COLLEGE CODE:9111

COLLEGE NAME : SRM Madurai College For Engineering and Technology

DEPARTMENT : B.Tech Information Technology

NAME	STUDENT NM-ID
Irudhaya Albert	16C4819348EEF23BC7949300C8AACEBC
Nishanthan S	6FD89376531AD80AC6F4FF5782127907
Thameej Ahamed	ACB9D50E54865509DF44C356F445FCFD
Pradeep S	1636178DC69762CAF364A6BC4F0A1684
Vikram KV	E6E2E5AB586408AB3A741885965448AF

DATE:15-09-2025

Completed the project named as

Phase _2_SOLUTION DESIGN & ARCHITECTURE PROJECT

NAME : PORTFOLIO WEBSITE

SUBMITTED BY,

Irudhaya Albert
Nishanthan S
Thameej Ahamed
Pradeep S
Vikram KV

1. Tech Stack Selection

The **choice of technology stack** plays a vital role in ensuring the success of any software project. For our **Portfolio Website**, the stack was carefully selected to balance **scalability, performance, developer productivity, and industry relevance**.

Frontend (Client-Side)

- **React.js** (preferred) or **HTML, CSS, JavaScript**:
  - React provides a **component-based architecture**, making it easier to maintain reusable UI elements such as project cards, skill progress bars, and navigation bars.
  - If React is not used, HTML/CSS/JS ensures simplicity and fast prototyping.
- **Bootstrap / TailwindCSS:** For mobile-first responsive design, reducing the time required for UI styling.
- **Axios / Fetch API:** For communication with backend REST APIs.

Backend (Server-Side)

- **Node.js with Express.js Framework:**
  - Provides an **event-driven, non-blocking I/O model**, ideal for scalable web applications.
  - Express.js simplifies routing, middleware, and API design.
- **Nodemon (development tool):** Enables auto-restarting of server during code updates.

Database

- **MongoDB (NoSQL):**
  - Chosen for its **flexibility** in handling unstructured or semi-structured data.
  - Stores projects, skills, and recruiter messages in collections.
  - No need for strict schema (easy to update content as portfolio evolves).

Other Tools and Services

- **Postman:** For API testing and validation.
- **Git & GitHub:** For version control and collaborative development.
- **Netlify / Vercel:** For hosting frontend (React apps).
- **Heroku / Render / Railway:** For deploying Node.js backend.
- **JWT (JSON Web Tokens):** Optional for authentication if secure admin login is added.
- **Cloudinary / Firebase Storage:** Optional for hosting images such as profile picture or project thumbnails.

Justification for Stack:
This stack ensures that the project remains **lightweight, easy to deploy, scalable, and industry-aligned**. Many recruiters and companies already use React + Node.js + MongoDB (MERN stack), so the portfolio itself demonstrates modern web skills.

2. UI Structure / API Schema Design

UI Structure (Frontend Pages)

The portfolio website is divided into logical sections that provide clarity and intuitive navigation.

1. **Home Page**
   - Profile image, student's name, short tagline (e.g., *"IT Student | Web Developer | Cloud Enthusiast"*).
   - Navigation bar for quick access to About, Projects, and Contact pages.
   - Optional banner or background animation to enhance visual appeal.
2. **About Page**
   - Detailed introduction.
   - Education details displayed in timeline format.
   - Achievements (certifications, awards).
   - Technical skills displayed in progress bars/cards (e.g., HTML 90%, Python 80%).
3. **Projects Page**
   - Projects displayed as cards with:
     - Title, description, technologies used.
     - GitHub/demo links.
     - Thumbnail image (optional).
   - Option to expand project details on click.
4. **Contact Page**
   - Contact form (Name, Email, Message).
   - Form validation before submission.
   - Data stored in backend database for future access.

API Schema Design (Backend + Database)

Projects Collection

```
{
  "id": "P001",
  "title": "Smart Attendance System",
  "description": "Face recognition-based system to automate attendance.",
  "technologies": ["Python", "OpenCV", "Flask"],
  "githubLink": "https://github.com/example",
  "demoLink": "https://example.com/demo",
  "dateAdded": "2025-09-15"
}
```

Skills Collection

```
{
  "id": "S001",
  "skillName": "JavaScript",
  "proficiency": "Intermediate",
  "category": "Programming Language"
}
```

Contact Collection

```
{
  "id": "C001",
```

```
  "name": "John Recruiter",
  "email": "john@example.com",
  "message": "Interested in your profile for a frontend developer role.",
  "date": "2025-09-15"
}
```

This schema ensures data is stored in a structured yet flexible way, suitable for quick retrieval and updates.

3. Data Handling Approach

The Portfolio Website will follow a **client-server architecture** with the following flow:

1. **Frontend (React/HTML):**
    - Displays user profile, skills, projects, and contact form.
    - Sends API requests (GET, POST, PUT, DELETE) to backend when dynamic content is required.
2. **Backend (Node.js + Express):**
    - Handles incoming requests from the frontend.
    - Validates data before storing it in MongoDB.
    - Returns JSON responses to frontend for display.
3. **Database (MongoDB):**
    - Stores all content including project details, skills, and recruiter contact submissions.
    - Provides dynamic updates without requiring changes to frontend code.

Data Handling Principles:

- **Validation:** All contact form inputs validated to avoid spam or SQL injection.
- **Error Handling:** Backend includes try/catch blocks and meaningful error messages.
- **Security:**
    - Use of HTTPS during deployment.
    - Sanitization of inputs before storing in MongoDB.
    - Rate-limiting to prevent API abuse (optional).
- **Performance:**
    - Indexing frequently queried fields in MongoDB (like project titles).
    - Optimizing API responses to reduce latency.

4. Component / Module Diagram

The system can be divided into three **main modules**:

1. **Frontend Module**
    - Home Component
    - About Component
    - Projects Component
    - Contact Component
2. **Backend Module**
    - API Router (handles endpoints `/api/projects`, `/api/skills`, `/api/contact`)

    - Controllers (logic for project, skill, and contact management)
    - Middleware (validation, error handling, logging)
    - Database Connector (MongoDB integration using Mongoose).
3. **Database Module**
    - **Projects Collection** (stores project details).
    - **Skills Collection** (stores skills and proficiency).
    - **Contact Collection** (stores recruiter queries).

This modular design ensures scalability, maintainability, and easy debugging.

5. Basic Flow Diagram

The **basic flow of the system** is as follows:

1. A recruiter/visitor opens the portfolio website via a browser.
2. The **frontend (React/HTML)** requests data (e.g., projects, skills) from the backend API.
3. The **backend (Node.js + Express)** receives the request, validates it, and fetches the required data from MongoDB.
4. MongoDB returns the data in JSON format to the backend.
5. The backend sends the response back to the frontend.
6. The frontend renders the data dynamically (project cards, skills list, etc.).
7. If a recruiter submits the contact form, the data flows:
    - Frontend → Backend → Database (stored in Contact Collection).

This flow ensures **real-time dynamic updates** and **smooth interaction** between user and system.

Additional Considerations

- **Scalability:** The architecture supports adding new modules (e.g., blog section, analytics) without major redesign.
- **Extensibility:** APIs are designed with REST principles, making future integration with mobile apps or external services possible.
- **Maintainability:** Code is divided into reusable components and modules for long-term use.
- **Industry Alignment:** The chosen MERN-like stack (React + Node.js + MongoDB) is highly demanded in the job market, so even the act of building this portfolio serves as skill demonstration.

Import CSV flow

Client uploaded a CSV file