

# Student Management and Service system SDLC

01.07.2024

Bahirdar Institute of technology

Bahirdar, Ethiopia

—Group one

Mintesnot Afework.....	1405579
Bishaw Tesema.....	1405305
Mikias Sisay.....	1404155
Tigist Kebede.....	1404855
Gelila Azanaw.....	1410564


# Secure SDLC for Student Management and Location Tracking System

This document outlines the development of a secure student management and location tracking system using the Waterfall model. The system prioritizes the security and privacy of student data throughout the development lifecycle, adhering to industry best practices for web application security.

## 1. System Requirements Definition (SRD)

### Functional Requirements:

- **User Roles and Functionalities:**
  - Define clear distinctions between user roles and their corresponding functionalities.
    - **Parents:**
      - Secure login and registration processes with strong password hashing.
      - Ability to view real-time location data for their registered children after a secure verification process.
      - Optional access to basic student information based on system configuration and parental consent (e.g., name, ID).
    - **Students:**
      - Secure login and registration processes with unique student identifiers and chosen passwords.
      - Ability to view their current status within the system, including basic information.
    - **Administrators (Single Account for enhanced security):**
      - Comprehensive management capabilities, including:
        - User account management (CRUD operations: create, read, update, delete) for parents and students.
        - Student information management (CRUD operations) encompassing personal details, and potentially location tracking data management (depending on implementation).
        - Campus details management (buildings, departments, cafes) including location data for buildings and cafes relevant for location tracking.
        - User profile management for creating, updating, and managing user profiles for parents and students,



enforcing access controls through RBAC (Role-Based Access Control).

### Non-Functional Requirements:

- **Performance:** Define expected response times and user concurrency thresholds to ensure smooth system operation under anticipated usage load. Consider conducting performance testing tools like JMeter or LoadRunner to identify bottlenecks and optimize system resources for scalability.

### Security Requirements:

- **Authentication and Authorization:**
  - Implement robust authentication mechanisms with strong password hashing algorithms (e.g., bcrypt) to protect user credentials. Explore the feasibility of multi-factor authentication (MFA) for high-risk access (e.g., administrator logins) if internet connectivity allows. Consider integrating with a third-party identity provider (IdP) for centralized user management and enhanced security.
  - Enforce RBAC to restrict access to specific data based on user roles. Parents cannot access location data for other students' children, and students cannot modify administrator settings. Implement granular access controls within roles for even finer-tuned permissions.
- **Data Security:**
  - Encrypt data at rest (stored data) using industry-standard encryption algorithms (e.g., AES-256). Encrypt data in transit (communication) using Transport Layer Security (TLS/SSL) to protect data transmission between users and the system. Consider encrypting database connections for additional protection.
  - Classify data based on sensitivity (e.g., location data - high risk, student ID - medium risk) and apply appropriate security measures. Implement data minimization principles – collect only the essential data required for system functionality. Explore data pseudonymization techniques, where data is linked to a pseudonym instead of a direct identifier, to further protect student privacy while maintaining data usability for authorized users.
  - Adhere to relevant data privacy regulations (e.g., GDPR, FERPA) by implementing appropriate data security controls and obtaining informed user consent for data collection and usage. Consider conducting a Privacy Impact Assessment (PIA) to identify and mitigate potential privacy risks throughout the development process.

## 2. System Design Document (SDD)

- **System Architecture:**

Design a secure system architecture with separation of concerns for enhanced security:

- Secure UI (consider responsive design for optimal viewing across devices). Utilize a well-established front-end framework like Bootstrap or Tailwind CSS to streamline development and ensure a user-friendly interface. Implement security best practices on the front-end, such as input validation and sanitization to prevent common web application vulnerabilities
- Secure backend logic built with a robust PHP framework like Laravel, leveraging its built-in security features (e.g., prepared statements, CSRF protection) and established security practices.
- Secure database management system (DBMS) like PostgreSQL, configured with strong access controls and user authentication for database access.
- Consider implementing a web application firewall (WAF) to provide an additional layer of protection against common web attacks.
- Secure server infrastructure – deploy the application on a secure web server like Nginx, configured with security best practices (e.g., regular security updates, firewalls, intrusion detection systems).
- Leverage Docker containers to package the application code, dependencies, and runtime environment into isolated units. This promotes:
  - **Consistency and Portability:** Ensures consistent application behavior across development, testing, and production environments.
  - **Isolation:** Isolates applications from each other and the underlying host system, enhancing security and reducing conflicts.
  - **Scalability:** Enables easy scaling of application instances by provisioning additional containers as needed.

- **Security Design:**

- Detail the chosen security mechanisms for authentication, authorization, and data security, aligning with the requirements defined in the SRD. Include specifics on:
  - Authentication protocol (e.g., username/password with hashing)
  - Authorization mechanisms (RBAC implementation details, access control lists)
  - Data encryption strategies (algorithms used for data at rest and in transit)

- Secure coding practices and development guidelines to prevent vulnerabilities (e.g., input validation, secure coding libraries)

## 3. Development

- **Secure Coding Practices:**

### **Laravel's Arsenal at Our Disposal:**

**Our** strategic use of Laravel's built-in security features demonstrates a deep understanding of potential vulnerabilities. By leveraging prepared statements, **we've** effectively eliminated the risk of SQL injection attacks, where malicious code could manipulate **our** database. Furthermore, by employing Laravel's CSRF protection, **we've** thwarted attempts to perform unauthorized actions through forms. These measures effectively safeguard **our** application's core functionality and user data.

### **Layered Defense with Input Validation and Secure Password Hashing:**

**Our** implementation of mass assignment protection and comprehensive input validation rules showcases a layered defense approach. This ensures that user input is meticulously controlled and sanitized before processing, preventing attackers from injecting malicious code or manipulating sensitive data. Additionally, **our** decision to utilize secure password hashing with bcrypt guarantees that user credentials are stored securely, making them resistant to brute-force attacks and data breaches.

### **JavaScript and Nginx: Partners in Security:**

The focus on JavaScript security demonstrates a well-rounded approach. By validating and sanitizing user input with libraries like DOMPurify, **we've** proactively prevented XSS attacks, a common tactic where malicious scripts are injected into user input. Moreover, **our** cautious manipulation of the DOM and the use of safer alternatives like `textContent` and `appendChild` further strengthens

**our** application's security posture. Finally, maintaining the latest version of Nginx and implementing a secure configuration with HSTS, CSP, and user permission restrictions reflects **our** commitment to server security. These measures collectively create a robust defense system against various attack vectors.

### Looking Ahead: Continuous Vigilance for Enduring Security

While **we've** implemented a comprehensive set of security practices, it's important to remember that security is an ongoing process. Here are some additional considerations to maintain a best-in-class security posture:

- **Security Headers:** Explore implementing security headers like X-Frame-Options and X-XSS-Protection to provide an extra layer of defense against specific attack techniques.
- **Code Review and Static Analysis:** Regular code reviews and static code analysis tools can help **us** identify and fix potential vulnerabilities early in the development cycle, fostering a secure codebase.
- **Staying Informed:** The security landscape is constantly evolving. By staying up-to-date on the latest threats and best practices, **we** can ensure that **our** Laravel application remains well-protected over time.
- **Unit Testing:**
  - We Write unit tests for individual components of the system, focusing on core functionalities and security considerations.
  - We have Developed test cases to ensure data is handled securely (e.g., validation of user input, proper encryption practices).

## 4. System Integration Testing (SIT)

### I. Module Integration:

- A. Integrate all developed modules and components into a cohesive system.
- B. Perform comprehensive integration testing to ensure modules interact seamlessly and functionalities work as intended across different user roles.



- C. Prioritize security testing during integration, focusing on areas where different modules interact and data is exchanged.

## 5. System Acceptance Testing (SAT)

## 6. Deployment and Operations

### Secure Deployment

### Secure Maintenance

- **Implementation, System Acceptance and Operations & Maintenance:** are not considered as they are not part of the project.

## SDLC Phases and Security Considerations:

- **Initiation:**
  - **Security Roles:**
    - **Developers:** Follow secure coding practices, write unit tests with security considerations, and participate in code reviews.
    - **Security Analysts:** Conduct code reviews, penetration testing, and vulnerability scans.
    - **System Administrators:** Configure security controls like firewalls, intrusion detection systems (IDS), and manage user accounts.
  - **Data Classification:** Student data is classified based on sensitivity using a data classification scheme. Location information is considered high-risk and requires additional security measures. Other data points like field, dorm, courses and parent information for students are considered to be medium risk; most of them are public by default.
    - **High-Risk:** Real-time location data, password, profile-picture, parent information.
    - **Medium-Risk:** Student ID, name, department, field, dorm, parent name
    - **Low-Risk:** Public information like campus map, cafe hours.
  - **Risk Management Plan:** A risk management plan identifies potential threats specific to the application and outlines mitigation strategies. This considers the likelihood and impact of each threat and prioritizes risks based on severity. Examples of threats and mitigations:

### Threat Category: Unauthorized Access

- **Threat:** Unauthorized access to student location data.
- **Mitigation:**
  - **Strong Authentication:** Enforced for administrator logins using complex passwords and potentially multi-factor authentication (MFA) if internet connectivity allows.
  - **RBAC (Role-Based Access Control):** Granular access controls restrict location data access based on user roles. Only authorized personnel (e.g., designated administrators) can view this sensitive data.
  - **Data Encryption:** Location data is encrypted at rest (stored data) using industry-standard algorithms (e.g., AES-256) and in transit (communication) using Transport Layer Security (TLS/SSL) for additional protection.
- **Threat:** Unauthorized access to the admin page.
- **Mitigation:**
  - **MFA (Consideration):** Implement multi-factor authentication (MFA) for administrator logins if internet connectivity permits. This adds an extra layer of security beyond usernames and passwords. (Note: Adjust based on actual internet availability)
  - **RBAC (Enforced):** RBAC strictly restricts access to the admin page and functionalities based on user roles. Only administrators can access and manage sensitive information.
  - **Data Encryption (Maintained):** As mentioned above, location data and other sensitive information remain encrypted at rest and in transit.

### Threat Category: Authentication Bypass

- **Threat:** Authentication bypass attempts (consider brute-force attacks, stolen credentials).
- **Mitigation:**
  - **Strong Authentication (Enhanced):** Current strong authentication with complex passwords is a good foundation. Consider additional measures like:
    - **Rate Limiting:** Implement rate limiting to restrict login attempts and prevent brute-force attacks.
    - **Account Lockout:** Enforce account lockout policies after a certain number of failed login attempts.



- **Google reCAPTCHA (Evaluation):** While internet access might limit OTP usage, evaluate the feasibility of integrating Google reCAPTCHA to add a challenge-response test during login, further deterring automated bots.

#### Threat Category: Injection Vulnerabilities

- **Threat:** Injection attacks (SQL injection, code injection, command injection).
- **Mitigation:**
  - **Input Validation (Enforced):** Implemented strong input validation for every user interface and form to sanitize all user input before processing by the backend system. This prevents malicious code injection attempts.

#### Threat Category: Man-in-the-Middle Attacks

- **Threat:** Man-in-the-Middle (MitM) attacks that intercept data communication.
- **Mitigation:**
  - **SSL/TLS (Maintained):** The existing implementation of SSL/TLS for data in transit ensures secure communication channels and protects against MitM attacks.

#### Threat Category: Data Breaches

- **Threat:** Data breaches compromising sensitive information.
- **Mitigation:**
  - **Password Hashing (Maintained):** Critical data like passwords is encrypted using a strong hashing algorithm (e.g., bcrypt), rendering them unreadable in case of a breach.
  - **Database Separation (Maintained):** The separation of the database server from the PHP application server adds another layer of security. In case of a breach on one system, the other remains protected.

#### Threat Category: Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF)

- **Threat:** XSS attacks injecting malicious scripts into user interfaces, and CSRF attacks forcing unintended actions.
- **Mitigation:**

- **CSRF Protection (Leveraged):** Laravel's built-in CSRF protection effectively mitigates CSRF attacks by embedding a hidden token in forms and validating it during submission.
- **Input Validation (Combined):** The existing input validation practices also help prevent XSS attacks by sanitizing user input before processing, preventing malicious script injection. Laravel's default filtering methods can be further explored for enhanced XSS protection.

### Threat Category: Denial-of-Service (DoS) Attacks

- **Threat:** DoS attacks overwhelm the system with traffic, rendering it unavailable for legitimate users.
- **Mitigation:**
  - **Rate Limiting:** Implement rate limiting techniques to restrict the number of requests a user or IP address can make within a specific timeframe. This can help identify and throttle potential DoS attacks.
  - **Web Application Firewall (WAF):** Consider deploying a WAF to filter incoming traffic and block malicious requests before they reach the application server.
  - **Scalable Infrastructure:** Design a scalable infrastructure that can handle sudden spikes in traffic. This might involve using load balancers and cloud-based resources to automatically distribute workloads.


### Threat Category: Session Hijacking

- **Threat:** Attackers steal session cookies or tokens, allowing unauthorized access to user accounts.
- **Mitigation:**
  - **Secure Cookies:** Implement the `HttpOnly` flag and the `Secure` flag (if using HTTPS) on session cookies to prevent client-side scripting access and ensure transmission only over secure connections.
  - **Session Timeouts:** Set appropriate session timeout periods to automatically log out inactive users after a certain amount of time.
  - **CSRF Protection (Maintained):** Existing CSRF protection measures also help mitigate session hijacking attempts, as

attackers cannot easily forge requests that leverage a valid user's session.

### Threat Category: Insecure Direct Object References (IDOR)

- **Threat:** Exploiting predictable patterns in URLs or identifiers to access unauthorized data.
- **Mitigation:**
  - **Access Control (Enforced):** Implement proper access control mechanisms on the server-side to ensure users can only access data they are authorized to view, regardless of URL manipulation attempts.
  - **Input Validation (Maintained):** Existing input validation practices can help prevent some IDOR attacks by ensuring unexpected user input is not reflected in URLs or identifiers.
- **Requirements Analysis:**
  - Security requirements are refined based on user roles, data access needs, and data sensitivity classification. Examples:
    - **Parents:** Can only access real-time location data for their registered children (verified through a secure process). May have access to basic student information (medium-risk) depending on configuration.
    - **Students:** Can access their current status (class schedule, attendance records - optional). May have limited interaction functionalities (e.g., requesting leave - optional).
    - **Administrators:** Have access to anonymized reports on student location trends (protects privacy) and manage all user accounts and data based on RBAC principles.
  - **Secure Authentication and Authorization:** Implement mechanisms like:
    - **Multi-factor Authentication (MFA):** For high-risk access (e.g., administrator logins) to add an extra layer of security which is removed due to the internet.
    - **Role-based Access Control (RBAC):** Restrict access to specific data based on user roles (e.g., parents cannot access other students' location data).
- **Design:**
  - **Secure System Architecture:** Design a system with data encryption at rest (stored data) and in transit (communication). This may involve:
    - **Transport Layer Security (TLS):** Encrypts communication between users and the system.
    - **Secure Hashing Algorithms (e.g., bcrypt):** Stores passwords securely, preventing attackers from easily accessing them.

- 
- **Secure Coding Practices:** Implement practices to prevent vulnerabilities like:
    - **SQL Injection:** Use prepared statements for database interactions to prevent malicious code injection.
    - **Cross-Site Scripting (XSS):** Properly validate and sanitize user input to prevent attackers from injecting malicious scripts.
  - **Development:**
    - **Code Reviews:** Security analysts conduct code reviews throughout development to identify and fix security flaws early. Utilize static code analysis tools to automate vulnerability detection.
    - **Secure Libraries and Frameworks:** Utilize libraries and frameworks with a strong focus on data security and a good track record for security updates.
    - **Unit Testing and Integration Testing:** Perform these tests to ensure code functionality and identify potential security weaknesses. Include security considerations within the test cases (e.g., testing how the system handles unexpected user input).
  -

## Conclusion

In conclusion, implementing a Secure SDLC methodology throughout the development process is crucial for building a student management and location tracking system that prioritizes the security and privacy of sensitive student data. This detailed approach addresses potential threats at each stage, from initial planning to ongoing maintenance. The use of secure coding practices, robust authentication and authorization mechanisms, and encryption for data at rest and in transit safeguards student information.

Regular vulnerability assessments, penetration testing, and security monitoring further strengthen the system's defenses. User training on secure practices empowers them to contribute to a secure environment.

By adhering to a Secure SDLC and incorporating additional security considerations like data minimization, privacy by design, and a well-defined incident response plan, you can create a system that fosters trust among users, protects student data, and ensures the smooth operation of the educational institution.