COLLEGE CODE	**:**	8203
COLLEGE NAME	**:**	A V C COLLEGE OF ENGINEERING
DEPARTMENT	**:**	CSE
STUDENT NM-ID	**:**	8D414D5EBBA7EBAB4CF46FAFCA22E896
ROLL NO	**:**	820323104087
DATE	**:**	26.09.2025
COMPLETED THE PROJECT NAMED AS	**:**	PHASE 2 TECHNOLOGY
PROJECT NAME	**:**	FEEDBACK COLLECTION SYSTEM

SUBMITTED BY,
SAHANA K

MOBILE NO :7810095672

1. Tech Stack Selection
The project will use a **modern MERN-style stack (MongoDB, Express, Node.js)** but in a simplified form for Phase 2.

- **Frontend (User Interface):**
    - A lightweight HTML, CSS, and JavaScript-based feedback form for Phase 2.
    - Users can enter their **name, email, and feedback message**.
    - Responsive design ensures it works on both desktop and mobile devices.
    - Future enhancement: A React-based frontend to provide an admin dashboard with real-time updates.
- **Backend (API Layer):**
    - **Node.js** as the runtime environment for fast, non-blocking I/O.
    - **Express.js** as the web framework to define routes, middleware, and handle requests/responses efficiently.
    - RESTful API design ensures scalability and interoperability with future systems (e.g., mobile app).
- **Database (Data Storage):**
    - **MongoDB** chosen for its flexibility and JSON-like document structure.
    - Feedback entries often vary in length and content, making NoSQL a better fit compared to relational databases.
    - MongoDB provides indexing, filtering, and efficient querying for admin analytics.
- **Enhancements & Supporting Tools:**
    - **Mongoose**: Schema validation, middleware hooks, and easier data modeling.
    - **Nodemailer**: To notify admins via email when new feedback is received.
    - **Sentiment Analysis Library** (like sentiment or natural in Node.js): To categorize user feedback as positive, neutral, or negative.
    - **Postman**: For API testing and documentation.

Why this stack?
It is lightweight, scalable, easy to extend in future phases, and widely used in real-world feedback systems.

---

2. UI Structure / API Schema Design
This section defines how **users interact with the system (UI)** and how data is exchanged between **frontend and backend (API schema)**.

- **UI Structure:**
    - **User Interface (Feedback Form):**
        - Fields: *Name*, *Email*, *Message*.
        - Simple validation (non-empty, email format).
        - On submit, the form sends a JSON payload to the backend.
        - Displays confirmation to the user once feedback is saved.
    - **Admin Interface (Phase 2 – minimal):**
        - Admin uses an API endpoint with filters (via Postman or browser).
        - Response includes a structured list of feedback, sorted by date.
        - Future: Interactive dashboard with charts and filters.

- **API Schema Design:**
  - **POST /api/feedback**

    Request body:
    ```
    {
      "name": "Alice",
      "email": "alice@example.com",
      "message": "Loved the service!"
    }
    ```
    Response body:
    ```
    {
      "success": true,
      "data": {

        "name": "Alice",
        "email": "alice@example.com",
        "message": "Loved the service!",
        "sentiment": "positive",
        "createdAt": "2025-09-26T09:30:00Z"
      }
    }
    ```
  - **GET /api/admin?sentiment=positive&email=alice@example.com**

    Request parameters: Sentiment, Email, Date (optional).

    Response body:
    ```
    {
      "success": true,
      "data": [
        {
          "name": "Alice",
          "email": "alice@example.com",
          "message": "Loved the service!",
          "sentiment": "positive",
          "createdAt": "2025-09-26T09:30:00Z"
        }
      ]
    }
    ```
- **Validation Rules:**
  - Name: Minimum 2 characters.
  - Email: Must match email regex.
  - Message: Minimum 5 characters.
  - Sentiment: Auto-generated based on message.

---

3. Data Handling Approach

The system ensures **reliable, secure, and structured data handling** for both users and admins.

- **For Users (Feedback Submission):**
    - The form captures input → Sends JSON payload via POST → API validates and sanitizes → Data stored in MongoDB.
    - Automatic **sentiment analysis** categorizes the message.
    - Timestamp is added using server time to maintain consistency.
    - Optional: Admin notified via email.
- **For Admins (Feedback Viewing):**
    - Admin queries the system using GET /api/admin.
    - The API checks for filters (e.g., sentiment=negative).
    - MongoDB retrieves matching records and sorts them by newest first.
    - Data is returned as JSON for easy consumption.
- **Error & Exception Handling:**
    - Invalid input → Returns 400 Bad Request.
    - Database errors → Returns 500 Internal Server Error.
    - Custom error messages ensure better debugging.
- **Security Considerations:**
    - Input sanitization to prevent **NoSQL injection**.
    - Email addresses validated with regex.
    - Future enhancement: **Authentication middleware** for admin routes.
- **Scalability & Future Proofing:**
    - Feedback collection scales horizontally with MongoDB's sharding.
    - APIs are stateless, so they can be containerized with Docker for deployment.

---

4. Component / Module Diagram

Breaking the system into smaller modules ensures **separation of concerns** and maintainability.

- **Frontend Modules:**
    - **Feedback Form Module:** Collects user input and sends data to backend.
    - **Admin Interface Module (basic):** Displays feedback results (currently via API response).
- **Backend Modules:**
    - **Feedback Controller:**
        - Handles new submissions.
        - Validates and sanitizes input.
        - Calls sentiment analyzer.
        - Saves entry into MongoDB.
    - **Admin Controller:**
        - Fetches data from MongoDB.
        - Applies filters (sentiment, email, date).
        - Returns results as JSON.
    - **Utility Modules:**
        - sentiment.js: Performs sentiment classification.
        - emailService.js: Sends notifications to admin when new feedback arrives.

- **Database Layer:**
  - **MongoDB Feedback Collection** with schema: name, email, message, sentiment, createdAt.

High-level Module Flow:

```
[User Form] → [API Routes] → [Feedback Controller] → [MongoDB]
                          ↓
                 [Sentiment Utility]
                 [Email Notification]

[Admin API] → [Admin Controller] → [MongoDB] → [Filtered Results]
```

---

5. Basic Flow Diagram

This represents **end-to-end system flow** for both user and admin.

```
   ┌─────────┐
   │  User UI  │
   └────┬────┘
        │
 (1) Submit Feedback Form
        │
   ┌────▼────┐
   │  API      │
   │ (Express)   │
   └────┬────┘
        │
 (2) Validate Input Schema
        │
 (3) Run Sentiment Analysis
        │
 (4) Store in MongoDB
        │
 (5) Send Email Notification (Optional)
        │
   ┌────▼────┐
   │  Database  │
   │ (MongoDB)  │
   └────┬────┘
        │
 (6) Admin Requests Feedback
        │
 ┌───────▼───────┐
 │  Admin API (/admin) │
 └───────┬───────┘
        │
```

```
(7) Apply Filters (Sentiment, Date, Email)
        |
(8) Return JSON Results to Admin
```