Software Requirements Specification

Name: Uma Sree. U

Class: PETV83L

Roll no: 13

Reg no: 12303980

Project title: Secure Login Form with GOOGLE reCAPTCHA

Submitted to: Mr. Manish Singh

Course: Cyber Security Essentials

Date of Submission: 08/07/2025

Table of Contents:

List of Figures

1.0. Introduction

1.1. Purposes

1.2. Scope of Project1

1.3. Glossary2

1.4. References

1.5. Overview of Document2

2.0. Overall Description

2.1System Environment

2.2 Functional Requirements Specification5

2.2.1 Reader Use Case5

Use case: Search Articles

2.2.2Author Use Case

Use case: Submit Article

2.2.3Editor Use Cases

Use case Update Author

Use case: Update Reviewer

Use case: Update Article

Use case: Receive Article10

Use case: Assign Reviewer 11

Use case: Receive Review11

Use case: Check Status12

Use case: Send Response12

Use case: Send Copyright13

Use case: Remove Article14

Use case: Publish Article14

2.3User Characteristics15

2.4Non-Functional Requirements15

3.0. Requirements Specification

3.1 External Interface Requirements17

3.2 Functional Requirements 17

3.2.1Search Article17

3.2.2Communicate18

3.2.3Add Author18

32.4Add Reviewer 19

32.5Update Person19

32.6Update Article Status 20

3.2.7Enter Communication20

3.2.8Assign Reviewer 21

3.2.9Check Status 21

3.2.10Send Communication22

3.2.11Publish Article22

3.3Detailed Non-Functional Requirements23

INTRODUCTION:

1.1 PURPOSE:

The purpose of this project is to develop a secure and interactive login form that integrates Google reCAPTCHA v2 to ensure that only human users can access the system. In the current digital environment, websites and web applications are constantly targeted by automated bots that attempt to perform malicious actions such as brute force logins, fake registrations, and spam submissions. This project addresses that problem by adding a layer of protection that verifies the authenticity of the user before the login form is submitted. The reCAPTCHA system serves as a challenge-response mechanism that helps in distinguishing human users from bots in a simple and effective way. The project is built using HTML, CSS, and PHP, and it demonstrates how to implement a real-time CAPTCHA verification system both locally using XAMPP and online through a web hosting platform. The overall goal is to enhance the security and reliability of user authentication while maintaining a smooth and accessible user experience.

1.2 Scope of Project:

The scope of this project includes the design, development, and deployment of a secure login form integrated with Google reCAPTCHA v2 for human verification. The system is designed to operate in both local and hosted environments, making it versatile for academic and real-world usage. It covers the basic functionality of a login interface, where users can enter their credentials, and must successfully complete a CAPTCHA verification before the form is processed. The server-side script, developed using PHP, handles the verification by interacting with Google's reCAPTCHA API to validate the user's response. This adds a significant layer of protection by filtering out non-human activity such as bots and automated scripts, which are commonly used for spamming or attacking web applications. The project demonstrates how modern web technologies and third-party APIs can be combined to improve security in even the simplest forms. While the current implementation focuses on basic login validation, the system can be extended in future to include

features like database authentication, password encryption, user sessions, and error logging. This project is particularly beneficial for students and beginners looking to learn about secure form development, reCAPTCHA integration, and basic server-side validation in a PHP-based environment.

1.3 Glossary:

TERMS	DEFINITION
1. CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart; used to block bots.
2. reCAPTCHA	A free CAPTCHA service from Google that protects websites from spam and abuse.
3. HTML	Hyper Text Markup Language used to create the structure of web pages.
4. CSS	Cascading Style Sheets used to design and style HTML elements on a webpage.
5. PHP	A server-side scripting language used for backend logic and form handling.
6. XAMPP	A local web server package that allows developers to run PHP and MySQL locally.
7. Local Host	Refers to the local computer being used as a server during development.
8. Site Key	A public key provided by Google reCAPTCHA to be embedded in the HTML form.

9. Secret Key	A private key used server-side to verify CAPTCHA responses with Google's API.
10. API	Application Programming Interface; allows software components to communicate.
11. Form	An HTML element that collects user input and submits it to a server for processing

1.4 REFERENCES:

1. Google Developers. “reCAPTCHA v2 Documentation.” Available at: https://developers.google.com/recaptcha/docs/display
2. W3Schools. “HTML Forms.” Available at: https://www.w3schools.com/html/html_forms.asp
3. W3Schools. “PHP Form Handling.” Available at: https://www.w3schools.com/php/php_forms.asp
4. XAMPP Official Website. “Apache Friends – XAMPP.” Available at: https://www.apachefriends.org/index.html

1.5 OVERVIEW OF DOCUMENT:

This document provides a detailed description of the software requirements and technical implementation of a secure login form integrated with Google reCAPTCHA v2. It is intended to guide developers, reviewers, and stakeholders through the purpose, scope, architecture, and behavior of the system.

The document begins by explaining the objective of the project, the background of CAPTCHA technologies, and the importance of user verification in modern web applications. It then outlines the key components of the system, including the technologies used such as HTML, CSS, PHP, and the Google reCAPTCHA API.

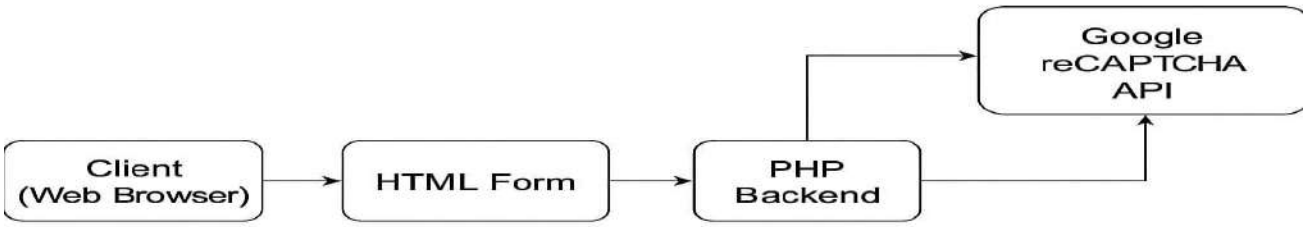
Additionally, the document defines the functional and non-functional requirements of the system, explains how CAPTCHA verification is implemented, and describes the system's environment and limitations. It also includes sections such as the glossary, references, and other supporting information that help in understanding the complete workflow of the project.

This SRS serves as a foundational document for implementing the project in both local and online environments, and can be extended in the future with more advanced features such as database integration and user sessions.

2.0 OVERALL DESCRIPTION:

System Environment:

System Environment



The system is developed to operate in a local environment using XAMPP, also be hosted online using a web hosting service such as 000webhost. The c requires a web browser such as Google Chrome or Mozilla Firefox, while t er-side requires PHP 7.0 or above. An internet connection is necessary to ren date the reCAPTCHA widget via the API.

2.2 Functional Requirements Specifications:

The system must allow users to enter their details through a web form and verify their identity using Google reCAPTCHA v2. When the form is submitted, the backend PHP script should validate the CAPTCHA response and process the data only if the user is verified as human. If the CAPTCHA fails, the submission must be rejected with an error message.

2.2.1 Reader use-case

The Reviewer (or Reader) is a user role responsible for checking and validating the functionality and security of the login system. This use case describes how a reviewer interacts with the application to ensure that CAPTCHA verification is working properly, and that user form submission is handled correctly. The reviewer may test the form multiple times, attempt with and without CAPTCHA, and verify success or failure messages. Their main goal is to evaluate the system's behavior under various input conditions and ensure it meets the project requirements.

ıse: Reader

ːader is a person who accesss the secure login form integrated with
ɔogle rẽCAPTCHA. It attempts to view or read content behind an
ɡin. It is simulates he experience of an end-user. Interact with tl
ɾm by loading the webpage displaying the login form. The reCAPT
dget is automatically enabled, visually confirms the presence of th
APTCHA, checks the "I'm not a robot" checkbox, and click the "Sub
ıtton to test the functionality or gain access.

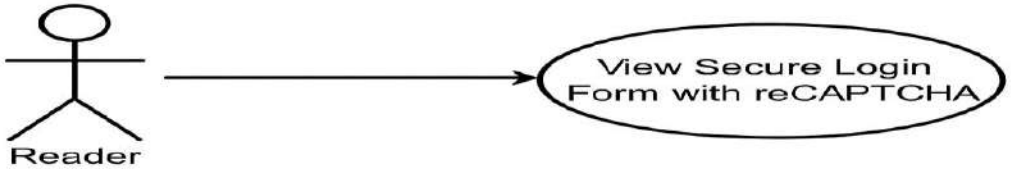


Figure: *Usse Case: Reader*

Reader use case diagram

2.2.2 Author use case:

ıse: Author

ı author is a person who loginsthrough the secure login form inte
ith Google reCAPTCHA. It listens for keyboard events while ty
ıring login. Interact with the form by loading the webpage displayi
gin form. The reCAPTCHA widget is automatically enabled, visi
ɔnfirms the presence of the reCAPTCHA, checks the "I'm not a robo
ıeckbox, and click the "Submit" buiton to gain access.

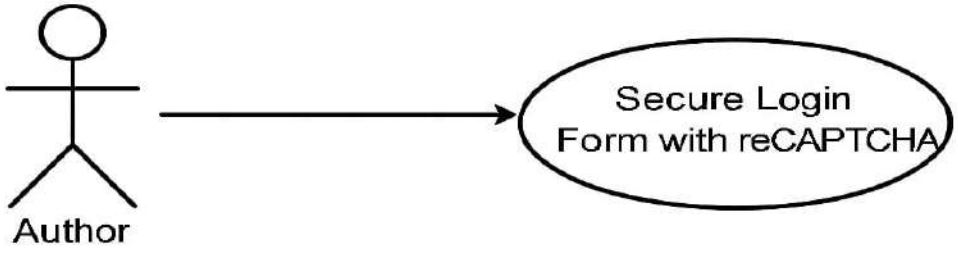


Figure: *Use Case: Author*

2.2.3 Editor use case

ce: Editor

n editor is a person who can access the secure login interface with
CAPTCHA enabled, listens for keyboard events during typing of lo
etails. Visuanatly confirms the presence of the reCAPTCHA widge
rough smart ther checkbox and clicks tab−access the system
ccess the system, and to access the system.

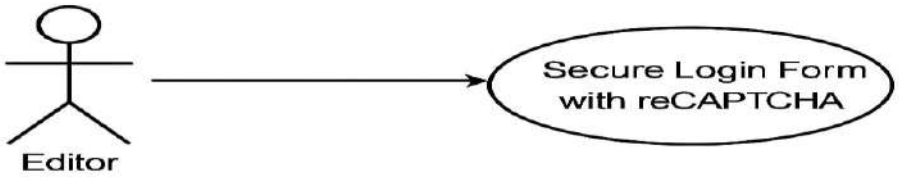


Figure: *Use Case: Editor*

Use Case: Update Author

This use case allows an editor or system admin to update the author's details, including their name, email, and credentials. This may be necessary when correcting information or modifying user roles within the system.

Use Case: Update Reviewer

This use case allows an editor or system admin to update the reviewer's profile or assigned tasks. This could include changing reviewer status, email, or assigned article categories.

Use Case: Update Article

This use case enables an author to update their submitted article before review. It may include editing the title, abstract, content, or attachments before it is finalized for review.

Use Case: Receive Article

This use case refers to the system receiving an article from the author and storing it in the database. It validates the content, stores metadata, and notifies the editorial team.

Use Case: Assign Reviewer

The editor assigns one or more reviewers to a submitted article based on topic match, availability, and expertise. The system sends notification emails to assigned reviewers.

Use Case: Receive Review

The system receives the completed review from a reviewer and logs it in the database. It also updates the article's review status and alerts the editor.

Use Case: Check Status

An author or reviewer can check the current status of an article (e.g., under review, accepted, rejected). The system displays the timeline and latest updates.

Use Case: Send Response

The editor sends a decision response to the author based on the review outcome (e.g., accept, revise, reject). The system logs the response and notifies the author.

Use Case: Send Copyright

If an article is accepted, the system prompts the author to submit copyright transfer forms. The system tracks submissions and confirms legal compliance.

Use Case: Remove Article

The editor or admin can remove an article from the system if it is withdrawn or disqualified. The system logs the removal with reasons for recordkeeping.

Use Case: Publish Article

Once all requirements are met, the final article is published online or in print. The system updates its status, notifies users, and makes it publicly accessible.

2.3 User Characteristics:

The system is intended for users with basic knowledge of web browsing and form submission. Users are expected to be able to interact with a simple user interface to enter personal information and complete CAPTCHA verification. No advanced technical skills are required. The primary users include authors submitting information, reviewers verifying functionality, and editors managing the system. Reviewers and editors may have additional responsibilities such as evaluating form logic, security features, and correctness of CAPTCHA responses. All users are assumed to have internet access and access to a modern web browser. Familiarity with basic computer operations and form-based interactions is sufficient for using the system effectively.

2.4 Non-Functional Requirements:

The system is designed to be efficient, secure, and user-friendly. It should provide fast response times for both form rendering and CAPTCHA verification to ensure a smooth user experience. The user interface must be simple, clean, and responsive across different screen sizes and devices. The CAPTCHA should load reliably and function properly even under moderate network latency. The system should maintain data privacy by ensuring that sensitive information like passwords is never stored or transmitted insecurely. Server-side validation using PHP must be implemented to avoid bypassing client-side restrictions. The system should be highly available in both local and hosted environments and should not crash or hang during multiple submissions. Scalability is not a primary concern for this basic version but can be considered for future integration with databases or user accounts. The codebase should be modular and easy to update if the CAPTCHA keys change or if new security features are added. Overall, the system must remain robust, accessible, and maintainable while fulfilling its security objectives.

3.0 Requirement Specification:

3.1 External Interface Requirements:

The system interacts with several external components to function effectively. The primary user interface is a web-based form designed using HTML and styled with CSS, allowing users to enter their personal information and solve the reCAPTCHA challenge. This interface must be compatible with modern web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge. Users should be able to access the form using standard desktop or laptop systems, and optionally mobile devices with proper screen responsiveness.

The hardware interface includes any personal computer or laptop used to run the application via a local server (XAMPP) or through a web hosting service. There are no special hardware requirements other than basic computing resources like a keyboard, mouse, and screen.

The software interface involves the interaction between the front-end form and the server-side script written in PHP. The PHP script sends a POST request to

the Google reCAPTCHA API using the provided secret key, and processes the JSON response to determine if the user has passed the CAPTCHA test. The PHP environment must be version 7.0 or higher, and the web server must support HTTP POST requests and allow outbound connections to Google's servers.

The communication interface depends on internet connectivity, as the system must communicate with Google's reCAPTCHA verification service via a secure HTTPS connection. Without a valid internet connection, the CAPTCHA cannot load or be verified, causing the form submission to fail. The application should handle such cases gracefully by alerting the user if the CAPTCHA is not available or if the verification fails due to connectivity issues.

Overall, the external interface requirements ensure that the system interacts seamlessly with the user, the local server, and the external CAPTCHA verification service, while maintaining compatibility, performance, and security standards.

3.2 Functional Requirements:

ID	Functional Requirement	Description
3.2.1	Search Article	Allows users (authors, reviewers, or editors) to search the article database using filters like title, author's name, keywords, and submission date. It enhances navigation and speeds up information retrieval. Advanced search may include filtering by submission status, review history, category, and publication date. The search feature should support partial matches and display results dynamically.
3.2.2	Communicate	Provides a structured channel for authors, reviewers, and editors to exchange messages. The communication is stored in the system for

		future reference and transparency. Supports threaded messaging, timestamps, and role-based access. Users can mark messages as read/unread, and search within conversations.
3.2.3	Add Author	Enables the system to register a new author, requiring input such as name, email, affiliation, and user credentials. Ensures the author can later submit and track articles. Validates data and prevents duplicate entries. Authors receive a welcome email with instructions.
3.2.4	Add Reviewer	Allows administrators to register new reviewers with details like name, email, area of expertise, and availability. This supports the assignment process for peer review. Reviewers can later update their profile and availability. The system checks for existing users to avoid redundancy.
3.2.5	Update Person	Permits updating of any registered user's profile data (author or reviewer), including contact info, institution, or assigned roles. Keeps user records up to date and consistent across the system. Changes are logged for audit purposes. Updated data reflects instantly across all linked articles or reviews.
3.2.6	Update Article Status	Used by editors to modify the current status of an article, such as "Under Review," "Accepted," "Rejected," or "Published." Essential for tracking progress. Automatically notifies users of status changes. Allows optional editor comments and timestamps for each update.
3.2.7	Enter Communication	Editors or reviewers can log specific comments or decision notes related to article reviews, which are stored as part of the review trail. Helps track editorial decisions and provides a record for accountability. These entries remain private and are visible only to permitted roles.
3.2.8	Assign Reviewer	Editors select and assign appropriate reviewers to articles based on topic relevance, workload, and past activity. Notifications are automatically

		triggered. Includes conflict of interest checks. Reviewer assignment is logged for accountability and audit purposes.
3.2.9	Check Status	Authors and reviewers can view the real-time status of their submissions or assignments. Displays timestamps, decisions, and pending actions. Encourages transparency and reduces manual status queries. The feature supports notification badges and filtering based on action required.
3.2.10	Send Communication	Editors use this function to formally send system-generated or manual messages regarding article decisions, revisions, or feedback to authors. Emails are sent along with in-system notifications. Message templates can be reused to ensure consistency and reduce manual input.
3.2.11	Publish Article	Once accepted, the editor can move the article to a published state, making it publicly viewable and marking it complete in the system. Published articles are tagged with DOI or reference numbers. The system updates metadata and archives a final version for backup.

3.3 Non-Functional Requirements:

The system is developed with an emphasis on quality attributes such as performance, security, usability, maintainability, and reliability. These non-functional aspects ensure that the solution not only works correctly but also provides a smooth and secure experience to all users.

Performance:

The system must respond quickly to user inputs and actions. CAPTCHA should load within 2–3 seconds under normal internet conditions, and form validation and submission should be processed within 1 second after clicking "Submit". The backend must handle server-side validation efficiently without delaying the user experience.

Security:

Security is paramount due to the handling of personal information. The Google reCAPTCHA v2 is implemented to prevent bots and spam attacks. All form inputs are sanitized to avoid injection attacks. Although no passwords are stored in this version, future versions should implement secure hashing (like bcrypt) for credential storage. Communication with Google's API occurs over secure HTTPS. Role-based access (editor, reviewer, author) is encouraged for future scope.

Usability:

The system interface is simple, clean, and visually engaging. Forms are structured clearly with proper labels, tooltips, and real-time feedback. Error messages guide users toward correction. Colors and layout are chosen for accessibility and clarity. No technical expertise is needed to operate the form.

Reliability:

The application must function consistently across different sessions, systems, and browsers. CAPTCHA codes are verified correctly with no false negatives/positives. System components (like PHP processing, form rendering, and API communication) must remain stable even during repeated usage.

Portability:

The application is portable across environments. It can run on local machines via XAMPP and be deployed on any standard web server. It is OS-independent and browser-compatible. Hosting it online allows global access without requiring XAMPP.

Maintainability:

Code is modular and separated into layers: HTML for structure, CSS for styling, and PHP for logic. CAPTCHA site/secret keys can be updated without rewriting major parts of the code. Comments are included to ease future updates or debugging.

Scalability:

While initially designed for local use, the structure supports scaling. The form can be integrated with a database, user login system, or admin panel. CAPTCHA can also be upgraded to v3 or invisible reCAPTCHA without major rewrites.

Availability:

The system must have 99% uptime in hosted environments. For local use, availability depends on Apache and internet access. Any failure (like CAPTCHA not loading) should be detected and handled gracefully with a user-friendly message.

Interoperability:

The system communicates effectively with Google's external CAPTCHA API and integrates seamlessly with the local PHP server (XAMPP). This confirms that it can work with third-party services without breaking compatibility.