### **1. Introduction**

**Project Title**: MERN Stack Survey Form  
 **Team Members**:

* Hiral Kothari
* Shubhi Shrivastava
* Jahnvi Saxena
* Akshadha Rathore

### **2. Project Overview**

#### **Purpose:**

The primary objective of this project is to build a robust, scalable, and user-friendly full-stack web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. This survey form application enables users to fill out and submit structured surveys, while also providing administrators with the ability to create, manage, and analyze these submissions. It bridges the gap between data collection and analysis, making it easier for organizations to gather feedback or insights.

By digitizing the survey process, the system reduces manual effort, increases data accuracy, and offers a centralized platform for managing feedback. The application is designed to be simple for end-users while being powerful and flexible for administrator.

#### **Goals:**

* **Collect Structured Feedback**: Allow users to respond to surveys in a standardized format for easy data interpretation.
* **Ensure Data Security and Integrity**: Store survey responses in a secure MongoDB database, ensuring confidentiality and data integrity.
* **Provide a Smooth User Experience**: Deliver a responsive and intuitive UI to make form filling seamless across devices.
* **Enable Survey Management**: Provide backend capabilities for administrators to view, filter, and manage survey responses.
* **Scalable Architecture**: Ensure the platform is scalable to support future growth, such as handling more complex surveys or larger user bases.

#### **Key Features:**

* **Fully Responsive Design**: The UI is optimized for desktops, tablets, and mobile devices using responsive web design principles.
* **Dynamic Form Handling**: React is used to dynamically update and validate form fields in real-time.
* **Admin Dashboard**: View submitted survey responses in a tabular format for analysis and decision-making.
* **Form Validation**: Built-in client- and server-side validations ensure that all required fields are completed before submission.
* **MongoDB Integration**: Form data is securely stored in a MongoDB collection, with schema definitions enforced via Mongoose.
* **RESTful APIs for CRUD Operations**: Backend endpoints handle creation, retrieval, and potential expansion for update/delete operations on surveys.
* **Error Handling**: Proper error messages and fail-safes are implemented for incomplete inputs, server errors, or connectivity issues.
* **Modular Codebase**: Cleanly separated frontend and backend code makes it easy to maintain and scale.
* **Scalable Deployment-Ready Architecture**: The project is structured for deployment on cloud platforms such as Heroku, Render, or Vercel.

### **3. Architecture**

#### **Frontend (React.js):**

The frontend is developed using **React.js**, a powerful JavaScript library for building interactive user interfaces. It is designed with a **component-based architecture**, ensuring reusability and maintainability. Functional components are used throughout the application, making it cleaner and easier to debug.

**React Hooks** like useState, useEffect, and useContext are utilized for efficient state management, handling form inputs, and managing global data. The UI is built using HTML5 and styled with modern **CSS3**, ensuring responsiveness across various screen sizes and devices, including mobile, tablet, and desktop.

Key frontend functionalities include:

* Dynamic rendering of form fields
* Real-time form validation and user feedback
* Seamless integration with backend API using **Axios**
* Modular structure for pages and components (e.g., Home, SurveyForm, AdminDashboard)
* Navigation handled using **React Router**

#### **Backend (Node.js + Express.js):**

The backend is built using Node.js, a runtime environment that enables JavaScript to run on the server, and Express.js, a minimalist web framework that provides a robust set of HTTP utility methods and middleware for API development.

The backend architecture includes:

* RESTful API design for interaction with frontend
* Endpoints for submitting, retrieving, and managing survey data
* Middleware for error handling, request logging, and body parsing (express.json())
* CORS configuration for secure cross-origin API communication
* Input validation on server-side to ensure form integrity and protect against malicious data

Routes are separated logically by purpose (e.g., /api/surveys, /api/admin) for modularity and scalability. The backend is also structured to allow future extensions, such as user authentication, form templates, and analytics.

#### **Database (MongoDB with Mongoose):**

The database layer uses **MongoDB**, a NoSQL document-based database that allows storage of structured and semi-structured survey data in **JSON-like** format. MongoDB is highly scalable, schema-flexible, and ideal for storing user-generated content like form responses.

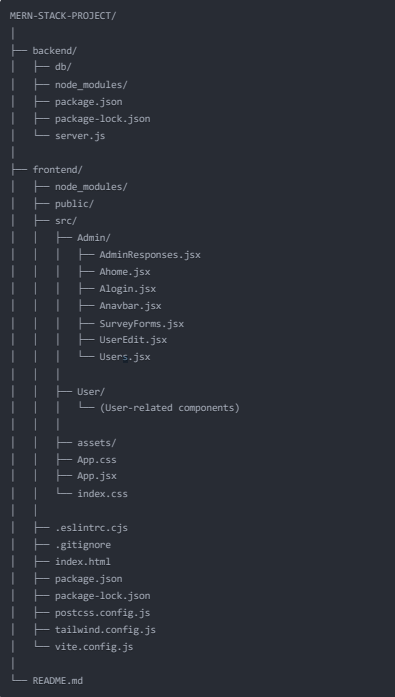
**Mongoose** is used as the Object Data Modeling (ODM) library to interact with MongoDB. It provides schema definitions, model-based data access, and built-in validation.

Key database elements include:

* **Survey Model**: Stores survey title, questions, options, and response structure
* **Response Model**: Captures individual user responses with timestamps
* **Admin/User separation**: Prepared structure for handling role-based access (future scope)

Benefits of MongoDB in this project:

* High performance and horizontal scalability
* Schema flexibility to accommodate varied survey structures
* Easy aggregation and filtering of responses for analytics



**4. Setup Instructions**

#### **Prerequisites:**

Before running the project, ensure the following tools and dependencies are installed on your system:

* **Node.js**: JavaScript runtime required to run the backend and frontend packages.
* **npm or Yarn**: Node package managers to install project dependencies.
* **MongoDB**: A NoSQL database system. You can use:  
  + **MongoDB Atlas (cloud-based)** or
  + **MongoDB Community Edition (local installation)**
* **Git**: Version control system to clone the repository.

**INSTALLATION STEPS:**

**1. Clone the Repository**

Use Git to clone the project from GitHub: git clone https://github.com/Hiralkothari19/MERN-STACK-PROJECT.git

**2. Install Frontend Dependencies**

Navigate to the Frontend folder and install all the necessary npm packages:

cd MERN-STACK-PROJECT/Frontend

npm install



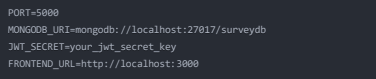
**3. Install Backend Dependencies**

Navigate to the Backend folder and install backend packages:



**4. Configure Environment Variables**

Inside the Backend folder, create a file named .env and define your MongoDB URI and backend port:



**5. Run MongoDB Locally** *(If using local MongoDB)*

Start your MongoDB server (typically this runs on localhost:27017):

Mongod

### **5. Folder Structure**

A well-organized folder structure enhances maintainability and scalability. Below is the folder structure for both the **Frontend** and **Backend**:

#### **Frontend (Client)**

Frontend/

├── public/ # Static public assets

├── src/

│ ├── components/ # Reusable React components

│ ├── App.js # Root component that defines routing and structure

│ ├── index.js # Entry point of the React application

├── package.json # Dependency management and scripts

**App.js**: Contains routing logic and core structure of the React app.

**components/**: Includes form components like input fields, buttons, etc.

**index.js**: Mounts the main App component to the DOM using ReactDOM.

Backend (Server):

Backend/

├── models/

│ └── Survey.js # Defines the schema for survey data using Mongoose

├── routes/

│ └── surveyRoutes.js # API routes to handle form submission and data fetching

├── server.js # Entry point that sets up Express server and routes

├── .env # Contains sensitive environment variables

├── package.json # Backend dependencies and scripts

* **Survey.js**: Mongoose schema that defines the structure of survey responses.
* **surveyRoutes.js**: RESTful routes for survey submission and retrieval.
* **server.js**: Initializes Express app, connects to MongoDB, applies middleware, and starts server.

**6. Running the Application**

**Frontend Server (React App)**

cd Frontend

npm start

**Backend Server (Express + MongoDB)**

cd Backend

npm start

### **7. API Documentation**

The backend RESTful APIs serve as the communication bridge between the frontend interface and the MongoDB database. These endpoints are designed using **Express.js** and follow industry-standard practices.

#### **API Endpoints Overview**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint** | **Method** | **Description** | **Input Format** | **Success Response Example** |
| /api/surveys | POST | Submits a new survey form | { "name": "John", "email": "john@example.com", "answers": {...} } | { "message": "Survey submitted successfully!" } |
| /api/surveys | GET | Fetches all submitted survey responses | None | Array of response objects |
| /api/surveys/:id | GET | Fetches a specific response by ID | URL Parameter: :id | Specific survey object |

#### **Error Handling**

* Each endpoint returns proper **HTTP status codes** such as:  
  + 200 OK: Successful response
  + 201 Created: Resource created successfully
  + 400 Bad Request: Invalid or missing fields
  + 404 Not Found: Survey not found
  + 500 Internal Server Error: Server crash or database failure
* **Error messages** are descriptive to assist developers in debugging.

#### **Security & Validation**

* Data validation is performed using **Express middleware** before processing inputs.
* Fields like name, email, and survey answers are required.
* Schema-level validation is also enforced via **Mongoose** models.

### **8. Authentication & Authorization**

Though the current version does not include user authentication, the project is structured to accommodate it in future versions with ease.

#### **Planned Authentication Features:**

* **JWT (JSON Web Token)**:  
  + After login, users will receive a secure token.
  + Token will be stored in HTTP-only cookies or localStorage to prevent XSS attacks.
* **Role-Based Access Control (RBAC)**:  
  + **User Role**: Can only access form submission pages.
  + **Admin Role**: Has access to a protected dashboard to view, filter, and export submissions.
* **Protected Routes**:  
  + Middleware will verify JWT before granting access.
  + Routes like /api/surveys (GET for admins) will be secured.
* **User Registration & Login APIs**:  
  Future versions will include secure password handling using **bcrypt.js** and **jsonwebtoken**.

### **9. User Interface**

The user interface is designed to be **clean, intuitive, and mobile-friendly** using **React.js functional components** and **plain CSS styling**.

#### **UI Flow**

* **Landing Page**:  
  + Introduction and navigation options.
  + Clear call-to-action to fill a survey.
* **Survey Form Page**:  
  + Includes form fields (text input, dropdown, radio, checkbox).
  + Responsive layout using Flexbox/Grid.
  + Form validation messages for incorrect/missing entries.
* **Thank You Page**:  
  + After submission, users are redirected to a success screen.
* **Admin Dashboard (optional)**:  
  + Table/grid view of survey responses.
  + Pagination, search, and filtering planned for future.
  + Export options (CSV/Excel) in the roadmap.

#### **UX Considerations**

* Error highlighting on invalid fields.
* Confirmation dialogs before form submission.
* Real-time feedback with loaders and success messages.
* Consistent font, color palette, and spacing across all pages.

### **10. Testing**

Testing ensures the application performs reliably across different environments and use cases.

#### **Manual Testing:**

* Each form field tested for:  
  + Required validation
  + Input constraints (e.g., email format)
* Network requests tested with:  
  + **Postman**
  + **Thunder Client (VSCode)**
* Simulated common user actions:  
  + Form submission with correct/incorrect values.
  + Reload scenarios and invalid URL access.

#### **Automated Testing (Planned):**

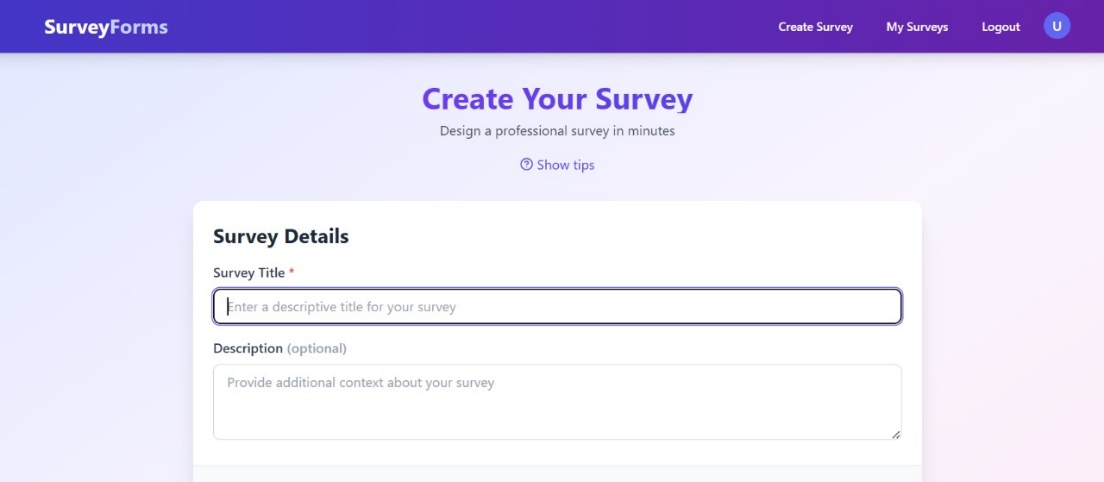
* **Frontend**:  
  + Unit tests with **Jest** and **React Testing Library** for component behavior.
* **Backend**:  
  + API route testing using **Mocha**, **Chai**, or **Supertest**.
* **Integration Tests**:  
  + Simulate the full process: submit form → store in DB → fetch/display.

### **11. Screenshots & Demo**

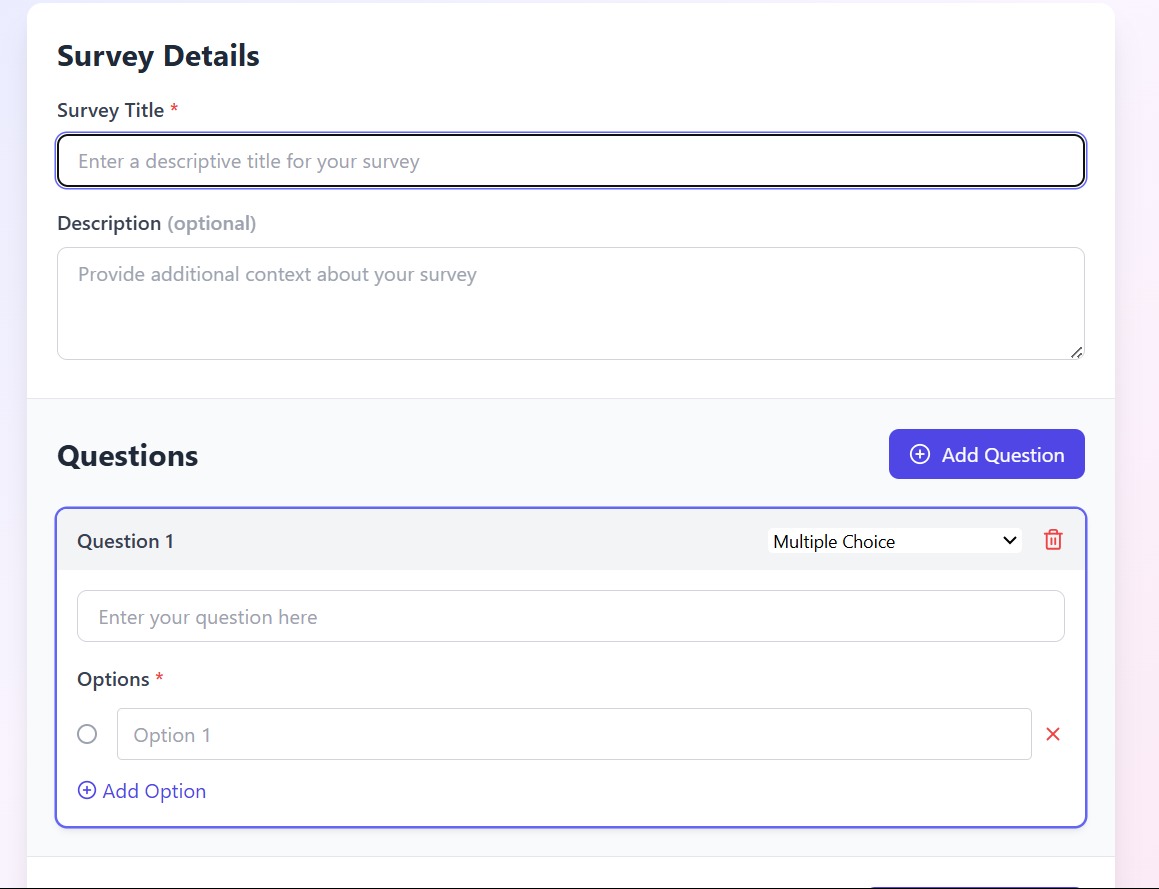
To visually represent the app's functionality and layout, the following assets are included:

#### **Screenshots:**

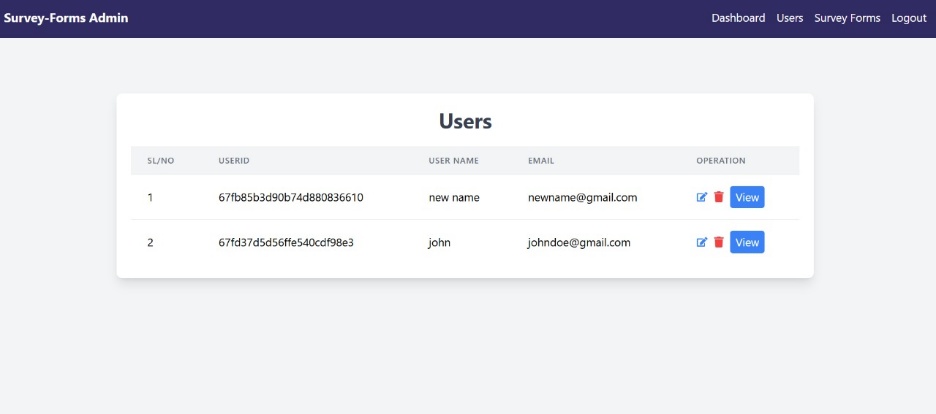
* **Landing Page** – Displays intro and "Start Survey" button.



* **Survey Form Page** – Responsive design with form elements.



* **Thank You Message** – Confirmation after successful submission.



* **(Optional) Admin Dashboard** – Table showing collected data.



#### **Live Demo *(Optional but Recommended)***

Demo Link - <https://drive.google.com/file/d/1nOy_hm2zn4H7kgQH024rq-GWkR2FDnYM/view?usp=sharing>

To make the app accessible:

* **Frontend**: Deploy on **Netlify**, **Vercel**, or **GitHub Pages**.
* **Backend**: Host API using **Render**, **Railway**, or **Heroku**.
* **Database**: MongoDB Atlas cloud deployment.

### **12. Known Issues**

Every software project faces limitations in its early stages. Here are the known constraints:

* No login/signup functionality implemented yet.
* Admin dashboard lacks edit/delete operations.
* No API rate-limiting or security throttling.
* No centralized error logger (e.g., Winston).
* Form design can be improved with UI libraries (TailwindCSS/Bootstrap).
* API does not support pagination for large datasets.

These issues are under consideration for upcoming releases.

### **13. Future Enhancements**

The project has strong potential for growth in terms of features, usability, and scalability.

#### **Planned Improvements:**

* **User Authentication System** with JWT and RBAC.
* **Admin Control Panel** to manage surveys, export data, and analytics.
* **Survey Builder UI**: Drag-and-drop interface for creating custom surveys.
* **Email Notifications** after submission.
* **CSV/Excel Export** for admin insights.
* **Mobile App** version using React Native.
* **Analytics Dashboard**: Charts and graphs to show trends.
* **Multi-language Support** for international users.
* **Autosave Form** Progress for longer surveys.