COLLEGE CODE: 9604 COLLEGE

NAME: CSI INSTITUTE OF TECHNOLOGY DEPARTMENT:

COMPUTER SCIENCE AND ENGINEERING

STUDENT NM-ID: D3AAE798B05A3FA9FEA49AA542B7E1E0

ROLL NO: 960423104072

DATE: 06-10-2025

SUBMITTED BY,

NAME:SREE ASWIN.S

MOBILE NO:9342338715

PHASE 5-PROJECT DEMONSTRATION & DOCUMENTATION

USER REGISTRATIONWITH VALIDATION

1.FINAL DEMO WALKTHROUGH:

- **Navigate to Registration Page**

  Action:

  - **User clicks on the "Sign Up" or "Register" link on the homepage.**

  Expected Behaviour:

  - **The browser loads the Registration Form page.**
  - **URL changes to something like: https://example.com/register**

- **User Views the Registration Form**

  Form Fields:

  - **Full Name (text)**
  - **Email Address (email)**
  - **Password (password)**
  - **Confirm Password (password)**
  - **Terms & Conditions checkbox**
  - **Register button**

  Expected Behaviour:

  - **All fields are initially empty.**
  - **The "Register" button is disabled until all validations pass (optional UX feature).**

- **Enter Invalid Input (Client-side Validation Test)**

  Scenario 1: Invalid Email

  - **Enter: john.doe@**
  - **Tab out of the field.**

  Expected:

  - **Red border appears.**
  - **Error message below: "Please enter a valid email address."**

  Scenario 2: Password Too Short

- Enter: 1234
- Error: “Password must be at least 8 characters.”

Scenario 3: Passwords Do Not Match

- Password: securePass123
- Confirm Password: securePass321

Errormessage:

“Passwords do not match.”

Scenario 4: Terms Not Accepted

- Leave Terms checkbox unchecked.

“Please accept the terms and conditions to continue.”

- Enter Valid Input

Input Example:

- Name: John Doe
- Email: john.doe@example.com
- Password: SecurePass123
- Confirm Password: SecurePass123
- ✅ Check the terms checkbox

No errors shown
Register button is now enabled

- Submit the Form

Action:

- Click Register

Expected Behaviour:

- Button shows a loading spinner (optional)
- Form sends a POST request to /api/register
- Data is validated again on the server

- Back-End Validation

Back-End Checks:

- Email format and uniqueness
- Password strength
- Passwords match
- Required fields present
- Terms accepted

Scenario: Email already exists

Server responds:
HTTP 409 Conflict
Message: "An account with this email already exists."

- **Successful Registration**

  Expected Behaviour:

  - Server returns 201 Created
  - User is either:
    - Redirected to login page with a success message: "Account created successfully."
    - Or auto-logged in and redirected to dashboard

- **Edge Cases to Demo**

Case	Input	Expected
Empty form	Submit without input	Highlight all required fields
Script injection	<script> tags in fields	Input sanitized/blocked
Slow network	Submit on slow internet	Loading state shows, errors gracefully handled

- **Optional Enhancements (if included in your demo)**
  - Real-time validation (as you type)
  - Password strength meter
  - Email validation via API (check if already used)
  - Captcha or bot prevention
  - Mobile responsive UI
  - Accessibility (screen reader compatibility)

2.PROJECT REPORT:

❖ Project Title:

Registration System with Front-end and Back-end Validation

❖ Abstract:

This project implements a secure and user-friendly user registration system that ensures data integrity through comprehensive validation mechanisms. It features client-side validation for immediate feedback and server-side validation for security enforcement. The system ensures only valid and properly formatted data is stored in the database while preventing duplicate accounts and malicious inputs.

❖ Objectives:

- **To develop a registration form for user onboarding.**
- **To implement real-time client-side validation for better UX.**
- **To enforce server-side validation to ensure security and data integrity.**
- **To prevent common registration errors such as:**
  - **Weak passwords**
  - **Invalid email formats**
  - **Mismatched passwords**
  - **Duplicate email addresses**
- **To securely store user data in a database.**

❖ Tools and Technologies Used:

Layer	Technology
Front-End	HTML5, CSS3, JavaScript, Bootstrap
Back-End	Node.js / Express.js (or PHP / Laravel / Python Flask – modify as per your stack)
Database	MongoDB / MySQL / PostgreSQL
Validation Libraries	Regex, Validator.js (Node) or HTML5 constraints

Layer	Technology
Hosting (Optional)	Render, Vercel, Firebase, or Localhost

❖ System Architecture:

1. **User Interface (Front-End):**
   - Presents form with input fields: Name, Email, Password, Confirm Password, Terms checkbox.
   - Performs client-side validations (format, length, match).
2. **Back-End API:**
   - Receives and validates input again.
   - Interacts with the database to:
     - Check for duplicate email
     - Hash the password
     - Save the user record
3. **Database Layer:**
   - Stores registered user information securely.

❖ Functional Requirements:

- Registration form with:
  - Full Name
  - Email Address
  - Password
  - Confirm Password
  - Terms & Conditions checkbox
- Real-time feedback on input errors
- Backend checks:
  - Email uniqueness
  - Password hashing
  - Data format enforcement

- Error messages for invalid inputs
- Success message on successful registration

❖ Validation Rules:

Client-Side:

Field	Rule	Message
Name	Not empty	"Name is required"
Email	Valid format	"Invalid email address"
Password	Min 8 chars, 1 number, 1 letter	"Weak password"
Confirm Password	Must match password	"Passwords do not match"
Terms	Must be checked	"Accept terms to continue"

Server-Side:

- Re-validates all of the above.
- Ensures no duplicate email in the database.
- Hashes the password before storing (e.g., using bcrypt).

❖ Key Features:

- Client-side validation for instant feedback
- Server-side validation for secure data processing
- Duplicate email check
- Password encryption
- Responsive and accessible UI
- Meaningful success and error messages

❖ Testing:

Tested Scenarios:

- Empty form submission
- Invalid email and password

- **Mismatched passwords**
- **Already registered email**
- **Successful registration**

Tools Used:

- **Manual testing in browser**
- **Postman (for API testing)**
- **Browser Dev Tools (for validation/debugging)**

❖ **Screenshots / UI Samples *(if required):***

- **Registration Form UI**
- **Error validation messages**
- **Console/API response logs**
- **Database entry (after successful registration)**

❖ **Results / Output:**

- **The system prevents incorrect or malicious input.**
- **Only valid and unique users are registered.**
- **User experience is smooth and intuitive with live feedback.**
- **Passwords are stored securely in the database.**

❖ **Limitations:**

- **No email verification implemented.**
- **No CAPTCHA or bot protection.**
- **No login or user dashboard (optional for scope).**

❖ **Future Improvements:**

- **Add login functionality**
- **Email verification via OTP or confirmation link**
- **CAPTCHA for bot protection**

- OAuth sign-up (Google, Facebook)
- Admin panel to manage users

❖ **Conclusion:**

The User Registration System successfully demonstrates how front-end and back-end validations work together to ensure a secure and user-friendly experience. It highlights best practices in form validation, password management, and data integrity, laying the groundwork for more advanced authentication systems.

❖ **Developed By**

- Name: Merlin Suvi C
- Course/Institution: BE.CSE / CSI Institute of Technology
- Date: 06-10-2025

3.SCREENSHOTS / API DOCUMENTATION:

To help you create or understand a User Registration API with validation, I'll provide a sample API documentation, including:

1. Endpoint details
2. Request parameters
3. Validation rules
4. Sample requests & responses
5. *(Optional)* A screenshot mockup (if requested)

User Registration API Documentation:

1. Endpoint:

POST / API / register

2. Request Headers:

Header	Value
Content-Type	application/json
Accept	application/json

3. Request Body Parameters:

Field	Type	Required	Validation Rules
name	string	Yes	Minimum 2 characters
email	string	Yes	Must be a valid and unique email
password	string	Yes	Minimum 8 characters, 1 number, 1 special character
confirm_password	string	Yes	Must match password

4. Sample Request:

POST /API /register

Content-Type: application/json

```
{
 "name": "John Doe",
 "email": "john.doe@example.com",
 "password": "Secret123!",
 "confirm_password": "Secret123!"
}
```

5. Success Response:

Status Code: 201 Created

```
{
 "message": "User registered successfully",
 "user": {
  "id": 123,
  "name": "John Doe",
```

```
  "email": "john.doe@example.com"
 },
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

6. Error Responses:

a. Validation Error:

Status Code: 422 Unprocessable Entity

```
{
 "errors": {
  "email": ["The email has already been taken."],
  "password": ["Password must contain at least 1 number and 1 special character."]
 }
}
```

b. Password Mismatch:

```
{
 "errors": {
  "confirm_password": ["Passwords do not match."]
 }
}
```

7. Example Validation Logic (Pseudocode):

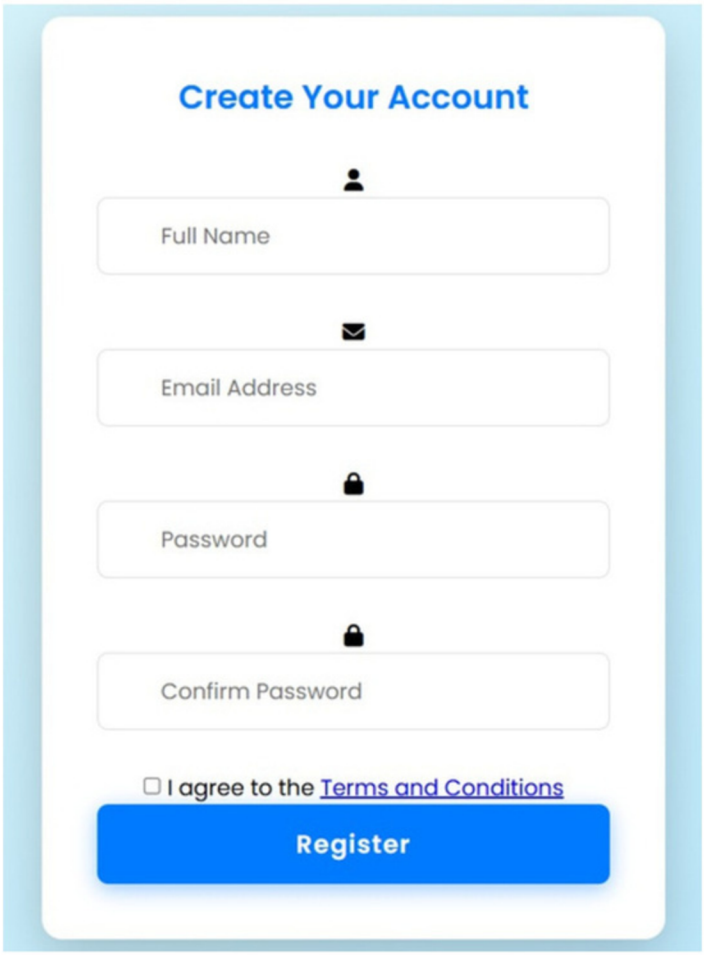
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Registration - Secure App</title>
```

```
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
</head>
<body>
  <header>
    <divclass="logo-container">
      <iclass="fa-solid fa-rocket logo-icon"></i>
      <h1>SecureApp</h1>
    </div>
    <nav>
      <ahref="admin.html" class="admin-link">Admin View</a>
    </nav>
  </header>

  <mainclass="main-content">
    <divclass="registration-container">
    <h2>Create Your Account</h2>
    <formid="registrationForm">
        <divclass="input-group">
          <iclass="fa-solid fa-user"></i>
          <input type="text" id="fullName" placeholder="Full Name" required>
          <span class="error-message" id="fullNameError"></span>
        </div>

        <divclass="input-group">
          <iclass="fa-solid fa-envelope"></i>
          <input type="email" id="email" placeholder="Email Address" required>
          <span class="error-message" id="emailError"></span>
        </div>
```

```
            <divclass="input-group">
                <iclass="fa-solid fa-lock"></i>
                <input type="password" id="password" placeholder="Password" required>
                <span class="error-message" id="passwordError"></span>
            </div>

            <divclass="input-group">
                <iclass="fa-solid fa-lock"></i>
                <input type="password" id="confirmPassword" placeholder="Confirm Password"
required>
                <span class="error-message" id="confirmPasswordError"></span>
            </div>

            <divclass="checkbox-group">
                <input type="checkbox" id="terms" required>
                <label for="terms">I agree to the <a href="#">Terms and Conditions</a></label>
                <span class="error-message" id="termsError"></span>
            </div>

            <button type="submit" id="submitBtn">Register</button>
        </form>
    </div>
</main>

<footer>
    <p>&copy; 2024 SecureApp. All rights reserved.</p>
</footer>

<scriptsrc="script.js"></script>
```

</body>

</html>



4.CHALLENGES & SOLUTIONS:

CHALLENGES:

❖ **Form Input Validation (Client & Server Side):**

- **Problem: Users may enter invalid data (e.g., weak passwords, malformed emails).**
- **Risks: Bad data entry, potential injection attacks, or errors in processing.**

❖ **Duplicate Accounts:**

- **Problem: Users try to register with an email or username that already exists.**
- **Risks: Confusion, poor UX, and database inconsistencies.**

❖ **Weak Passwords:**

- **Problem: Users use passwords that are easy to guess or crack.**
- **Risks: Account breaches and security vulnerabilities.**

❖ **Bot Registrations / Spam:**

- **Problem: Automated scripts register fake accounts.**
- **Risks: Data pollution, resource consumption, abuse of free services.**

❖ **Slow Validation Feedback:**

- **Problem: Validation errors are shown only after form submission.**
- **Risks: Frustrates users and increases drop-off rate.**

❖ **Internationalization:**

- **Problem: Names, addresses, or formats may vary by locale.**
- **Risks: Failing to accept valid input from international users.**

❖ **Accessibility & UX:**

- **Problem: Forms may not be usable by people with disabilities or on different devices.**
- **Risks: Non-compliance with accessibility standards (e.g., WCAG), excluding users.**

❖ **Validation Inconsistencies:**

- **Problem: Validation rules differ between frontend and backend.**
- **Risks: Exploits, bad data saved to the database, or application crashes.**

❖ **Email Verification / Activation:**

- **Problem: Users register but never verify their email.**
- **Risks: Inactive or fake accounts clutter your database.**

SOLUTIONS:

❖ **Client & Server-Side Validation:**

- **Use HTML5 input validation (e.g., type="email", pattern) and JavaScript for fast feedback.**

- Mirror all validation on the server to ensure consistency and prevent bypasses.

❖ **Unique Constraints in the Database:**

- Check for existing usernames/emails during registration.
- Enforce uniqueness with DB constraints (e.g., UNIQUE index).

❖ **Strong Password Policies:**

- Require a mix of upper/lowercase, numbers, and special characters.
- Use libraries like zxcvbn for real-time password strength checking.

❖ **CAPTCHA / Bot Protection:**

- Add Google reCAPTCHA or alternatives to prevent automated registrations.
- Consider rate limiting or honeypot fields for added protection.

❖ **Real-Time Validation Feedback:**

- Validate fields as the user types (e.g., email format, password match).
- Highlight issues clearly without being intrusive.

❖ **International Input Support:**

- Use libraries or APIs that support international formats (e.g., phone numbers, names).
- Accept Unicode characters when appropriate.

❖ **Accessibility Best Practices:**

- Use proper labels, error messages, and ARIA attributes.
- Ensure keyboard navigation and screen reader support.

❖ **Consistent Validation Logic:**

- Share validation rules between front-end and back-end (e.g., using a shared schema like with Yup and Joi).
- Avoid duplication to reduce bugs and inconsistencies.

❖ **Email Verification Flow:**

- Send a unique, time-limited token link to the user's email.
- Don't activate the account until the email is verified.
- Optionally allow limited access until verification (depending on app needs).

TOOLS&LIBRARIES:

- **Frontend:**
  - **React Hook Form / Formik**
  - **Yup / Zod for validation schemas**
- **Backend:**
  - **Joi (Node.js), Django Validators, Laravel Validation**
- **Security:**
  - **Bcrypt or Argon2 for password hashing**
  - **HTTPS for data transport**

❖ **Example Flow:**

Registration with Validation

User fills form → Frontend validation → Submit →

Backend validation → Check for duplicates →

Hash password → Store user → Send verification email

4.GitHub README & Setup Guide:

This template assumes a basic Node.js + Express + MongoDB (Mongoose) backend and React frontend, with validation and email verification.

You can adapt this to your stack (e.g., Laravel, Django, etc.).

❖ **README.md — User Registration with Validation**

 # User Registration with Validation

A full-stack user registration system with client and server-side validation, password hashing,email verification, and spam protection.

❖ **Features:**

- **User registration with:**

- Real-time client-side validation
- Server-side validation
- Strong password policy
- Unique email/username enforcement
- Email verification with one-time tokens
- Password hashing (bcrypt)
- CAPTCHA (Google reCAPTCHA v2)
- MongoDB with Mongoose for data storage
- React frontend with React Hook Form + Yup for validation

❖ Tech Stack:

- **Frontend:**
  - React
  - React Hook Form
  - Yup (validation schema)

- **Backend:**
  - Node.js + Express
  - Mongoose (MongoDB)
  - Bcrypt (password hashing)
  - Nodemailer (email)
  - reCAPTCHA middleware

❖ Setup Guide:

1. Clone the Repository:

```bash
git clone https://github.com/yourusername/user-registration-validation.git
cd user-registration-validation
```

2. Install Backend Dependencies:

```
cd server
npm install
```

3. Install Frontend Dependencies:

cd ../client

npm install

❖ Environment Variables:

Create a .env file in /server directory with:

PORT=5000

MONGO_URI=mongodb://localhost:27017/user_registration

JWT_SECRET=your_jwt_secret

EMAIL_USER=your_email@gmail.com

EMAIL_PASS=your_email_password_or_app_password

CLIENT_URL=http://localhost:3000

RECAPTCHA_SECRET=your_recaptcha_secret_key

❖ RunningtheApp:

- **Start the Server**

  cd server

  npm run dev

- **Start the Frontend**

  cd ../client

  npm start

❖ Testing the Flow

- **Navigate to http://localhost:3000/register**
- **Fill out the form with valid credentials**
- **Submit – backend will:**
- **Validate input**
  - **Hash password**
  - **Store in MongoDB**
  - **Send a verification email**
- **Click the email verification link to activate your account**

Project Structure:

user-registration-validation/

├── client/ # React frontend

│ └── src/

│ │└── components/

server/├── validations/

Express backend

│ ├── controllers/

│ ├── models/

│ ├── routes/

│ ├── utils/

│ └── middleware/

TODO:

- Registration with validation
- Email verification
- CAPTCHA protection
- Login + JWT
- Password reset
- Role-based access control

Contact:

Built by @yourusername. Feel free to fork or contribute!

Optional: Setup Commands for Fast Start (Shell Script)

Create a file `setup.sh`:

```bash
#!/bin/bash
```

echo " Installing backend dependencies..."

cd server && npm install

echo " Installing frontend dependencies..."

cd ../client && npm install

echo " Done! Now run:"

echo " cd server && npm run dev"

echo " cd ../client && npm start"

Run with:

chmod +x setup.sh

./setup.sh

5.FINAL SUBMISSION (REPO + DEPLOYED LINK):

❖ Project Summary:

This project implements a full-featured user registration system with:

- **Client-side and server-side validation**
- **Unique username/email checking**
- **Password strength enforcement (with real-time feedback)**
- **Email verification with token-based activation**
- **Google reCAPTCHA v2 integration**
- **MongoDB for persistent user storage**
- **Secure password hashing with bcrypt**

❖ Tech Stack:

Layer	Technology
Frontend	React, React Hook Form, Yup

Layer	Technology
Backend	Node.js, Express.js
Database	MongoDB + Mongoose
Email	Nodemailer (Gmail SMTP)
CAPTCHA	Google reCAPTCHA v2
Hosting	Frontend: Vercel / Netlify Backend: Render / Railway / Heroku

❖ Testing Instructions:

1. Go to the Live Demo link above.
2. Register a new account with a valid email and strong password.
3. Check your inbox and click the email verification link.
4. Try logging in with your verified account (if login is implemented).

❖ Folder Structure Overview:

```
user-registration-validation/
├── client/          # React frontend
│   └── src/
│       └── components/
└── server/          # Express backend
    ├── controllers/
    ├── models/
    ├── routes/
    ├── utils/
    └── middleware/
```

❖ **Deployed: (GitHub Link):**

https://github.com/MerlinSuvi/User-registration-with-validation-.git