



National Cyber  
Security Authority

# Application Software Security Directives

---



April 2026

Document Title: Application Software Security Directives

Document History:

Publication Date	Version No.	Description
April 2026	1.0	First release

## **CONTENTS**

<b>I. INTRODUCTION .....</b>	<b>3</b>
1.1. Purpose .....	3
1.2. Scope.....	3
1.3. Terms and definitions .....	4
<b>II. RECOMMENDED BASELINES TO SOFTWARE DEVELOPMENT.....</b>	<b>5</b>
<b>III. SECURE APPLICATION SOFTWARE DEVELOPMENT .....</b>	<b>6</b>
3.1. Software Requirements Gathering and Analysis Phase .....	6
3.2. Software Design Phase .....	7
3.3. Software Development Phase .....	7
3.4. Software Testing Phase.....	13
<b>IV. SECURE SOFTWARE DEPLOYMENT AND MAINTENANCE.....</b>	<b>15</b>
4.1. Environment configuration and deployment .....	15
4.2. File management .....	16
4.3. Database security .....	16
4.4. Security monitoring .....	17
4.5. Maintenance .....	18
<b>V. APPLICATION PROGRAMMING INTERFACE (API) SECURITY .....</b>	<b>19</b>
<b>VI. REFERENCES .....</b>	<b>22</b>

## I. INTRODUCTION

The use of application software is a critical aspect of today's businesses and everyday lives. We rely on digital applications and exchange of massive volume of data, some of it is sensitive, to accomplish a wide variety of tasks. This scenario requires us to take a firm stance at protecting and securing the data and underlying IT systems.

In this regard, the National Cyber Security Authority (NCSA) avails directives for secure application software development, determining the security requirements to be built into those platforms by design, in order to assure the maximum level of protection.

To ensure effective security outcomes, each phase in the application software development and use in the production environment, and corresponding initial tender preparation and procurement processes, where applicable, should refer to these directives.

### **1.1. Purpose**

NCSA published minimum cybersecurity standards that included secure application coding principles<sup>1</sup>; therefore, the procedures in these directives aim to detail the implementation of the standards when it comes to securely developing and maintaining application software.

These directives define the most essential security controls that must be implemented on any application software in order to ensure the confidentiality, integrity, and availability of those platforms and their data.

### **1.2. Scope**

These directives apply to application software of any type that is accessible from the internet or intranet; and targets all IT engineers, IT managers, software developers, software testers, IT security personnel, software owners and anyone who has in his/her responsibility the application software for public or private institutions that operate in Rwanda.

They are aligned with global secure coding development frameworks such as and ISO 27002:2022, control 8.28<sup>[4]</sup> and OWASP Top 10:2021<sup>[5]</sup>.

These directives are not a replacement for and do not supersede the national and sectorial regulations that software owners and developers must comply with as part of their regulatory obligations.

---

<sup>1</sup> Appendix 2 – Secure Application Coding Principles, Minimum Cybersecurity Standards <sup>[1]</sup>

### 1.3. Terms and definitions

The terms and definitions provided as used in this document:

TERMS	DEFINITION
Application software, Application, Web application, Website, Information Portal, Platform	A computer program that performs a specific function directly for an end user or, in some cases, for another application.  Each application is designed to assist users with a particular task related to productivity, communication or creativity.
Software Developer, Software Engineer, Programmer	A professional responsible for designing and building computer programs, by writing code, testing, and integrating software components to create functional application software.
Software tester, Quality Assurance engineer.	An individual with skills, or group of professionals, that tests software for bugs, errors, defects, or any problem that can affect the performance of application software.
IT Security Engineer, Information security engineer, IT Security personnel, Cybersecurity engineer.	An individual or group of professionals responsible for designing, implementing, and maintaining an organization's security infrastructure.  They are responsible for identifying potential risks and vulnerabilities and implementing mitigating solutions.
Web application firewall (WAF)	A security tool that protects applications against common web-based threats by monitoring, filtering, and blocking malicious traffic to a web application from the internet.
Secure coding	Practice of writing source code or code base in a manner that avoids the unintentional introduction of security vulnerabilities.
Software development lifecycle (SDLC)	Framework that helps define tasks for designing, creating, and maintaining software. It provides a well-structured flow of phases to follow to quickly produce high quality software.
Software Bill of Materials (SBOM)	Inventory of all constituent components and software dependencies involved in the development and delivery of application software.  It identifies and lists all libraries, modules, plugins, and other dependencies incorporated into the software's codebase. SBOM helps to understand, and manage software.

## II. RECOMMENDED BASELINES TO SOFTWARE DEVELOPMENT

These best practices define a common approach to software development and help ensure that a software is developed in a consistent, reliable, and efficient manner.

Here are some recommended baselines to software development every developer should apply:

1. choose the right software development methodology;
2. reduce code complexity, codebase must be developed with low complexity to make it efficient. Code complexity refers to the degree to which code is difficult to understand, modify, or maintain.
3. document software code properly;
4. set up continuous code integration, every time a task is finished, to ensure the code base is up-to-date;
5. carry out code reviews to improve code quality;
6. do thorough unit tests; unit testing involves assessing individual code components to verify their functionality;
7. maintain naming conventions of the variables throughout the code;
8. use a specific method for all comments;
9. review all third-party code, applications, and libraries to ensure safe functionality;
10. use hashes or checksums to verify the integrity of libraries, interpreted codes, configuration files, and executables;
11. prepare and submit a Software Bill of Materials (SBOM) after coding to enable the maintenance of the application.

### III. SECURE APPLICATION SOFTWARE DEVELOPMENT

This chapter defines, in the first four phases of application software development, the security procedures to apply for secure development.

This means that, while building the software, it is crucial that the developer adheres to secure coding standards and best practices by design.

The ideal time to define the security requirements starts early in the project initiation phase, and those security measures should be integrated in all phases of application software development as per security by design principles.

#### 3.1. Software Requirements Gathering and Analysis Phase

At this phase:

- a. the software developer gathers together all security specifications required by the various parties involved in the project.
- b. the software requirements provide the software developer with an overview of the application's core purpose.

This information is crucial for the development team to identify and define key security controls necessary for the application's protection at an early stage.

- c. the software development team should follow the relevant security policy or guidelines that outlines the security measures to implement.

To enable this, it may include steps such as implementing security controls, continuously monitoring activity in the system, and responding to potential threats in a timely and effective manner.

#### **Practices**

The software developer should:

- i. perform a threat and risk assessment to identify threats and vulnerabilities.  
*This may include identifying and analyzing potential vulnerabilities, potential malicious actors, attack vectors that the malicious actors can exploit and, assess the potential damage an incident could cause.*
- ii. identify a secure software development lifecycle, relevant cybersecurity regulations, standards and guidelines that addresses security in all stages of development and conform to them.
- iii. ensure that they have documented and justified the trust boundaries, components, and data flow within the application software.

### 3.2. Software Design Phase

The design phase transforms the identified requirements into a workable application software design.

In the design phase:

- a. the overall architecture of the application software is designed by taking into account the functional and security requirements;
- b. the software developer determines which security control mechanisms are required to ensure the confidentiality, integrity, availability, and accountability of the data.

#### **Practices**

The software developer should:

- i. implement and use secure design patterns and frameworks;
- ii. determine and review security features and controls that should be implemented to remediate meaningful risks;
- iii. incorporate security best practices into the functional and design specifications;
- iv. define and review the security architecture. The overall application software architecture should be designed with security in mind.
- v. appropriately identify and address all threats related to the design before moving on to the development phase;
- vi. implement threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing;
- vii. set up and configure software development tools to ensure the security of all the code base created.

*Such tools can be Integrated Development Environments (IDEs) or collaborative software development platforms used.*

### 3.3. Software Development Phase

The development phase focuses on transforming the design into an operable application software.

In the development phase:

- a. development must take place in conformance with secure coding standards. The coding standards shall include collections of rules that determine the methods for each programming language used;

- b. the developer should have up-to-date knowledge of the relevant security standards and guidelines and how they apply to the current project.

### Practices

The software developer should:

- i. Write code that meets security requirements.  
This means the source code must apply the security measures outlined in security policy and guidelines stated in previous phases, *such as applying recommended baselines in software development, and implementing proper authentication and encryption.*
- ii. Test code to ensure it meets the security requirements.  
This involves running tests to check for potential vulnerabilities and ensuring that the code is properly structured and secure against attack, *such as doing code reviews to identify any potential bugs or issues.*
- iii. Use only secure development tools (libraries, frameworks);
- iv. incorporate testing early (shift-left strategy) in the development phase and continue testing frequently throughout, to catch bugs early, and validate the code often;
- v. use algorithms with strength equal to or greater than AES-256, when using symmetric encryption;
- vi. Use algorithms with strength equal to or greater than RSA-4096, when using asymmetric encryption.

### 3.3.1. Secure Coding Practices

#### A. Input Validation

The software developer should perform input validation to ensure that only properly formatted data that meets expected criteria, are entered in the application system.

### Practices

The software developer should:

- i. apply input validation at both syntactical and semantic level;

Examples:

- syntactic validation should enforce correct syntax of structured fields (*e.g.: name, date, email, phone number*);

- semantic validation should enforce correctness of their values in the specific business context  
(*E.g.: the start date is before the end date, the price is within the expected range*);
- Proper handling of hazardous characters such as < > " ' % ( ) \* = & + - \ ' .
- ii. specify accurate character sets, such as UTF-8;
- iii. set minimum and maximum value range check for numerical parameters and dates, and minimum and maximum length check for strings;
- iv. conduct all data validation on the authenticated server or a trusted system;
- v. validate data from redirects to prevent malicious content from being directly submitted to the redirect target;
- vi. Ensure that all validation failures result in input rejection.

## B. Authentication and password management<sup>2</sup>

The software developer and software tester should verify and ensure that the following controls are considered to ensure that an effective authentication strategy is in place.

### Practices

The software developer and software tester should:

- i. ensure that the authentication controls are enforced on a trusted server;
- ii. use secure channels to transmit authentication requests;
- iii. require authentication for all pages and resources, except for those specifically intended to be public;
- iv. Verify that system-generated initial passwords or activation codes are securely randomly generated, contain letters and numbers, and expire after a short period;  
These initial secrets must not be permitted to become the long term password.
- v. enforce password complexity requirements established by policy or regulation  
(*e.g.: requiring the use of alphabetic as well as numeric and/or special characters, setting the minimum length of characters to be at least 10 characters, the password entry to be obscured on the user's screen*);
- vi. ensure that the password change functionality requires the user's current and new passwords, and prevents the reuse of previous passwords;
- vii. authentication failure responses should not indicate which part of the authentication data was incorrect

<sup>2</sup> Section 9: Identity Management and Authentication, Minimum Cybersecurity Standards for ESPs.

*(e.g.: instead of "Invalid username" or "Invalid password", recommend using "Invalid username and/or password" for both.)*

Error responses must be truly identical in both the message displayed and source code);

- viii. verify that all password fields do not echo the user's password when it is entered, and that password fields (or the forms that contain them) have autocomplete disabled;
- ix. verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, and logging in from unknown devices;
- x. ensure that passwords are not stored in plaintext and that strong password-hashing functions are applied  
*(e.g.: Bcrypt, Scrypt, Argon2);*
- xi. Implement multi-factor authentication and leverage modern authentication methods to improve security and user experience  
*(e.g.: time-based one-time password, authenticator app, biometrics and passkeys).*

### C. Session and Cookie Management

The software developer ensures that the application satisfies the following high-level session management requirements:

- a. sessions should be unique to each individual and cannot be guessed or shared;
- b. sessions should be invalidated when no longer required and timed out during set periods of inactivity.

#### Practices

The software developer should ensure:

- i. the application never reveals session tokens in URL parameters or error messages;
- ii. a session ID is meaningless and must not include sensitive information or personally identifiable information (PII);
- iii. concurrent sessions and session bypassing are disallowed;
- iv. A user is completely logged out after clicking on the logout button. Therefore, the user should not be able to go back to the previous page.
- v. And verify that the application terminates all other active sessions after a successful password change and that this is effective across the application and any relying parties;

- vi. And verify that cookie-based session tokens have the 'Secure' attribute set;
- vii. If multiple cookies are used in the application, all cookies are verified before allowing access to the application sessions.  
Multiple cookies utilization: used to identify a client across different pages or sessions, *for instance in storing session management and user preferences data.*

#### **D. Error Handling and Logging**

The primary objective of error handling and logging is to provide useful information to the user, system administrators, and incident response teams.

##### **Practices**

The software developer and IT security personnel should ensure:

- i. Generic error messages are implemented with custom error pages used in the application software. The application error text must not leak any sensitive information;
- ii. security controls are configured to deny access by default when encountering errors or unexpected conditions in the application;
- iii. The errors that need to be logged are defined and documented. This shall include authentication and session management data. All necessary information that allows detailed investigation of the timeline when an event happens, must be defined and logged properly;
- iv. Logs related to the application, including but not limited to access logs, transaction logs, and security logs, shall be stored in a read-only medium. Replicating the logs is important and should be followed according to the organization's data retention policy;
- v. Only designated individuals are allowed to access the log files. Access to the log files should be monitored, recorded, and reviewed regularly;
- vi. And verify that security logs are protected from unauthorized access and modification;
- vii. a separate log server is maintained where applicable;
- viii. Sensitive information is not stored in logs, including unnecessary system details, user session identifiers, or passwords.

## E. Data protection

The software developer should build the application software in compliance with the law relating to the protection of personal data and privacy in Rwanda.<sup>3</sup>

### Practices

The software developer and IT security personnel should:

- i. remove unnecessary application documentation from application software directories as this can reveal useful information to a malicious actor;
- ii. remove comments in user accessible production code that may reveal the backend system or other sensitive information;
- iii. ensure no sensitive information is included in HTTP GET request parameters;
- iv. disable auto-complete features on forms expected to contain sensitive information, including authentication;
- v. Ensure no passwords, connection strings, or other sensitive information are stored in clear text or any non-cryptographically secure manner on the client side.  
This includes embedding in insecure formats like MS ViewState, Adobe Flash, or compiled code;
- vi. limit the use and storage of sensitive data;
- vii. implement the least privilege principle, and restrict users to only the functionality, data, and system information that is required to perform their tasks;
- viii. protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files as soon as they are no longer required;
- ix. perform regular backups of important data and test the restoration of data;
- x. verify that backups are stored securely to prevent data from being stolen or corrupted.

---

<sup>3</sup> Law N° 058/2021 of 13/10/2021 relating to the protection of personal data and privacy

### 3.3.2. Static Analysis

The static analysis shall be performed by the developers on the application software code and server-side code after completing the code development process.

The process helps identify potential poor coding practices, programming flaws and security vulnerabilities to the code. The code review could be performed, either manually by going through each line of code or by using automated tools, but without executing the code.

Recommended actions are as follows:

**A. Planning:**

The objectives of the planning stages are to identify the code that needs to be reviewed, a team to perform the code review, the schedule to review the code and process, follow-up activities involved need to be clearly identified.

**B. Preparation:**

The code and other related material needs to be studied by the code reviewers. The role and the corresponding responsibilities of the reviewing team shall be assigned. Also, the recent error types and reviewing techniques shall be adopted.

**C. Inspection:**

As the name suggests, the errors in the code shall be identified at this stage. The code's author should make it easier for others to inspect the errors in the code. True errors shall be identified and noted with their severity levels. All findings shall be documented.

**D. Rework:**

All errors shall be remediated, and responses are to be provided. The author shall fix the code and revert with a response.

**E. Follow-up:**

Once rework is done, the project moderator follows up with the author to check if the changes have been made as mentioned. Unresolved observations shall be documented.

### 3.4. Software Testing Phase

The testing phase validates your application software before deployment.

The software tester should:

- a. test various deployment scenarios to ensure that the application will function properly in all environments;
- b. perform application security acceptance testing to check for the security weaknesses of an application software and ensure their remediation.

## Practices

The software tester and IT security personnel should:

- i. perform functional testing and non-functional testing  
*(e.g.: tests features of the application software and then evaluate its overall performance, usability, and reliability);*
- ii. perform penetration testing and vulnerability assessment by using different methods and technologies  
*(e.g.: test the application software against known security testing frameworks, such as OWASP Application Security Verification Standard, with automated tools or manually);*
- iii. describe identified vulnerabilities and specify remediation measures to be implemented before deployment;
- iv. conduct user acceptance testing to verify that the application performs the required tasks in accordance to given specifications;
- v. test the application software on an identical staging environment to the production environment prior to commissioning;
- vi. Keep record of the most valuable testing documentation.

The software developer should fix identified security weaknesses in the application and conduct regression testing before deployment. Regression testing is for ensuring that the application software works even after deploying a change.

## IV. SECURE SOFTWARE DEPLOYMENT AND MAINTENANCE

Once your application software is tested, it is time to deploy it. Software deployment typically includes activities such as provisioning environments, configuration, and setting up security monitoring.

### 4.1. Environment configuration and deployment

#### Practices

The software engineer and IT security personnel should:

- i. define and follow a clear and approved deployment process;
- ii. use the latest stable and secure version of application dependencies (e.g.: operating system, web server, CMS, plugins);
- iii. ensure that all security protocols are in place before deployment;
- iv. verify that the server(s) configuration is hardened enough to address security adequately, and when exceptions occur, fail securely;
- v. remove or disable unnecessary services and applications installed on the host operating system and the web server;
- vi. remove test code, files, or any functionality not intended for production;
- vii. verify that the web or application server debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures;
- viii. remove unnecessary information from HTTP response headers related to the Operating System, web server version, and application frameworks;
- ix. define and configure which HTTP methods, Get or Post, the application will support and whether it will be handled differently on different pages in the application;
- x. implement currently supported authentication and encryption technologies (e.g.: *Transport Layer Security (TLS)*), and verify that the message headers and payload are trustworthy and not modified in transit;
- xi. turn off directory listings;
- xii. prevent disclosure of your directory structure in the '*robots.txt*' file by placing directories, not intended for public indexing, into an isolated parent directory;
- xiii. double check that the system buffer is enough to handle all system functionalities;
- xiv. ensure multi-factor authentication is implemented where applicable;
- xv. implement endpoint detection and response (EDR) and data loss prevention (DLP);
- xvi. Disable the connection of testing and development environments to the internet, when it is no longer required.
- xvii. Isolate the development environment from the production network, and provide access only to authorized development and testing groups.

Development environments are often configured less securely than production environments and malicious actors may use this environment setup to discover shared weaknesses or as an avenue for exploitation.

## 4.2. File management

The application software should satisfy the following file management requirements:

### Practices

The software developer and IT security personnel should:

- i. define acceptable file types and sizes, and verify that the application will not accept unacceptable file types or sizes to prevent threats such as denial of service attacks;
- ii. Validate that uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient;
- iii. prevent or restrict the uploading of any file that may be interpreted by the web server (*e.g.: file named "image.php.jpg" that contains PHP code*), and prevent this by sanitizing the file names of files uploaded by users;
- iv. ensure application files and resources are read-only;
- v. separate file and application servers where applicable;
- vi. implement sandboxing to scan uploaded files.

## 4.3. Database security

The following actions should be implemented for securing a database system attached to the application software.

### Practices

The software developer and a system administrator should ensure:

- i. Connection strings are not hard coded within the application. Connection strings should be stored in a separate configuration file on a trusted system and they should be encrypted  
*(e.g.: If the credential in connection string is hard-coded, it is the same for each installation and cannot be changed or disabled by the system administrator without manually modifying the program. Anyone with this knowledge can discover and access the system);*
- ii. All default database administrative passwords are removed or changed. Strong passwords or passphrases are used, and multi-factor authentication is implemented as applicable;

- iii. restrict users, applications and APIs to access database; every access should be identified and authenticated;
- iv. remove unnecessary services and protocols  
(*e.g.: remove or disable unnecessary open ports*);
- v. the database server should not be assigned a publicly accessible IP address and access to the database should be allowed only from the web server on a particular port;
- vi. database and application servers are separated to reduce damage in case one of them is compromised;
- vii. database activity is audited and continuously monitored;
- viii. Data at rest is encrypted.

#### 4.4. Security monitoring

##### **Practices**

The IT security personnel and software engineer should:

- i. put deployed application software behind a web application firewall (WAF);
- ii. verify that logging is enabled and ensure all critical system access is logged and monitored;
  - Log monitoring can be performed using a syslog or a Security Information and Event Management (SIEM) and can be depending on set criteria of events.
- iii. monitor network traffic to and from the web server;
- iv. Provide official channels for users to report security identified vulnerabilities or bugs in the application software.

## 4.5. Maintenance

### Practices

The IT security personnel and software engineer should:

- i. Provide ongoing support, security assessment, and maintenance to ensure the application remains secure. *This includes monitoring event logs, and responding to any security incidents quickly and effectively in case they happen;*
- ii. create and implement a patch and change management process to address potential security vulnerabilities and ensure that the application software and its components are consistently updated according to the approved change management policy;
- iii. verify that third-party components come from pre-defined, trusted, and continually maintained repositories;
- iv. always take backups for the application software and keep them secure.

If the application software has been developed by a third party, the established contract should contain rules for maintenance and support (e.g.: timely removal or fixing reported errors) in the Service Level Agreement (SLA), containing indicators regarding cooperation procedures, timeliness for support, as well as sanctions for delay and failure to support.

Important and timely information on software vulnerabilities and mitigation strategies can be obtained from incident response teams such as the national Computer Security Incident Response Team (Rw-CSIRT). (<https://cyber.gov.rw/updates/alerts/>)

## V. APPLICATION PROGRAMMING INTERFACE (API) SECURITY

This chapter outlines API security measures that should be implemented to reduce risks associated with the use of APIs.

API cybersecurity is crucial for protecting sensitive data and ensuring the integrity and availability of the application.

Here are recommended baselines for software developers to enhance API security.

### Practices

The software engineer should implement the following:

#### A. Authentication:

- i. Implement strong authentication mechanisms to verify the identity of clients accessing your API.
- ii. Use methods like OAuth 2.0, JWT (JSON Web Tokens), API keys, or client certificates for authentication.

#### B. Authorization:

- i. Enforce proper authorization controls to limit access to API resources based on the authenticated user's permissions.
- ii. Implement role-based access control (RBAC) or attribute-based access control (ABAC) to ensure users only have access to the resources they are authorized to use.

#### C. Input Validation:

- i. Validate and sanitize all input received from clients to prevent injection attacks such as SQL injection, NoSQL injection, or Cross-Site Scripting (XSS).
- ii. Use parameterized queries, input validation libraries, and output encoding to sanitize user input effectively.

#### D. Output Encoding:

- i. Encode output data before sending it to clients to prevent Cross-Site Scripting (XSS) attacks.
- ii. Encode special characters such as <, >, &, -, \*, ", and ' to their respective HTML entities to prevent them from being interpreted as HTML or JavaScript code.

#### E. Rate Limiting:

- i. Implement rate limiting to prevent abuse, denial-of-service (DoS), and brute-force attacks on your API endpoints.
- ii. Define rate limits based on the number of requests per second or minute allowed from a single client or IP address.

- iii. Monitor API usage and set appropriate thresholds for rate limits based on expected traffic patterns (e.g.: 1,000 requests per hour).

**F. API Security Testing:**

- i. Conduct thorough security testing of your API, using tools like OWASP ZAP, Burp Suite, Postman, or API security scanners, to identify and remediate security vulnerabilities.
- ii. Perform regular security assessments, penetration testing, and code reviews to ensure the ongoing security of your API.

**G. Data Encryption:**

- i. Encrypt sensitive data stored in databases or transmitted over the network using strong encryption algorithms.
- ii. Implement encryption mechanisms such as TLS for data in transit and encryption for data at rest to protect against unauthorized access.

**H. Error Handling:**

- i. Implement proper error handling to provide informative error messages to clients without revealing sensitive information about the API's internal workings.
- ii. Avoid returning detailed error messages that could be used by attackers to exploit vulnerabilities.

**I. API Gateway:**

- i. Use an API gateway for centralized management, security, and monitoring of your APIs.

API gateways provide features such as authentication, authorization, rate limiting, logging, and analytics to enhance the security and scalability of your APIs.

**J. Security Headers:**

- i. Use security headers such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), X-Content-Type-Options, and X-Frame-Options to protect against common web security threats like cross-site scripting (XSS), clickjacking, and MIME sniffing attacks.

**K. Cross-Origin Resource Sharing (CORS):**

- i. Implement proper CORS policies to restrict access to APIs based on origin, HTTP methods, and headers; use whitelisting to specify trusted origins that are allowed to access the API.

**L. API Versioning:**

- i. Implement versioning to ensure backward compatibility as your API evolves. Versioning can be done through URL paths (e.g.: /v1/resource) or HTTP headers.

**M. Documentation:**

- i. Create comprehensive and up-to-date documentation that explains how to use your API, including endpoint descriptions, request/response examples, authentication methods, and error codes.

**N. Testing and Monitoring:**

- i. Thoroughly test your API endpoints using automated and manual tests to ensure reliability and consistency.
- ii. Implement logging and monitoring to track API usage, performance metrics, and error rates in real time.

**O. Third-party security:**

- i. Before using third party APIs, verify and assess their security practices;
- ii. Regularly monitor and review their security posture.

**P. Performance Optimization:**

- i. Optimize your API for performance by minimizing latency, reducing unnecessary data transfer, and caching frequently requested resources where applicable.

Furthermore, API security testing must be carried out periodically, at least annually.

## VI. REFERENCES

- [1] Minimum Cybersecurity Standards for Essential Service Providers, 2023. National Cyber Security Authority (NCSA).
- [2] Law N° 058/2021 of 13/10/2021 relating to the protection of personal data and privacy in Rwanda.
- [3] Digital Adoption Implementation Guidelines for the Government of Rwanda, 2025. Rwanda Information Society Authority
- [4] ISO/IEC 27002:2022, Information Security, cybersecurity and privacy protection – Information Security Controls. Control 8.28 – Secure Coding.
- [5] OWASP Top 10:2021.
- [6] OWASP Application Security Verification Standard 4.0.3., 2021.
- [7] OWASP API Security Top 10 2023.
- [8] Secure Coding Guidelines for Java SE version 10.0, 2023. ORACLE