**Full Stack Development with MERN Project Documentation**

# 1. Introduction

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

| **NAME** | **ROLE** | **KEY RESPONSIBILITIES** |
| --- | --- | --- |
| **Syed Atif Hussain** | **Full Stack Lead Developer** | **Architecting system, backend API development, database design, deployment pipeline** |
| **Shreya Upadhyay** | **Frontend Engineer** | **Implementing responsive UI, state management, form validation, chart visualizations** |
| **Mohammad Rammez Khan** | **UI/UX Designer** | **User flows, wireframing, design system, accessibility compliance** |
| **Hemansh Bhagtani** | **QA & Testing Engineer** | **Test automation, security audits, performance benchmarking** |

# 2. Project Overview

**Purpose:**

1. To build a feature-rich survey application with:
2. Secure user authentication
3. Dynamic survey creation
4. Real-time response collection
5. Data visualization

**Features:**

|  |  |
| --- | --- |
| FEATURE | DESCRIPTION |
| **User Auth** | JWT-based authentication |
| **Survey Creation** | Drag-and-drop form builder |
| **Response Analysis** | Real-time charts and statistics |
| **Multi-Question** | Support for various question types |
| **Admin Dashboard** | Manage surveys and view analytics |

# 3. Architecture

* **Frontend:**
* **Libraries**: react-router-dom, axios, chart.js
* **State Management** :Redux for global state
* **Styling**: CSS Modules + Bootstrap

* **Backend:**
* **RESTful API** with:
* JWT authentication middleware
* Role based access control
* Validation with express validator

* **Database:**

// User Schema

{

username: String,

email: String,

password: String, // Hashed

role: String // 'admin' or 'user'

}

// Survey Schema

{

title: String,

questions: [{

text: String,

type: String, // 'multiple-choice', 'text', etc.

options: [String]

}],

createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

}

# 4. Setup Instructions

* **Prerequisites:**

1. Node.js v18+
2. MongoDB Atlas account
3. Git

* **Installation:**

1. Clone the repo:

git clone https://github.com/KnightmareAtif/SurveyAppMERN.git

cd SurveyAppMERN

1. Backend setup:

cd backend

npm install

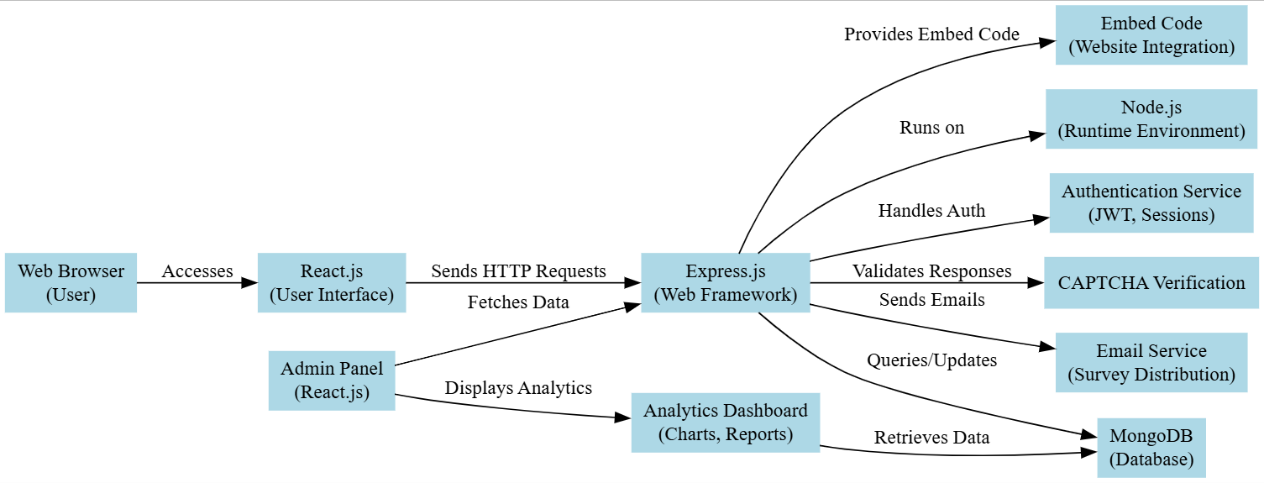
cp .env.example .env # Add your MongoDB\_URI, JWT\_SECRET

1. Frontend setup:

cd frontend

npm install

# 5. Folder Structure



* **Client:**

frontend/

├── public/

├── src/

│ ├── components/

│ │ ├── auth/

│ │ ├── dashboard/

│ │ └── surveys/

│ ├── redux/

│ │ ├── actions/

│ │ └── reducers/

│ ├── pages/

│ ├── services/

│ └── styles/

* **Server:**

backend/

├── config/

├── controllers/

│ ├── authController.js

│ └── surveyController.js

├── middleware/

├── models/

├── routes/

├── utils/

├── app.js

└── server.js

1. **Client-Side Organization**:
   * Logical separation of components (UI), contexts (state), and services (API calls).
   * Route-based pages for better scalability.
2. **Server-Side Modularity**:
   * MVC pattern (models, controllers, routes).
   * Dedicated middleware for auth and error handling.
3. **Scalability**:
   * Easy to add new features (e.g., /services/paymentService.js).
   * Clear separation of concerns (e.g., validation.js for input checks).

# 6. Running the Application

o **Frontend:** cd frontend && npm start # Port 3000

**Backend:** cd frontend && npm start # Port 3000

# 7. API Documentation

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **Method** | **Body (Example)** | **Response (200)** |
| /api/auth/register | POST | { email: "user@demo.com" } | { token: "jwt\_token" } |
| /api/survey | GET | - | [{ id: 1, title: "Survey"}] |

**Example Request**:

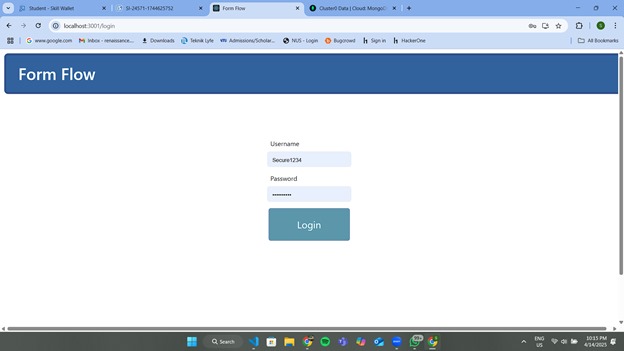
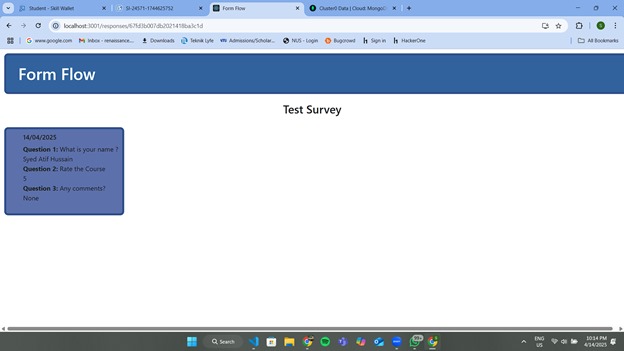
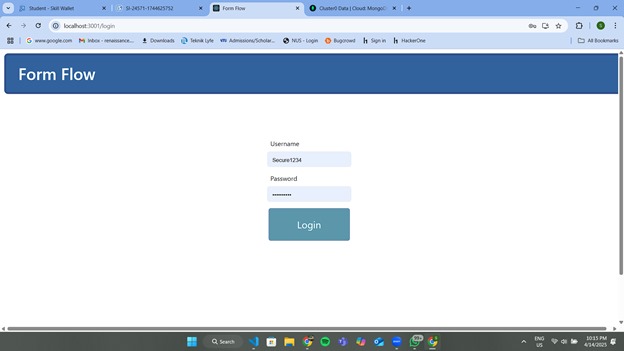
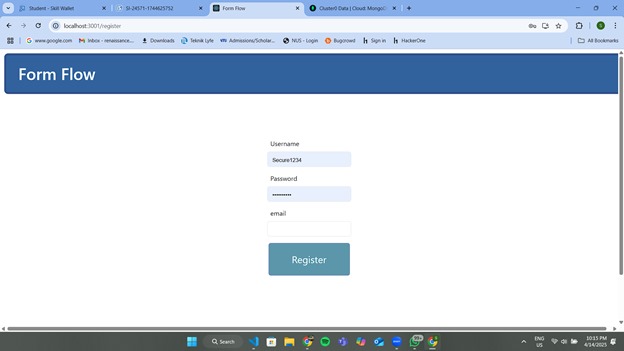
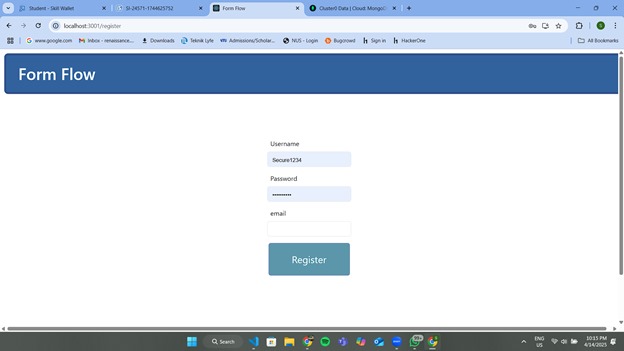
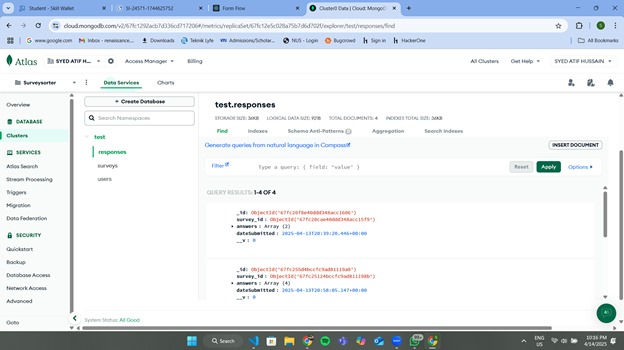
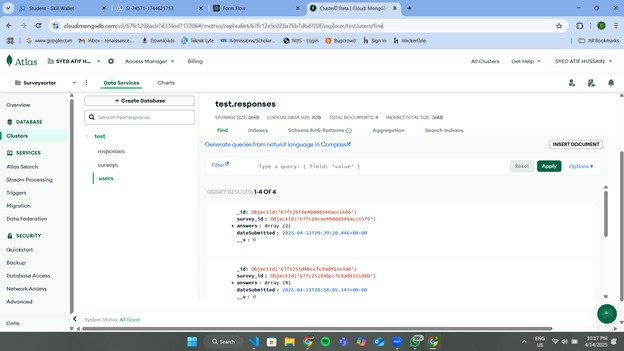
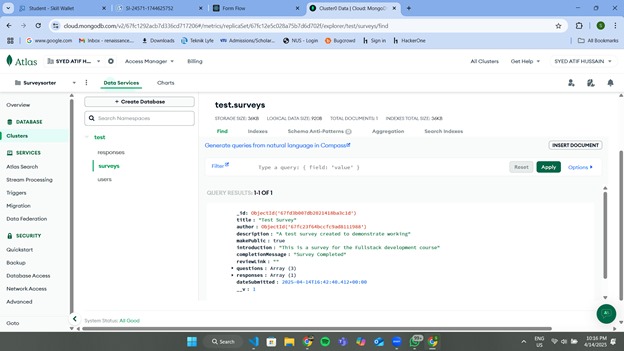
# POST /api/auth/login

# { "email": "test@user.com", "password": "123456" }

# 8. Authentication

* **Flow**:
  1. User registers → Server creates account → Returns JWT
  2. Token stored in localStorage → Attached to API requests via axios interceptor
* **Security**:
  1. Passwords hashed with bcryptjs
  2. JWT expires in **1h**

1. **User Interface**

1. **Testing**

A robust testing strategy ensures the reliability, security, and performance of the **SurveyAppMERN** platform. We follow the **Testing Pyramid** methodology to maximize efficiency and coverage:

**1. Unit Testing (70%):**

Objective: Test individual components, functions, and logic in isolation.

Tools Used:

* Jest (JavaScript testing framework)
* React Testing Library (for React components)

Test Coverage:

Frontend:

Components:

* Auth forms (Login/Register)
* Survey creation UI (drag-and-drop, question types)
* Dashboard widgets
* Redux:
  + Action creators
  + Reducers (state updates)
* Utility Functions:
  + Input validation, date formatting

Backend:

* API Controllers:
* authController (login, registration)
* surveyController (CRUD operations)
* Middleware:
  + Authentication checks
  + Error handling
* Models:
  + Database schema validation

**2. Integration Testing (20%)**

**Objective:** Verify interactions between modules (frontend + backend, APIs + DB).

**Tools Used:**

* **Supertest** (HTTP assertions for Express.js)
* **Jest** (for mocking API responses)

**Test Coverage:**

**API Endpoints:**

* Authentication flow (/api/auth/login, /api/auth/register)
* Survey CRUD operations (/api/surveys)
* Response submission (/api/responses)

**Frontend-Backend Integration:**

* Form submissions → API calls
* Data fetching (e.g., loading surveys in dashboard)

**3. End-to-End (E2E) Testing (10%)**

**Objective:** Simulate real user workflows across the entire system.

**Tools Used:**

* **Cypress** (interactive browser testing)

**Test Scenarios:**

**User Journey:**

1. **Registration → Login → Survey Creation → Response Submission**
2. **Admin Dashboard Access → Survey Analytics View**

**Edge Cases:**

* Invalid form submissions
* Unauthorized access attempts

1. **Demo :**
2. **Known Issues**

|  |  |
| --- | --- |
| **Issue** | **Workaround** |
| Slow survey loading with many questions | Implement pagination |
| Image uploads not optimized | Use compression middleware |

# 13. Future Enhancements

**1. Advanced Question Types**

**Problem:** Limited question types restrict survey flexibility.  
**Solution:** Support complex question formats.

**New Question Types to Add:**

| **Type** | **Description** | **Example Use Case** |
| --- | --- | --- |
| **Matrix Grid** | Likert-scale questions in a grid format (rows = statements, columns = ratings) | Employee satisfaction surveys |
| **File Upload** | Allow respondents to upload images/docs | Customer feedback with screenshots |
| **Ranking** | Drag-and-drop prioritization of options | Product feature prioritization |
| **Signature** | Digital signature capture | Contract agreements |

**Technical Implementation:**

**Backend Schema Update:**

// Extended Survey Schema

{

questions: [{

type: {

type: String,

enum: ['multiple-choice', 'text', 'matrix', 'file', 'ranking', 'signature']

},

matrixRows: [String], // For grid questions

fileTypes: [String], // e.g., ['image/png', 'application/pdf']

maxFileSize: Number // in MB

}]

}

**Frontend Components:**

* Use **react-beautiful-dnd** for ranking questions
* **react-dropzone** for file uploads

**2. Real-Time Collaboration**

**Problem:** Only one user can edit a survey at a time.  
**Solution:** Multi-user editing with live sync.

**Tech Stack:**

* **Socket.io** (WebSockets)
* **Operational Transform (OT)** / **CRDTs** for conflict resolution

**Implementation Plan:**

1. **Backend:**

// Socket.io server

io.on('connection', (socket) => {

socket.on('join-survey', (surveyId) => {

socket.join(surveyId);

});

socket.on('edit-question', (data) => {

io.to(data.surveyId).emit('question-updated', data);

});

});

1. **Frontend:**
   * Show live cursors (like Google Docs) with **PartyKit** or **Yjs**
   * Conflict resolution for concurrent edits

**3. Export Capabilities**

**Problem:** Users need offline access to survey data.

**Export Options:**

| **Format** | **Library** | **Features** |
| --- | --- | --- |
| **PDF** | pdfkit | Branded reports with charts |
| **Excel** | exceljs | Pivot tables, raw data export |
| **CSV** | json2csv | Lightweight spreadsheet format |

**Example (PDF Generation):**

const PDFDocument = require('pdfkit');

const doc = new PDFDocument();

doc.text('Survey Report: Customer Feedback', { align: 'center' });

survey.responses.forEach((response) => {

doc.text(`Q: ${response.question} | A: ${response.answer}`);

});

doc.pipe(fs.createWriteStream('report.pdf'));

doc.end();

**4. Theming System**

**Problem:** Surveys look generic; users want branding.

**Customization Features:**

* **Color Picker** (Primary/secondary colors)
* **Font Selector** (Google Fonts integration)
* **CSS Variables** for dynamic theming:

css

Copy

:root {

--primary-color: #4285f4;

--font-family: 'Roboto', sans-serif;

}

* **Save Themes** to user profiles

**5. AI-Powered Insights**

**Problem:** Manual analysis of text responses is time-consuming.

**AI Integrations:**

| **Feature** | **Tool/API** | **Output Example** |
| --- | --- | --- |
| Sentiment Analysis | TensorFlow.js / Hugging Face | "80% positive feedback" |
| Topic Modeling | GPT-3.5 (OpenAI API) | "Common themes: Pricing, Support" |
| Auto-Tagging | Natural Language Toolkit (NLTK) | Tags: #billing, #usability |

**Implementation:**

// Sentiment analysis API call

const analyzeText = async (text) => {

const response = await fetch('https://api.openai.com/v1/chat/completions', {

method: 'POST',

headers: { 'Authorization': `Bearer ${API\_KEY}` },

body: JSON.stringify({

model: 'gpt-3.5-turbo',

messages: [{ role: 'user', content: `Analyze sentiment: "${text}"` }]

})

});

return response.json();

};

**6. Mobile App (React Native)**

**Problem:** Mobile users need a dedicated app.

**Approach:**

* **Code Reuse:** Share 70% logic between React Web and React Native
* **Libraries:**
  + react-native-paper (UI components)
  + react-navigation (routing)

**Key Screens:**

1. Survey List (Offline support with **SQLite**)
2. Response Collector (Camera integration for file uploads)
3. Dashboard (Sync via **Firebase/FastAPI**)

**7. Payment Gateway (Monetization)**

**Problem:** Free tier limits scalability.

**Stripe Integration Flow:**

1. **Frontend:**

// Stripe Elements in React

<Elements stripe={stripePromise}>

<CheckoutForm priceId="price\_123" />

</Elements>

1. **Backend:**

// Node.js webhook

app.post('/stripe-webhook', (req, res) => {

const event = stripe.webhooks.constructEvent(...);

if (event.type === 'payment\_succeeded') {

grantPremiumAccess(event.data.object.customer);

}

});

**Subscription Tiers:**

| **Tier** | **Price** | **Features** |
| --- | --- | --- |
| **Free** | $0 | Basic surveys, 100 responses/mo |
| **Pro** | $15/mo | Advanced analytics, unlimited |
| **Enterprise** | Custom | White-labeling, API access |

**Prioritization**

1. **Payment Gateway** (Revenue generation)
2. **Mobile App** (User reach)
3. **AI Insights** (Competitive edge)