

# Secure Input Handling System

**CS3SP Secure Programming**

200240866 Navleen Singh  
200055352 Maryam Asif

Word count: Section 1 and 2: 735

## TABLE OF CONTENTS

Secure Input Handling System .....	3
Section 1:.....	3
Design and Implementation Overview .....	3
Form Specification and Expected Formats for Input:.....	3
Section 2:.....	4
Exception Handling Implementation.....	4
Appendix: .....	6
IIFE Immediately Invoked Function Expression:.....	6
Builder Design Pattern: .....	6
Session Management:.....	6
Tools: .....	8
Testing: .....	8

# Secure Input Handling System

## Section 1:

### Design and Implementation Overview

Our team has successfully developed a web-based input handling system that seamlessly integrates PHP for the backend and JavaScript for the frontend. This dynamic system functions as a comprehensive form, empowering users to input diverse data types, including but not limited to textual information, addresses, phone numbers, postcodes, dates, and JSON preferences. Comprising a total of 11 inputs, each designed to accommodate various formats, our solution exemplifies a versatile and user-friendly approach to data collection. This report intricately examines the architecture of our system, shedding light on the thoughtful design decisions aimed at elevating user experience, guaranteeing data integrity, and reinforcing robust security measures.

### Form Specification and Expected Formats for Input:

On the front end, JavaScript functions employ regular expressions and custom logic for real-time validation, ensuring a seamless user experience. For instance, the system checks the validity of usernames, passwords, and education descriptions using specific criteria. Date of birth undergoes format validation and an age check to ensure users are at least 18 years old. The system also validates phone numbers and enforces different formats based on the selected country. JSON preferences are scrutinized against a predefined schema.

The username, set between 5-50 characters, undergoes front-end checks for length, alphanumeric format, and uniqueness, with the back-end ensuring its uniqueness in the database.

Username:  
  
Enter a username (5-50 characters, alphanumeric and hyphens allowed)

Fig.1-Test Username

Invalid username. Please enter a valid username.

Fig.2-Error Username

Password and retype password fields enforce a complex policy through JavaScript on the front-end, while the back-end hashes and verifies passwords for secure storage.

Passwords do not match. Invalid password. Please enter a password with at least 8 characters, including one uppercase, one lowercase, and one special character.

Fig.3-Password Validation

Education, phone number, date of birth, country of residence, street, number, and postcode undergo similar front-end validations for length and format, with corresponding back-end validations ensuring the integrity of the stored data.

Tell us about your education:  
  
Please lengthen this text to 100 characters or more (you are currently using 96 characters).

Invalid number. Please enter a valid number.

Invalid Date of Birth. Please use the format DD/MM/YYYY and ensure you are at least 18 years old.

Fig.4-Validation

The postcode validation varies based on the country selected.

JSON preferences, capturing additional settings, undergo front-end validation for JSON format and further scrutiny against a predefined schema on the back-end. This comprehensive approach ensures a secure, user-friendly, and standardized input system.

The expected formats are transparently presented to users on the form (Fig.5)

**User Registration Form**

**Username:**  
  
Enter a username (5-50 characters, alphanumeric and hyphens allowed)

**Password:**  
  
Please enter a password with at least 8 characters, including one uppercase, one lowercase, and one special character.

**Retype Password:**

**Tell us about your education:**  
  
Provide a detailed description of your education (100-300 characters) 0 characters

**Phone Number:**  
  
Enter a valid number

**Date of Birth:**  
  
DD/MM/YYYY

**Country of Residence:**

**JSON Preferences:**  
  

```
{
  "notificationSettings": {
    "post": true / false,
    "sms": true / false,
    "push": true / false,
    "frequency": "immediate" / "daily" / "weekly"
  }
}
```

**Register**

Fig.5 Form Inputs

## Section 2:

### Exception Handling Implementation

Validation is a critical aspect of our system, employing a dual-layered approach to ensure security and reliability. Front-end validation, implemented through JavaScript, enhances user experience by preventing incorrect data input and providing real-time feedback on format and length requirements. However, recognizing its limitations, our system places equal emphasis on back-end validation, executed within the robust context of the ConcreteUserBuilder class, acting as the ultimate defence to guarantee only valid and sanitized data is stored.

The ConcreteUserBuilder class plays a crucial role in executing back-end validation checks, safeguarding data integrity and security. This includes enforcing specified formats and constraints for various input fields, such as username, password criteria, education details, phone number, date of

birth, country of residence, street format, number format, postcode based on the country of residence, and validating JSON preferences against a predefined schema. These validations collectively contribute to a robust back-end validation strategy, fortifying the application against potential security threats. (Fig.6)

```
// Function to sanitize input
function sanitizeInput($input) {
    return htmlspecialchars(trim($input), ENT_QUOTES, 'UTF-8');
}

// Function to validate username format
function isValidUsername($username) {
    // Allow alphanumeric characters and hyphens, 5 to 50 characters
    return preg_match('/^[a-zA-Z0-9-]{5,50}$/', $username);
}

// Function to validate password format
function isValidPassword($password) {
    // Require 1 upper, 1 lower, 1 special character, and at least 8 characters
    return preg_match('/^(?=.*[A-Z])(?=.*[a-z])(?=.*[!@#%&*]).{8,}$/', $password);
}

// Function to validate education format
function isValidEducation($education) {
    // Allow only text with a minimum length of 100 and a maximum length of 300 characters
    $trimmedEducation = trim($education);
    $educationLength = mb_strlen($trimmedEducation);

    return $educationLength >= 100 && $educationLength <= 300;
}
```

Fig.6 Validation

Exception handling is central to maintaining system stability, especially in the face of unexpected scenarios or invalid inputs. In the process.php file, a custom exception class, ValidationException.php, is created and utilized. This exception class catches errors in the back-end validation process within the ConcreteUserBuilder class. When validation fails, a ValidationException is thrown, and its message is logged. Additionally, PHP's robust exception handling mechanism is employed to address different types of exceptions. Notably, database-related exceptions are handled separately from unexpected exceptions, streamlining error resolution, and aiding in identifying and mitigating issues promptly. (Fig.7a/b) This method not only avoids abrupt crashes but also provides meaningful error messages, facilitating issue identification during both development and maintenance phases.

```
function displayErrorAndLog($errorMessage) {
    echo $errorMessage;
    error_log($errorMessage);
    exit();
}
```

Fig.7a Form Inputs

```
} catch (mysqli_sql_exception $e) {
    // Handle database-related exceptions
    displayErrorAndLog("A database error occurred. Please try again later. Details: " . $e->getMessage() . "\n" . $e->getTraceAsString());
} catch (Exception $e) {
    // Handle other unexpected exceptions
    displayErrorAndLog("An unexpected error occurred. Please try again later. Details: " . $e->getMessage() . "\n" . $e->getTraceAsString());
} finally {
    // Close the database connection
    $conn->close();
}
?>
```

Fig.7b Display errors and log

Our implementation ensures that even in a production environment where users may submit diverse and potentially malicious inputs, the exception-handling strategy guarantees application resilience,

preventing crashes and contributing to a more secure and reliable user experience. The collective implementation of both front-end and back-end validation, along with robust exception handling, establishes a dependable framework that persists in the application under abuses and extreme cases, promoting data integrity and security.

In conclusion, the input handling system successfully combines front-end and back-end validation, creating a robust and secure application. The implemented exception handling mechanisms fortify the system's resilience, ensuring a reliable user experience even in the face of potential vulnerabilities or abuses.

## **Appendix:**

We chose PHP for backend development due to its strong integration with MySQL, facilitating seamless database interactions. JavaScript, a fundamental language for web development, was selected for frontend tasks, ensuring dynamic and interactive user interfaces. This combination aligns with web development best practices, leveraging the strengths of each language in their respective domains.

### **IIFE Immediately Invoked Function Expression:**

The use of Immediately Invoked Function Expressions (IIFE) was a deliberate choice in our implementation. An IIFE is a self-executing anonymous function that executes immediately after being defined. We employed IIFE primarily to encapsulate variables and functions, preventing them from polluting the global scope. This encapsulation enhances code modularity and minimizes the risk of naming conflicts, contributing to a more maintainable and secure codebase. Additionally, IIFE helps achieve data privacy, as variables defined within the function are not accessible from the outside scope, reinforcing the security of our input handling system.

### **Builder Design Pattern:**

The Builder Design Pattern played an essential role in structuring and constructing user objects within our system. Implementing the Builder pattern facilitated a clean separation of concerns, improving the code's readability. The `ConcreteUserBuilder` class, responsible for constructing user objects, enabled the step-by-step setting of user attributes. This modular approach not only simplified the code but also allowed for the easy addition of new attributes or modifications without significantly impacting the existing codebase. The Builder pattern's flexibility and scalability proved invaluable in handling various user attributes within our form, enhancing the maintainability and extensibility of our input handling system.

Additionally, we employed the Immutable Builder variant to enhance the builder class's immutability. By modifying the `build` method to clone the `User` object before returning it, we ensured that users of the builder couldn't modify the built object after its creation. This approach adds an extra layer of security and stability to our codebase, aligning with best practices for creating robust and maintainable systems.

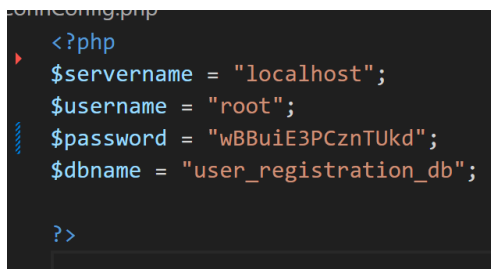
### **Session Management:**

Session management is a critical aspect of our web-based input handling system, contributing to both security and user experience. PHP's built-in session management mechanism was utilized to maintain user sessions throughout their interaction with the system. Sessions play a crucial role in securely storing user-specific information and persisting data across multiple pages. By leveraging sessions, we ensured that users remained authenticated during their interactions, enhancing the

overall security of our application. Additionally, session variables facilitated the storage of temporary data, such as CSRF tokens, crucial for preventing cross-site request forgery attacks. The implementation of session management contributes significantly to the reliability and security of our input handling system.

## Server Hardening

A crucial initial step involves altering the default password for phpMyAdmin, which is initially set to blank, exposing vulnerability to potential attackers who can easily identify and access the database. Changing default passwords is imperative to mitigate risks such as data breaches, unauthorized control, and misuse (Nayak, G.R., 2020). (Fig.8)



```
connect.php
<?php
$servername = "localhost";
$username = "root";
$password = "wBBuiE3PCznTUKd";
$dbname = "user_registration_db";

?>
```

Fig.8 Secure password

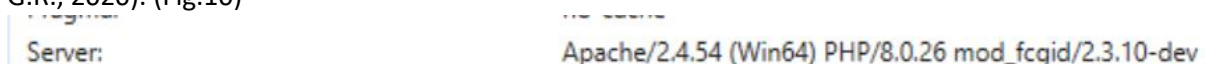
In addition, enhancing server security can be achieved by making specific modifications in the httpd.conf file. By setting ServerSignature to Off, Apache refrains from revealing sensitive details, such as the Apache version and the operating system, thereby bolstering security. Fig.9



*Apache/2.4.54 (Win64) PHP/8.0.26 mod\_fcgid/2.3.10-dev Server at localhost Port 80*

Fig.9 Apache details

When ServerSignature is enabled (default), sensitive information is displayed, whereas disabling it conceals this information. Furthermore, configuring ServerTokens to Prod conceals the Apache version in the server response headers, contributing to a more secure server environment (Nayak, G.R., 2020). (Fig.10)



Server: Apache/2.4.54 (Win64) PHP/8.0.26 mod\_fcgid/2.3.10-dev

Fig.10 Apache details

Maintaining an updated server is crucial for security, as updates often include patches for vulnerabilities. Alternatively, addressing specific security flaws without updating the entire server is a viable option (Nayak, G.R., 2020).

To fortify data security, particularly regarding sensitive information like user passwords in a database, it is imperative not to store passwords as plaintext. Storing passwords in plaintext exposes sensitive credentials if unauthorized access to the database occurs. Utilizing a hashing mechanism transforms passwords into irreversible values, providing robust protection against potential hackers (php.net, n.d.). The implementation of this hashing mechanism is illustrated in the accompanying image. (Fig.11)

<input type="checkbox"/>	Edit Copy Delete	8 test1	\$2y\$10\$inBvmmf29UifRDYdZzuxH0kg5SL68GSLU/2J0w...	AWDSHUPNEMKGL_HJKGBJBIEVGFHUDGOSFACSWVDEHfJRGKTL...	07448520095	2001-01-20	UK	Stuart Road	20	B65 9JA	{ "notificationSettings": { "post": true, ...	2023-12-19 23:23:14
<input type="checkbox"/>	Edit Copy Delete	9 test2	\$2y\$10\$1dG8WwMDMYPc2QAjgmet3J0Xe 68vVFBF 1a63w1...	fcgvahbjwdctfo.vgzhkkgfhdqgswevqafghshwipkemt...	07556830092	2022-09-19	UK	Stuart Road	20	B65 9JA	{ "notificationSettings": { "post": true, ...	2023-12-20 00:45:00
<input type="checkbox"/>	Edit Copy Delete	10 ulehousenf	\$2y\$10\$GxTVd8mH8PNEj3sdB s1en67UVP9Q 3R0VupP44...	aashvgzuehfcvslgahgpfchwaqfcaashncdshref v...	+447448520095	2001-01-20	UK	Stuart Road	20	B65 9JA	{ "notificationSettings": { "post": true, ...	2023-12-20 21:32:47

Fig.11 Hashing Password

## Tools:

The provided web application utilizes PHP for server-side logic, HTML and CSS for page structure and styling, and JavaScript to enhance user interactivity and perform client-side validation. The MySQL database stores and retrieves user registration data, with SQL queries executed through PHP. The application incorporates a JSON Schema Validator library for validating user preferences. (Justinrainbow (no date); *JSON Schema* (no date)) This design choice was made to meet specific requirements of the application, contributing to data consistency and integrity. **Fig.**

Composer is employed for dependency management, and Content Security Policy headers are set for added security. Random bytes are utilized to generate CSRF tokens, preventing cross-site request forgery attacks. Git, a version control system, is commonly used for code management. Collectively, these tools contribute to the functionality, security, and maintainability of the user registration system.

## Testing:

### PHPUnitTesting:

PHPUnit testing was conducted on the PHP codebase using version 9.6.15, verified by running the command `phpunit UserTest.php`. The results indicate success, with all 15 assertions passing. (Fig.12) This testing approach ensures the reliability and correctness of the code, contributing to the overall quality assurance of the software development process.

```
C:\wamp\www\Secure-Programming>phpunit UserTest.php
PHPUnit 9.6.15 by Sebastian Bergmann and contributors.

.                                                                    1 / 1 (100%)

Time: 00:00.098, Memory: 4.00 MB

OK (1 test, 15 assertions)
```

Fig.12 PHPUnit

## ZAP Zed Attack Proxy

ZAP (Zed Attack Proxy) was employed for fuzzy testing to enhance the security robustness of our application. The testing revealed a set of 10 alerts, encompassing areas such as Content Security Policy (CSP) Header absence, missing HttpOnly flag for cookies, absence of the SameSite attribute in cookies, and the lack of the X-Content-Type-Options Header. Additionally, the assessment identified potential issues related to authentication requests, information disclosure through suspicious comments, loosely scoped cookies, and aspects associated with modern web application security. Specifically, session management responses were flagged in four instances, and the User Agent Fuzzer detected 36 anomalies. To fortify the application, addressing these identified areas, including the implementation of robust CSP headers, securing cookies with HttpOnly and SameSite attributes, and ensuring proper authentication mechanisms, is recommended. This proactive approach aligns with industry best practices and bolsters the overall security posture of our application.



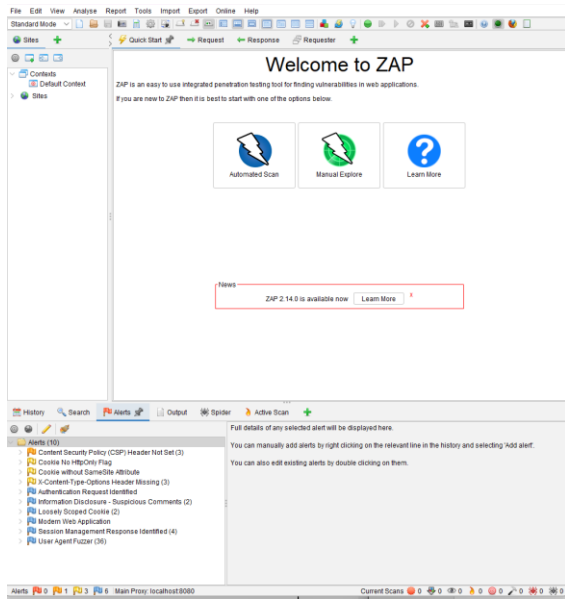


Fig.13 ZAP

## References

*JSON Schema* (no date) *JSON Schema - Creating your first schema*. Available at: <https://json-schema.org/learn/getting-started-step-by-step> (Accessed: 21 December 2023).

Justinrainbow (no date) *Justinrainbow/JSON-schema: PHP implementation of JSON schema. fork of the <http://jsonschemaphp.sourceforge.net/> Project, GitHub*. Available at: <https://github.com/justinrainbow/json-schema> (Accessed: 21 December 2023).

Php.net, (n.d.). Password hashing function, php. Retrieved from <https://www.php.net/manual/en/ref.password.php> (Accessed: 09 November 2023).

Nayak, G.R. (2020). Secure wamp server with ease, Bobcares. Retrieved from <https://bobcares.com/blog/secure-wamp-server/> (Accessed: 09 November 2023)