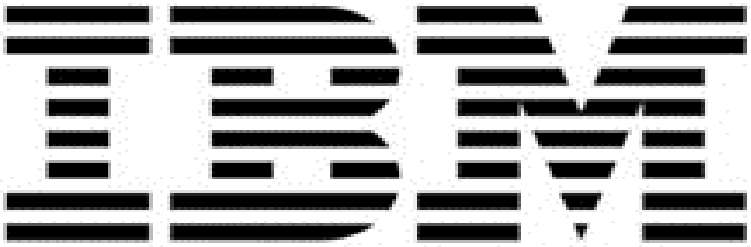


COLLEGE CODE COLLEGE NAME DEPARTMENT STUDENT NM-ID

ROLL NO : 953023104001

DATE : 15/09/2025

NM-ID : FA34C28B947844DA3DA41C937B77D2DE

Completed the project Phase : 2

Project: USERS REGISTRATION WITH VALIDATION

SUBMITTED BY,

NAME : AARICK DARREN

MOBILE NO : 9176644845

1. Tech Stack Selection

To build a **secure and scalable user registration module**, the following tech stack is selected:

- **Frontend**:
  - **React.js** → For creating a dynamic, interactive registration form.
  - **HTML5, CSS3, JavaScript** → For basic structure, styling, and client-side validation.
  - **Bootstrap / TailwindCSS** → To design responsive and modern UI quickly.
- **Backend (API Layer)**:
  - **Node.js** → Non-blocking event-driven environment, ideal for handling multiple requests.
  - **Express.js** → Lightweight framework for building REST APIs with ease.
- **Database**:
  - **MongoDB** (NoSQL) → Flexible schema for storing user details.
  - Alternatively, **MySQL/PostgreSQL** can be used for relational data consistency.
- **Security**:
  - **bcrypt.js** → For hashing passwords securely.
  - **Joi / Validator.js** → For validating email, password, and input data.
  - **Helmet.js & CORS** → For API security.
- **Development Tools**:
  - **Postman** → API testing.
  - **GitHub/Git** → Version control.
  - **VS Code** → Development IDE.

---

2. UI Structure / API Schema Design

UI Structure (Frontend)

The registration form includes:

- **Input Fields**:
  - Full Name
  - Email
  - Password
  - Confirm Password

- **Validation Messages**:
    - "Email is invalid"
    - "Password must be at least 8 characters with uppercase, lowercase, number, and symbol."
    - "Passwords do not match."
- **Submit Button**

API Schema Design (Backend)

- **Endpoint: POST /api/register**
    - **Request Body:**

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "Strong@123",
  "confirmPassword": "Strong@123"
}
```

Response (Success):

```
{
 "message": "User registered successfully",
 "status": 201
}
```

Response (Error – Invalid Email):

```
{
    "error": "Invalid email format",
    "status": 400
}
```

3. Data Handling Approach

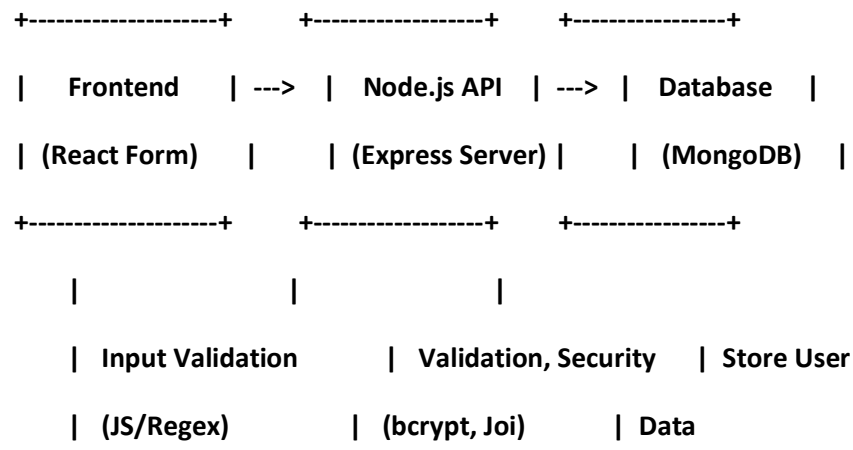
The data handling strategy ensures accuracy, security, and performance:

1. **Frontend Validation**
    - **Check required fields.**
    - **Validate email format using regex.**

- Ensure password length ≥ 8 with special rules.
- Confirm password matches.

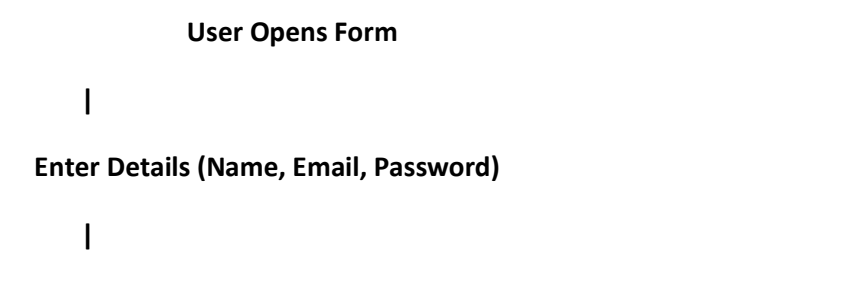
2. Backend Validation (Server-Side)
   - Verify email uniqueness by querying the database.
   - Hash password before storing using bcrypt.
   - Use middleware for validating requests (e.g., Joi schema).
3. Database Storage
   - Store only necessary fields (id, name, email, hashedPassword, created_at).
   - Never store plain-text passwords.
4. Error Handling
   - Return meaningful error messages.
   - Use standard HTTP status codes (400 Bad Request, 201 Created, 409 Conflict).

4. Component / Module Diagram



- Frontend Module → Handles input and basic validation.
- Backend Module → Handles business logic and validation.
- Database Module → Stores secure user information.

5. Basic Flow Diagram



Frontend Validation

(Empty fields? Invalid email? Weak password?)

|

Passes? ---------- No ----------> Show error message

| Yes

|

Send Data to API (POST /api/register)

|

Backend Validation

(Duplicate Email? Password Hashing?)

|

Passes? ---------- No ----------> Return error JSON

| Yes

|

Save User in Database

|

Return Success Response

Conclusion

The solution design for the User Registration with Validation module ensures a balance between usability, security, and scalability. By selecting a modern tech stack (React.js, Node.js, Express, and MongoDB/MySQL), the system is capable of handling user input efficiently while maintaining flexibility for future growth.

The UI structure focuses on simplicity and clarity, providing users with a straightforward registration experience and instant feedback for invalid inputs. The API schema is designed to follow REST principles, ensuring smooth communication between the frontend and backend.

A clear data handling approach has been defined, ensuring that validation is performed both on the client side (for user convenience) and on the server side (for security and reliability). Passwords are secured using hashing techniques, and error handling ensures meaningful messages are returned to users.

The component/module diagram and basic flow diagram clearly outline how different layers—frontend, backend, and database—interact with each other. This modular approach simplifies development, testing, and future upgrades.