www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 10, October 2025, pp : 1489-1493

e-ISSN : 2583-1062
Impact Factor : 7.001

DATA VALIDATION

Chetana Bobade[1], Siddhi Chimkurkar[2], Tanisha Khangar[3], Tanvi Kurhadkar[4], Mohammad Tahir[5]

[1,2,3,4]Student, Department Of Information Technology, Nagpur, Maharashtra, India.

[5]Professor, Department Of Information Technology, Nagpur, Maharashtra, India.

ABSTRACT

This project presents a full-stack web application focused on validating and securely storing user data. The frontend is developed using React with React Hook Form, providing a responsive form that collects user inputs such as name, email, phone number, and age. Client-side validation ensures that names contain only alphabets and spaces, emails include an '@' and end with '.com', phone numbers are exactly 10 digits, and age is restricted between 18 and 120. This immediate validation improves user experience by preventing incorrect data submission.

On the backend, FastAPI is used alongside Motor for asynchronous communication with a MongoDB database. The server performs additional validation and prevents duplicate records based on email and phone number, while assigning unique IDs and timestamps to each entry. Environment variables and security features are managed using libraries like python-dotenv and

PyJWT, enabling safe and confgurable deployment. The project also integrates testing and code quality tools such as pytest and fake8, ensuring maintainability. Together, this solution ensures robust, scalable, and efcient data handling suitable for modern web applications.

Keywords: Data validation, Full-Stack Web Application, ReactstAPI, MongoDB, Client-Side Validation, Server-Side Validation.

1. INTRODUCTION

In today's digital world, collecting accurate and validated user data is critical for building reliable and secure applications. Incorrect or malicious data input can lead to faulty analytics, poor user experience, and security vulnerabilities. Hence, implementing robust data validation both on the client and server side is essential to ensure data integrity before storage.

This project develops a full-stack web application that demonstrates efective data validation techniques. The frontend uses React along with React Hook Form to provide real-time validation feedback, ensuring users enter valid names, emails, phone numbers, and age values. On the backend, FastAPI with Motor enables asynchronous processing and additional validation, while managing data storage in MongoDB. The system also includes measures to avoid duplicate entries and uses environment-based confgurations for fexibility and security. This approach exemplifes best practices in modern web development for handling user data efciently and securely.

2. METHODOLOGY

2.1 System Architecture

The architecture of the application is designed to be modular, scalable, and efcient by separating frontend and backend responsibilities:

2.2 Frontend (Client-side):

Developed using React, the frontend is responsible for rendering the user interface and collecting user input through a form. It leverages React Hook Form for efcient form management and client-side validation to provide immediate feedback to users, reducing invalid submissions.

2.3 Backend (Server-side):

The backend is built using FastAPI, a modern, asynchronous Python web framework. It exposes RESTful API endpoints that the frontend consumes. The backend performs server-side validation, handles business logic, checks for duplicates, and manages database operations to ensure data consistency and security.

2.4 Database Layer:

MongoDB is used as the NoSQL database for storing user data. Motor, an asynchronous MongoDB driver for Python, facilitates non-blocking database operations to maintain high performance under concurrent requests.

2.5 Communication:

The frontend and backend communicate via HTTP using Axios, with JSON as the data interchange format. CORS middleware is confgured on the backend to allow secure cross-origin requests from the frontend.

www.ijprems.com
editor@ijprems.com
INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 10, October 2025, pp : 1489-1493
e-ISSN : 2583-1062
Impact Factor : 7.001

2.6 Tools and Libraries

The project utilizes a combination of tools and libraries that streamline development, ensure data integrity, and maintain code quality:

2.7 Frontend Libraries:

React for building dynamic UI components.

React Hook Form for managing form state and validation.

Axios for making HTTP requests to backend APIs.

React Router to enable client-side routing between the form and data display pages.

2.8 Backend Libraries:

FastAPI for creating high-performance, asynchronous API endpoints.

Motor for asynchronous interaction with MongoDB.

Pydantic for data parsing and validation models.

python-dotenv to manage environment variables securely.

PyJWT and CORS Middleware to handle authentication and cross-origin resource sharing, respectively.

2.9 Development and Quality Tools:

pytest for testing.

black and fake8 for code formatting and linting.

mypy for static type checking.

2.10 Workfow

The application workfow ensures smooth data handling from input to storage and retrieval:

Data Entry and Client-Side Validation:

Users fll out the form felds (name, email, phone, age) on the frontend. React Hook Form applies validation rules such as ensuring the name contains only alphabets, email contains '@' and ends with '.com', phone number is exactly 10 digits, and age is within the valid range (18–120). Invalid inputs immediately show error messages.

Data Submission and Server-Side Validation:

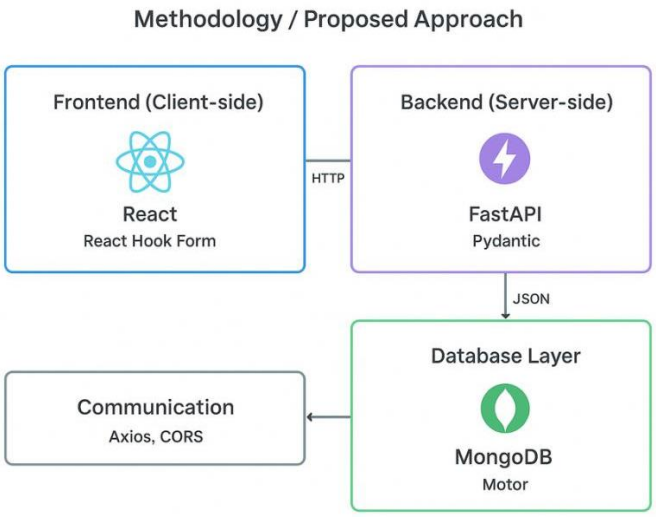
Upon passing client-side validation, the data is sent via Axios POST request to the FastAPI backend. The backend again validates the data to prevent any bypass of client-side rules, checks for duplicate email or phone number entries, and rejects duplicates with appropriate error messages.

2.11 Data Processing and Storage:

Valid data is augmented with a unique UUID and timestamp, then stored asynchronously in MongoDB using Motor. This ensures data integrity and traceability.

2.12 Data Retrieval and Display:

Users can navigate to the stored data page, where the frontend sends a GET request to the backend to fetch all user records. The backend returns the data after fltering to ensure only valid entries are sent. The frontend displays this data in a tabular format, allowing users to verify stored records.



INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)
(Int Peer Reviewed Journal)
Vol. 05, Issue 10, October 2025, pp : 1489-1493
www.ijprems.com
editor@ijprems.com
e-ISSN : 2583-1062
Impact Factor : 7.001

3. MODELING AND ANALYSIS

3.1 System Architecture

The system is divided into two main components: the frontend and the backend. The frontend is developed using React, which provides a user-friendly interface for data input and validation. It utilizes React Hook Form to handle form state and validation logic efciently. The backend is built using FastAPI, a high-performance asynchronous Python framework, responsible for processing, validating, and storing data. MongoDB serves as the database to persist user information. Communication between frontend and backend occurs via RESTful API calls.

3.2 Tools and Libraries

React & React Hook Form: Used for building the interactive user interface and managing form validation on the client side, ensuring immediate feedback for users.

FastAPI: Handles server-side validation and API endpoints. It uses Pydantic models to enforce strict data validation rules.

MongoDB & Motor: MongoDB stores user data in a fexible document format, while Motor provides an asynchronous driver to interact with the database efciently.

Axios: Used on the frontend to send HTTP requests to the backend API.

CORS Middleware: Confgured in FastAPI to allow secure cross-origin requests between frontend and backend during development.

Python-dotenv: Manages environment variables securely, including database connection strings and API keys.

3.3 Workfow

User Input: The user flls out the form with felds including name, email, phone number, and age.

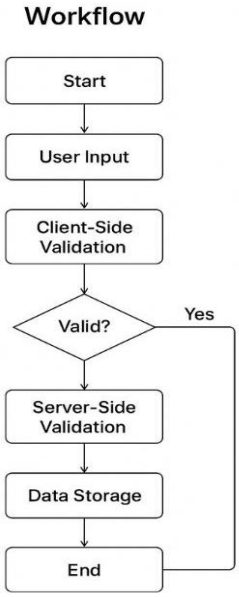
Client-Side Validation: React Hook Form validates inputs against predefned rules (e.g., name must not contain numbers, email must include "@" and end with ".com", phone must be 10 digits, age between 18 and 120). Errors are displayed immediately if any validation fails.

Form Submission: On passing client-side checks, the form data is sent to the backend API via Axios.

Server-Side Validation: FastAPI revalidates the data using Pydantic models to ensure integrity and security. It also checks for duplicates in the MongoDB database.

Data Storage: Valid data is stored in MongoDB with unique user IDs and timestamps.

Data Retrieval: Stored valid records can be fetched from the backend and displayed in a tabular format on the frontend under the "Stored Data" section.



4. RESULT AND DISCUSSION

4.1 Form Validation Performance

The implemented data validation website efectively enforces both client-side and server-side validation rules. When users input their details in the form, instant validation feedback is provided, helping users correct errors before submission. For example:

Names containing numbers trigger immediate error messages.

Emails missing "@" or not ending with ".com" are rejected.Phone numbers must be exactly 10 digits, else an error is displayed.

Age input is restricted to the range between 18 and 120.

This dual-layer validation ensures data integrity and enhances user experience by minimizing invalid data entries.

4.2 Backend Validation and Data Storage

After passing client-side validation, data is sent to the FastAPI backend, which performs additional verifcation to prevent duplicate entries based on email or phone number. Valid entries are assigned a unique UUID and timestamped before being stored in MongoDB. Attempts to submit duplicate emails or phone numbers return appropriate error messages, maintaining data consistency.

4.3 Display of Stored Data

The application includes a dedicated page displaying all validated and stored user records in a tabular format. The table includes columns for Name, Email, Phone, and Age. Only data that meets the validation criteria on both frontend and backend is shown, ensuring the reliability of displayed information.

4.4 User Feedback and Error Handling

Throughout the process, users receive clear messages indicating success or specifc validation errors, facilitating smooth interaction. Network errors or backend failures are also gracefully handled with informative alerts.

5. CONCLUSION

This project successfully demonstrates the design and implementation of a full-stack data validation system using modern web technologies. The application efectively enforces input validation at both the client and server levels, ensuring that only clean, accurate, and secure data is stored in the database. By leveraging React for an interactive frontend and FastAPI for a fast and reliable backend, the system achieves a seamless fow of data from form submission to database storage.

The use of tools such as React Hook Form, Pydantic, and MongoDB contributes to robust validation, fexible data handling, and an overall efcient architecture. The results confrm that proper validation mechanisms signifcantly reduce the risk of incorrect or malicious data entries, thus improving data quality and user experience. This project lays the foundation for future enhancements such as authentication, analytics, and advanced validation, making it a scalable and practical solution for real-world applications.

6. FUTURE SCOPE

The current system successfully implements basic data validation and secure storage for user inputs. However, there are several opportunities to enhance the project in future iterations to make it more robust, scalable, and feature-rich:

1. Advanced Validation Features

Implementing contextual validation such as domain-specifc email checking (e.g., blocking disposable email domains).

Adding international phone number validation using libraries like libphonenumber.

Real-time validation with suggestions or auto-correction (e.g., typo correction in emails).

2. Authentication and User Management

Integrating secure user login and registration systems using JWT-based authentication.

Role-based access control to restrict sensitive operations to authorized users.

Option to update or delete existing records through an admin dashboard.

3. Data Analytics and Visualization

Adding data dashboards using libraries like Chart.js or D3.js to show age distribution, submission trends, etc.

Export options to download stored data as CSV or Excel fles for external use.

4. Deployment and Scalability

Hosting the application on cloud platforms like AWS, Azure, or Vercel for public access.

Using Docker containers for easy deployment and environment management.

Scaling the backend using tools like Kubernetes or serverless functions for handling high trafc.

5. Security Enhancements

Implementing rate-limiting and CAPTCHA to prevent bot submissions.

Adding audit logs and encryption for sensitive felds like phone numbers or emails.

Enforcing HTTPS and securing API endpoints with tokens or OAuth.

6. Multi-language and Accessibility Support

Making the UI accessible to users with disabilities (WCAG compliance).

Supporting multilingual forms to reach a wider audience.

7. REFERENCES

[1] React Documentation. (n.d.). React – A JavaScript library for building user interfaces. Retrieved from https://reactjs.org/docs/getting-started.html

[2] React Hook Form. (n.d.). Performant, fexible and extensible forms with easy-to-use validation. Retrieved from https://react-hook-form.com/

[3] FastAPI. (n.d.). FastAPI framework, high performance, easy to learn, fast to code, ready for production. Retrieved from https://fastapi.tiangolo.com/

[4] MongoDB, Inc. (n.d.). MongoDB Manual. Retrieved from https://docs.mongodb.com/manual/

[5] Motor: The async Python driver for MongoDB. (n.d.). Retrieved from https://motor.readthedocs.io/en/stable/

[6] Axios. (n.d.). Promise based HTTP client for the browser and node.js. Retrieved from https://axios-http.com/

[7] Pydantic. (n.d.). Data validation and settings management using Python type annotations. Retrieved from https://pydantic.dev/

[8] Mozilla Developer Network (MDN). (n.d.). Form Validation. Retrievedfromhttps://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

[9] CORS in FastAPI. (n.d.). Cross-Origin Resource Sharing (CORS) with FastAPI. Retrieved from https://fastapi.tiangolo.com/tutorial/cors/

[10] Python-dotenv. (n.d.). Read key-value pairs from a .env fle and set them as environment variables. Retrieved from https://github.com/theskumar/python-dotenv