table 14: *Test Case 2 Execution Details and Evidence*

Test Case ID	**TC02**
Test Title	Dynamic Web Application Security Assessment
Test Description	Verify that the deployed web application is resistant to common web vulnerabilities using OWASP ZAP.
Test Data	Running Secure Login System URL
Test Steps	1. Launch OWASP ZAP. 2. Configure the application URL. 3. Perform automated scan. 4. Review identified vulnerabilities.
Expected Result	No High or Critical vulnerabilities are reported.
Actual Result	OWASP ZAP completed the scan. No High or Critical vulnerabilities were detected.
Test Status	**Pass**
Evidence	

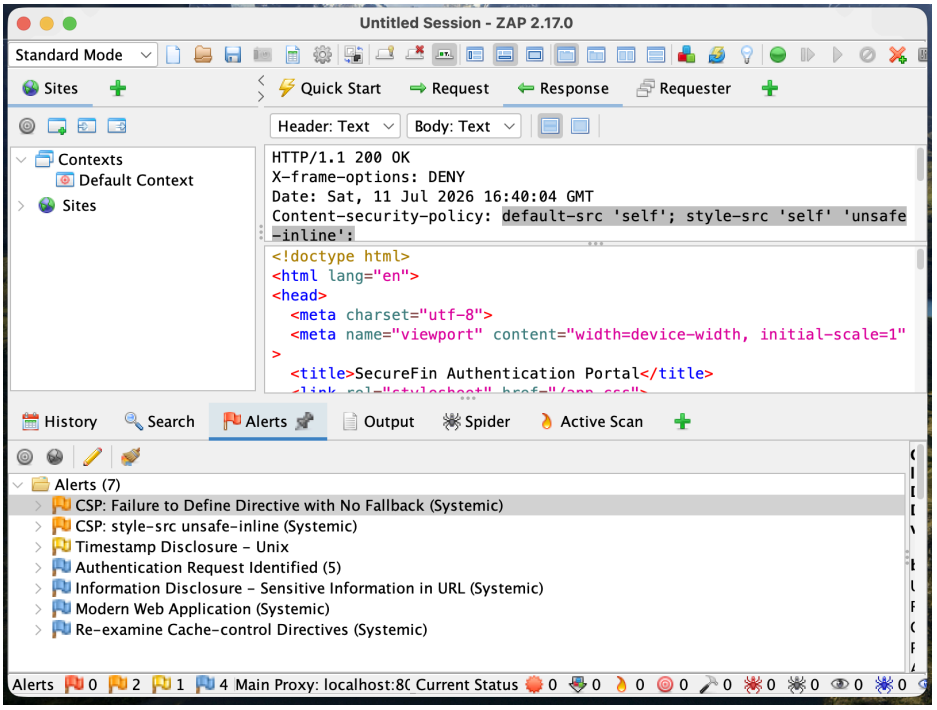


Figure 7: *OWASP ZAP DAST Summary Report*

Table 15: *Test Case 3 Execution Details and Evidence*

Test Case ID	**TC03**
Test Title	Register New User with Valid Credentials
Test Description	Verify that a new user can successfully register with valid information.
Test Data	Username: Sara | Password: Sara123! | Security Answer: test1, test2
Test Steps	1. Open Register page. 2. Enter valid user details. 3. Click the 'Create Account' button.
Expected Result	User account is created successfully and stored in the database.
Actual Result	User account was successfully created and stored in the database.
Test Status	**Pass**
Evidence	

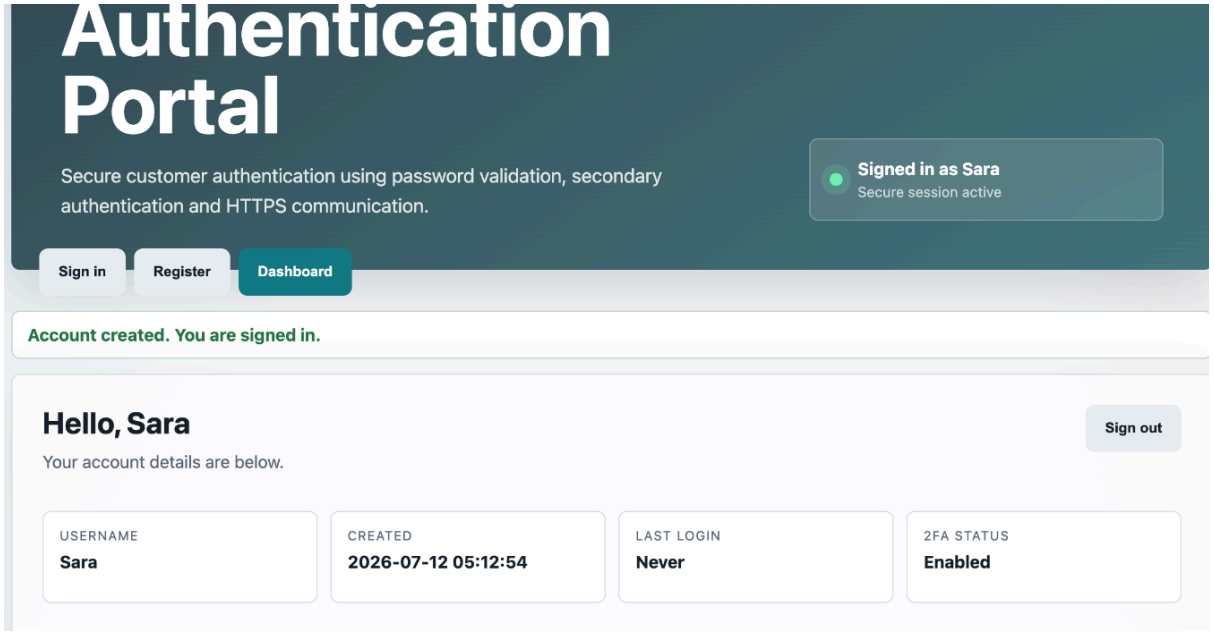


Figure 8: *Successful Account Registration*

Table 16: *Test Case 4 Execution Details and Evidence*

Test Case ID	**TC04**
Test Title	Authenticate User Using Two-Step Verification
Test Description	Verify that users can only log in after providing correct username, password and security question answer.
Test Data	Username: Sara | Password: Sara123! | Security Answer: test1, test2
Test Steps	1. Enter predefined username. 2. Enter predefined password. 3. Answer the security questions correctly. 4. Click 'Verify Identity' button.
Expected Result	User is successfully authenticated and granted access.
Actual Result	User successfully logged in after completing two-step verification.
Test Status	**Pass**
Evidence	

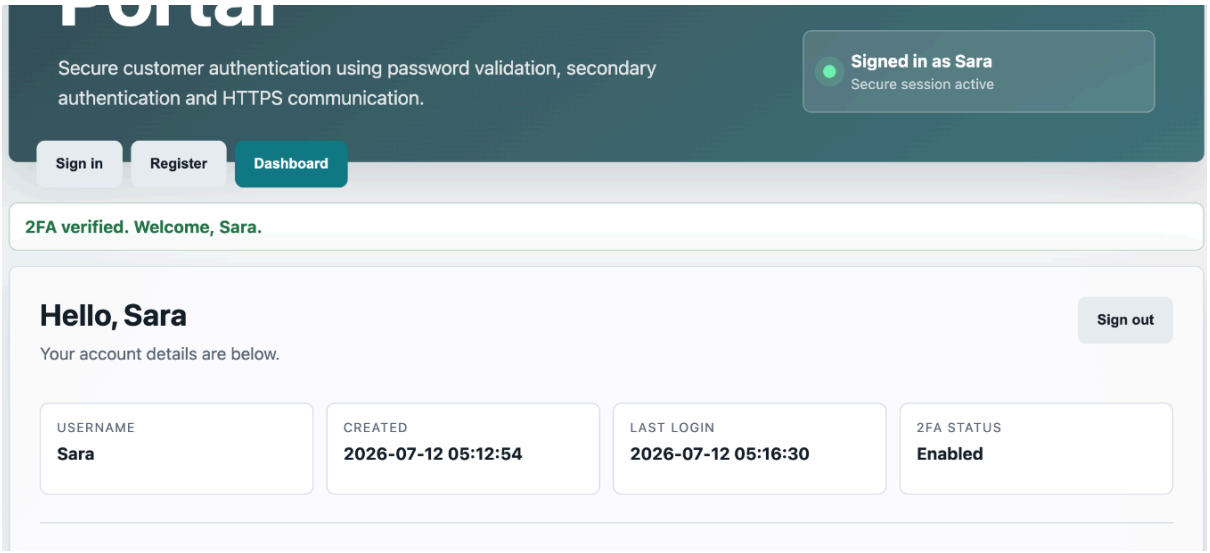


Figure 9: *Successful Login*

Table 17: *Test Case 5 Execution Details and Evidence*

Test Case ID	**TC05**
Test Title	Reject SQL Injection Payload During Login
Test Description	Verify that SQL injection attempts are rejected and do not bypass authentication.
Test Data	Username: ' OR '1'='1 | Password: ' OR '1'='1
Test Steps	1. Open Login page. 2. Enter SQL injection payload as username and password. 3. Click Login.
Expected Result	Login is rejected. No database error or unauthorized access occurs.
Actual Result	SQL injection attempt was rejected and login was denied.
Test Status	**Pass**
Evidence	

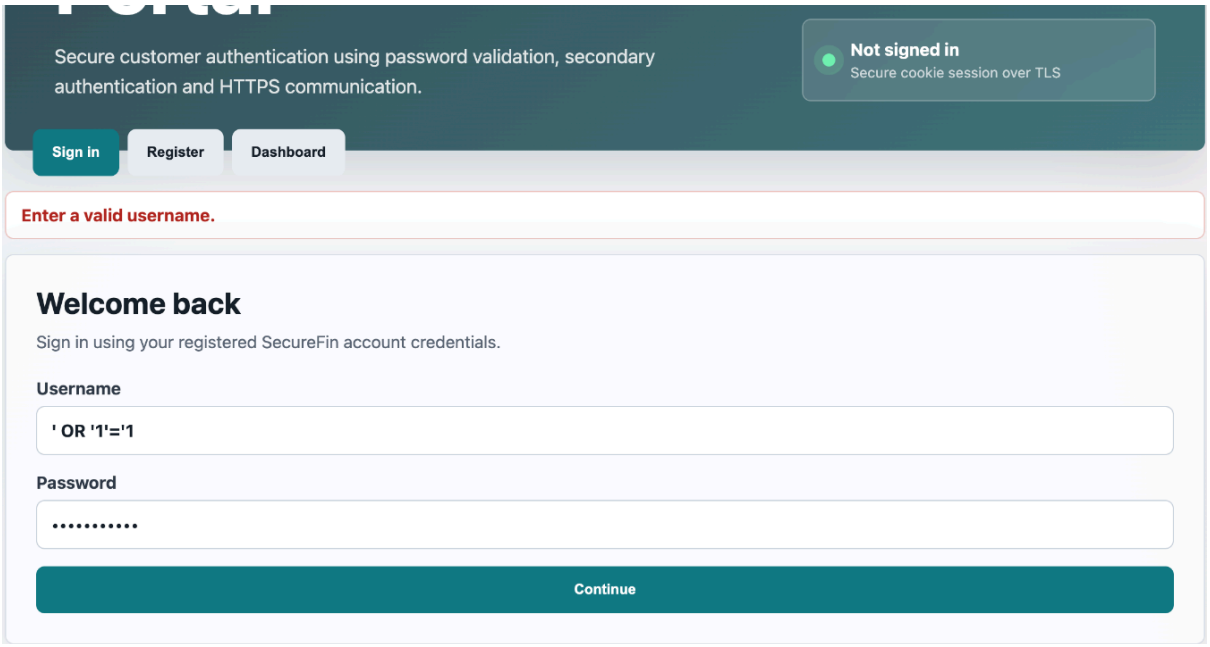


Figure 10: *Unsuccessful Login with SQL Injection Payload*

Table 18: *Test Case 6 Execution Details and Evidence*

Test Case ID	**TC06**
Test Title	Reject Weak Password During Registration
Test Description	Verify that passwords not meeting the password policy are rejected during registration.
Test Data	Username: test | Password: **abc123**
Test Steps	1. Open Register page. 2. Enter valid username. 3. Enter weak password as stated in test data. 4. Submit registration.
Expected Result	Registration is rejected and an appropriate password validation message is displayed.
Actual Result	Weak password was rejected and registration was prevented.
Test Status	**Pass**
Evidence	

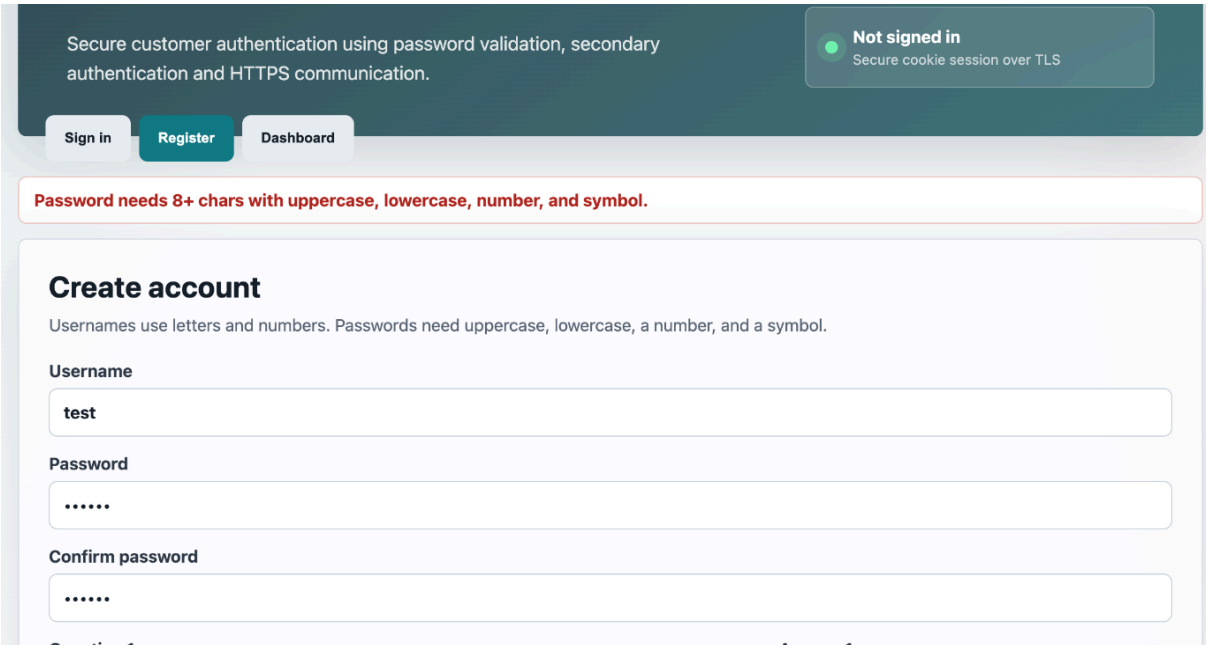


Figure 11: *Registration Denied*

Table 19: *Test Case 7 Execution Details and Evidence*

Test Case ID	**TC07**
Test Title	Deny Login with Incorrect Security Question Answer
Test Description	Verify that login is denied when the security question answer is incorrect even if the username and password are valid.
Test Data	Username: Sara | Password: Sara123! | Security Answer: no, no
Test Steps	1. Enter valid username. 2. Enter valid password. 3. Enter incorrect security answer. 4. Click Login.
Expected Result	Login is denied and an authentication failure message is displayed.
Actual Result	Login was denied due to incorrect security question answers.
Test Status	**Pass**
Evidence	

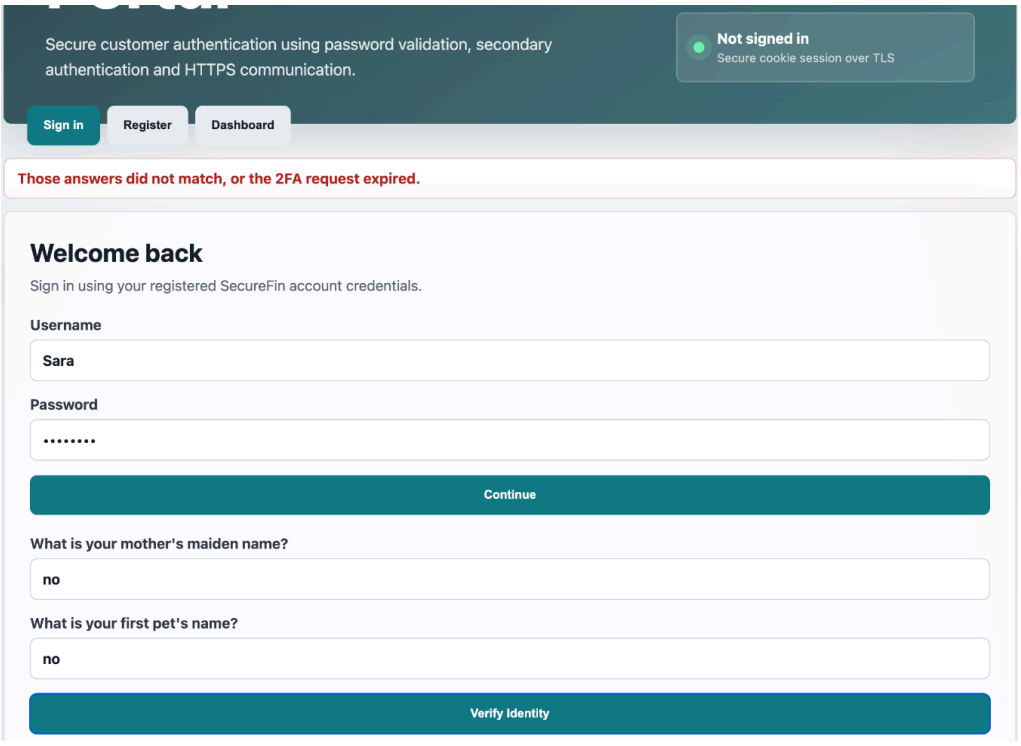


Figure 12: *Login Denied*

4.3 Defect Summary

The executed security tests identified several observations from the SAST and DAST assessments. No critical vulnerabilities were detected that prevented the system from meeting its security objectives. The findings mainly relate to code quality and security hardening recommendations, while the core authentication and two-step verification features functioned as expected.

Table 20: *Defect Summary Findings*

Defect ID	Source	Finding	Severity	Root Cause	Improvement Action	Status
D01	SonarQube Cloud (SAST)	Reliability issues detected (6 findings)	Medium	Minor code quality issues affecting reliability	Refactor affected methods and improve exception handling in future iterations.	Pass (Accepted)
D02	SonarQube Cloud (SAST)	Maintainability issues (127 code smells)	Low	Code duplication, long methods and style inconsistencies	Refactor duplicated code, improve naming conventions and modularize methods.	Pass (Accepted)
D03	OWASP ZAP (DAST)	Content Security Policy (CSP) configuration warnings	Medium	CSP directives are incomplete and allow unsafe-inline styles	Configure a stricter Content Security Policy by removing unsafe-inline and defining fallback directives.	Pass (Accepted)
D04	OWASP ZAP (DAST)	Timestamp disclosure in HTTP response headers	Low	Server exposes timestamp information	Disable unnecessary timestamp headers to reduce information disclosure.	Pass (Accepted)
D05	OWASP ZAP (DAST)	Cache-control configuration recommendation	Low	Cache-control headers can be strengthened	Apply stricter cache-control headers for authentication-related pages.	Pass (Accepted)

5.0 Conclusion

In this project, we demonstrated the design, development, and security validation process for the SecureFin User Authentication and Authorization component using secure software engineering approaches. The process began with threat modeling using STRIDE methodology and then all the threats were mitigated by adopting a multi-layered security approach and secure design and coding principles as per OWASP ASVS guidelines.

Multiple security controls have been incorporated into the architecture design, which include trust boundaries, network segmentation, multi-factor authentication, session management, encryption, role-based access control, and monitoring functions. Such controls have been chosen for their ability to mitigate the identified risks while ensuring that the process of authentication remains structured and scalable.

Implementation phase involved development of the prototype based on the suggested security controls, and validation phase was performed through conducting security testing using SAST, DAST and manual testing techniques for evaluating the effectiveness of the implemented solution. The findings from the testing process revealed that the authentication module satisfied the suggested security requirements and did not contain any critical security vulnerabilities; all other remarks concerned code quality only.

On the whole, this project shows how taking into account security issues at all stages of development—from design to implementation and testing—helps build a more secure authentication system. Despite the fact that the prototype is a simplified version of the implementation, the suggested architecture can serve as a basis for developing something more advanced in a fintech setting.

References

- Siroto, J. (2024, May 20). *Guide to the Differences Between Fintech vs Banks*. SoFi. https://www.sofi.com/learn/content/fintech-vs-traditional-banks/?msockid=09ae6cb3269c678d2f087ae6229c6544

- *Fintech Cybersecurity: Key Risks & Solutions [2025 Guide]*. (2025, April 29). Moon Technolabs. https://www.moontechnolabs.com/blog/fintech-cybersecurity/

- rajeshkumar. (2026, February 20). *What is Trust Boundary? Meaning, Architecture, Examples, Use Cases, and How to Measure It (2026 Guide) - DevSecOps School*. DevSecOps School. https://devsecopsschool.com/blog/trust-boundary/

- Datla, L. S., & Thodupunuri, R. K. (2021). Designing for Defense: How We Embedded Security Principles into Cloud-Native Web Application Architectures. International Journal of Emerging Research in Engineering and Technology, 2(4), 30-38. Designing for Defense: How We Embedded Security Principles into Cloud-Native Web Application Architectures | International Journal of Emerging Research in Engineering and Technology

- Iskandar, A., Virma, E., & Ahmar, A. S. (2019). Implementing DMZ in improving network security of web testing in STMIK AKBA. arXiv preprint arXiv:1901.04081. The Format of the IJOPCM, first submission

- Fadlalla, F. F., & Elshoush, H. T. (2023). Input validation vulnerabilities in web applications: Systematic review, classification, and analysis of the

current state-of-the-art. IEEE Access, 11, 40128-40161. IEEE Xplore Full-Text PDF:

- Sriramya, P., & Karthika, R. A. (2015). Providing password security by salted password hashing using bcrypt algorithm. ARPN journal of engineering and applied sciences, 10(13), 5551-5556. jeas_0715_2288-libre.pdf

- Prasad, B. V. V. S., Mandapati, S., Haritha, B., & Begum, M. J. (2020, August). Enhanced Security for the authentication of Digital Signature from the key generated by the CSTRNG method. In 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 1088-1093). IEEE. Enhanced Security for the authentication of Digital Signature from the key generated by the CSTRNG method

- Kumar, K. J., Sai, K. A., Reddy, A. D., Reddy, C. P. D., & Rajagopal, S. M. (2025, February). A Comprehensive End-to-End Solution for Web Security with Cryptography, Multi-Factor Authentication, and Secure Communication. In 2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT) (pp. 325-332). IEEE. A-Comprehensive-End-to-End-Solution-for-Web-Security-with-Cryptography-Multi-Factor-Authentication-and-Secure-Communication.pdf

- Johnston, P. (2004). Authentication and session management on the web. Retrieved December, 13(2004), 2009. 1_Paul_Johnston_GSEC.pdf

- OWASP. (2024). OWASP Application Security Verification Standard. Owasp.Org. https://owasp.org/www-project-application-security-verification-standard/

Appendices

Appendix A: DFD diagram

Appendix B: Secure Architecture Diagram

Appendix C: ITS68504_SSS_Assignment_Website

Appendix D: TestEvidencePack