COLLEGE CODE : 9604

COLLEGE NAME : CSI INSTITUTE OF TECHONOLGY

DEPARTMENT : AI & DS

STUDENT NM- ID : 332E259F6622205A5184DD2072774F0D

ROLL NO : 960423243006

DATE : 22/09/2025

SUBMITTED BY,

NAME : ASHICK JOSHUA C
MOBILE NO: 9487944311

Phase 2 - USER REGISTRATION WITH VALIDATION

Tech Stack Selection:

For this project, we will use the MERN stack, which is modern, efficient, and scalable.

- **Frontend (Client-Side):** React.js
  A popular JavaScript library for building dynamic and responsive user interfaces (UIs). It provides a seamless user experience with real-time validation feedback.

- **Backend (Server-Side):** Node.js with Express.js
  Node.js is a JavaScript runtime that allows running JavaScript on the server. Express.js is a lightweight framework that simplifies the process of building robust APIs.

- **Database:** MongoDB
  A NoSQL database that stores data in flexible, JSON-like documents. It integrates well with Node.js and is ideal for managing user profiles.

- **Validation Libraries:**

- Frontend: React Hook Form for managing form state and client-side validation.

- Backend: Zod or Joi for defining schemas and performing rigorous server-side validation.

- **Password Hashing**: bcrypt.js

Used to hash and salt user passwords before storing them in the database for security.

UI Structure / API Schema Design:

UI Structure

The registration form will include:

Heading: "Create Your Account"

Input Fields:

Full Name (text)

Email Address (email)

Password (password)

Confirm Password (password)

Error Message Placeholders: For displaying validation errors (e.g., "Email is invalid," "Passwords do not match").

Submit Button: Labeled "Register."

Feedback Area: Displays success messages or general form errors.

API Schema Design

Endpoint: POST /api/auth/register

Description: Creates a new user in the database.

Request Body (Client → Server):

```
{
  "fullName": "John Doe",
  "email": "john.doe@example.com",
  "password": "strongPassword123"
}
```

Success Response (Server → Client):

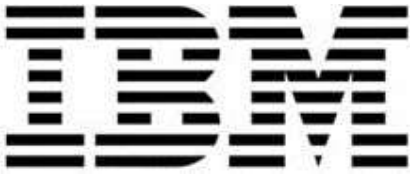
Status Code: 201 Created

Body:

```
{
  "message": "User registered successfully!",
  "userId": "60c72b2f9b1d8c001f8e4c9a"
}
```

Error Response (Server → Client):

Status Code: 400 Bad Request (validation errors) or 409 Conflict (email already exists).



Body:

```
{
  "error": "Email already exists."
}
```

Data Handling Approach:

Client-Side (Frontend)

1. Data Capture: The React form captures user input in state as the user types.

2. Live Validation: Runs checks while the user types (e.g., email format, password matching).

3. Submission: On clicking Register, final validation runs. If valid, data is sent to the backend using a POST request.

4. Response Handling:

On success → Display "Registration complete!" and redirect to login.

On error → Display the error message returned from the server.

Server-Side (Backend)

1. Receive Data: The Express.js server receives JSON data.

2. Security Validation: Re-validates all data, checking:

Correct types (email is a string, password length ≥ 8).

Business rules.
Email uniqueness in MongoDB.

3. Password Hashing: Hash password using bcrypt.

4. Database Storage: Save the new user to MongoDB.

5. Response: Send back a success or error message to the client.

Component / Module Diagram:

Frontend (React App)

RegistrationPage.js: Main page holding the registration form.

RegistrationForm.js: Contains form logic, input fields, and submission handler. Communicates with ApiService.

Input.js: Reusable component for form fields.

ApiService.js: Handles HTTP requests to the backend API.

Backend (Node.js/Express App)

authRoutes.js: Defines API routes (e.g., /api/auth/register) and maps them to controllers.

authController.js: Main registration logic (validation, hashing, database interaction).

User.js (Model): MongoDB schema for a user.

validation.js (Middleware): Validates incoming data before it reaches the controller.

Basic flow Diagram: