COLLEGECODE:9528

COLLEGE NAME: SCAD COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT COMPUTER SCIENCE ENGINEERING

STUDENT NM ID: F99679D4DE022AAAFB54C276C36A6B1C

Roll no : 952823104164

DATE : 26.09.2025

Completed the project named as:

Phase 1

TECHNOLOGYPROJECTNAME : **USER REGISTERATION AND VALIDATION**

SUBMITTED By,

NAME: N.SUJITHRA

MOBILE NO: 9944738661

Phase 2– Solution Design And Architecture

1.Tech Stack Selection

_For building the User Registration with Validation project, the following technology stack is chosen:

Frontend:
HTML, CSS, JavaScript → For creating user interface forms.
React.js (optional for advanced frontend) → To handle dynamic validation feedback (like showing errors instantly).

Backend:
Node.js with Express.js → To handle API requests (user registration, login, validation).

Database:
MongoDB → For storing user details securely.

Fields: username, email, password (hashed), phone number, created.

Validation Libraries:
Joi or Validator.js for backend validation.
Regex for email/password format checking.

Security:
bcrypt.js for password hashing.
JWT (JSON Web Token) for authentication after registration/login.

Reason for selection:
Node.js + Express provides lightweight, fast, and scalable API.
MongoDB suits unstructured data like user profiles.
Libraries simplify strong validation and security.

2. UI Structure/ API Schema Design

The User Interface (UI) is the entry point of the project where users provide their registration details. The registration page contains labeled input fields for username, email, password, confirm password, and phone number. Each field has real-time validation:

for example, the email field must follow a standard email format, the password must contain at least 8 characters including uppercase, lowercase, and digits, and the phone number must be exactly 10 digits. Error messages are displayed directly below each field so that users can correct mistakes before submission. This improves usability and reduces the chance of invalid requests reaching the server.

On the backend, an API Schema is defined to handle requests. The registration API accepts a POST request to /api/register, where the data is sent in JSON format. A sample request looks like:

```
{
  "username": "Sowetha",
  "email": "sowetha@example.com",
  "password": "StrongPass123",
  "phone": "9876543210"
}
```

3. Data Handling Approach

The data handling approach describes how user data moves through the system from input to storage. When a user fills out the registration form, frontend validation checks the input instantly using JavaScript or Regex. For example, it ensures the email format is correct, the password is strong, and the phone number has exactly 10 digits. If the data is invalid, an error message is displayed immediately, preventing the user from submitting until all fields are corrected.

Once the frontend validation is successful, the form data is sent to the backend via a POST request to the /register endpoint. On the backend, the request body is validated again using a Joi schema or server-side Regex patterns. This second layer of validation ensures that even if someone bypasses the frontend validation (for example, by modifying the request in tools like Postman), the server still protects the database from invalid data.

After validation, the password is processed using encrypt hashing to ensure security. This means the original password is never stored directly in the database. Instead, a secure hash is stored, which cannot be easily reversed. The data is then saved to the MongoDB database in a users collection. If the email is already registered, the backend sends an error response (409 Conflict) and does not save the data. If everything is valid, the backend responds with a success message. This approach ensures that all data passing through the system is validated, secured, and handled correctly at each stage.