**JAGAN INSTITUTE OF MANAGEMENT STUDIES**

**SECTOR – 5, ROHINI,NEW DELHI**



**(Affiliated To)**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY**

**SECTOR – 16 C, DWARKA, NEW DELHI**



**Project: Mystery Messages**

Submitted By:  **Shivansh** Enrollment No.: **00214004424**

MCA 2nd Year (Section - A) [3rd Semester, Batch: 2024-2026]

**Table of Contents**

| **Chapter No.** | **Title** | **Subsections** |
| --- | --- | --- |
| **1** | **Introduction** | **1.1 Background & Problem Statement**  **1.2 Objectives of the Project**  **1.3 Scope & Significance**  **1.4 Technologies Used** |
| **2** | **System Analysis** | **2.1 Existing System & Its Limitations**  **2.2 Proposed System & Its Advantages**  **2.3 Feasibility Study**  **- Technical Feasibility**  **- Economic Feasibility**  **- Operational Feasibility**  **2.4 Requirement Specification**  **- Functional Requirements**  **- Non-functional Requirements** |
| **3** | **Implementation** | **3.1 System Overview**  **3.2 Getting Started**  **3.3 Installation & Setup**  **3.4 Screenshots of the Running System**  **3.5 Code Snippets**  **3.6 Tools, Languages, and Frameworks Used**  **3.7 Testing Strategy & Test Cases** |
| **4** | **References** | **4.1 Books**  **4.2 Research Papers**  **4.3 Websites**  **4.4 Tools & Libraries Used** |

**Chapter 1: Introduction**

**1.1 Background & Problem Statement**

In an era where digital communication dominates, users crave more than just instant messaging—they seek meaningful, engaging, and anonymous interactions. Traditional messaging platforms often lack the element of mystery, novelty, or emotional depth, which are essential to sustaining user curiosity and connection.

**Mystery Messages** addresses this gap by offering a unique, interactive messaging experience designed to uncover secrets, spark curiosity, and build deeper connections between users. The system allows users to send and receive anonymous messages, adding intrigue while maintaining security and control.

A key use case is **receiving anonymous feedback**—whether it's for personal growth, peer evaluation, or candid communication. Most platforms lack a secure and user-friendly way to collect such feedback without compromising trust or privacy. Mystery Messages offers a simple, structured channel for this type of interaction.

The current market lacks customizable, secure, and developer-friendly messaging solutions that include real-time capabilities, email integration, and extensible UI components in one unified framework. This project tackles these pain points through a robust, scalable full-stack solution.

**1.2 Objectives of the Project**

The primary objectives of **Mystery Messages** are:

* To build a secure, full-stack social messaging platform with real-time capabilities.
* To offer developers a toolkit that simplifies the integration of core messaging features.
* To support anonymous/semi-anonymous interactions while preserving user privacy and control.
* To integrate authentication and email-based verification using tools like Resend.
* To create a visually appealing, customizable UI component library using Tailwind CSS and React.
* To provide a reliable backend with database connectivity and optimized API endpoints.

**1.3 Scope & Significance**

**Scope:**

* The project is scoped as a modular, extensible web-based platform.
* Focuses on one-on-one messaging with optional anonymous features.
* Includes email verification, message delivery, and basic moderation tools.
* Supports both developers (as a toolkit) and end users (as a product).

**Significance:**

* Helps developers accelerate the creation of messaging platforms without reinventing the wheel.
* Provides end users with a fresh take on digital communication by blending mystery and functionality.
* Encourages curiosity-driven interactions while ensuring a safe and structured environment.
* Demonstrates modern full-stack development practices using industry-standard tools.

**1.4 Technologies Used:**

The following technologies and tools were used in developing **Mystery Messages**:

| **Category** | **Technologies & Tools** |
| --- | --- |
| **Frontend** | **Next.js**, TypeScript, Tailwind CSS, React Hook Form, Zod |
| **Backend** | |  | | --- | |  |  |  | | --- | | API routes in Next.js (Server-side), Mongoose | |
| **Database** | MongoDB (local/cloud instance) |
| **Authentication** | Custom auth with session/token handling, Email verification (Resend) |
| **APIs & Networking** | Axios for HTTP requests, RESTful APIs |
| **Code Quality** | ESLint, Prettier |
| **Data Format** | JSON, Markdown (for documentation) |

**Chapter 2: System Analysis**

**2.1 Existing System & Its Limitations**

Most existing messaging platforms (like WhatsApp, Messenger, Instagram DM) focus on identity-based communication, prioritizing speed and media sharing. While effective for general use, they **lack features like anonymous feedback**, built-in developer tooling, and custom integrations for real-time interactions or modular UI systems.

**Limitations of Existing Systems:**

* No support for **anonymous or semi-anonymous messaging** in a structured way.
* Feedback systems are typically third-party, clunky, or untrustworthy.
* Lack of real-time customization tools for developers building niche communication apps.
* Authentication systems are either too basic or require complex integrations.
* Frontend UI kits are often inconsistent or require building from scratch.
* Limited flexibility in building scalable, secure, and fast full-stack messaging solutions.

**2.2 Proposed System & Its Advantages**

**Mystery Messages** proposes a modular, scalable messaging platform that prioritizes curiosity, privacy, and engagement, especially in scenarios like feedback collection, anonymous interaction, and social discovery.

**Key Advantages:**

* **Anonymous Messaging Support**: Enables sending/receiving anonymous feedback safely.
* **Secure Authentication**: Includes email verification and session/token management via Resend.
* **Real-time Messaging API**: Backend endpoints for live messaging, retrieval, and deletion.
* **Component-Based UI**: Reusable, styled components using Tailwind CSS & React.
* **Full-stack with SSR**: Next.js enables server-side rendering, fast routing, and API handling.
* **Developer Toolkit**: Designed to be repurposed or extended by other developers.
* **MongoDB Integration**: Efficient, scalable NoSQL database connectivity with Mongoose.

**2.3 Feasibility Study**

**Technical Feasibility:**

* Built with modern technologies: Next.js, TypeScript, MongoDB.
* Easily deployable on cloud platforms like Vercel, Render, or Railway.
* Lightweight frontend with Tailwind ensures performance across devices.
* Email verification handled securely with Resend API.

**Economic Feasibility:**

* Uses free/open-source tools (Next.js, MongoDB Atlas free tier, Tailwind CSS).
* Low hosting costs for MVP deployment.
* APIs and infrastructure can scale with demand.

**Operational Feasibility:**

* Simple UI and UX make it easy for users to interact with the system.
* Admin or user moderation tools can be added easily.
* Can be used as a standalone app or integrated into other platforms.

**2.4 Requirement Specification**

**Functional Requirements:**

* User registration, login, and email verification.
* Ability to send and receive anonymous messages.
* Real-time message delivery and inbox refresh.
* Feedback system with basic moderation/reporting.
* Message deletion and archival functionality.
* User profile management (optional features).

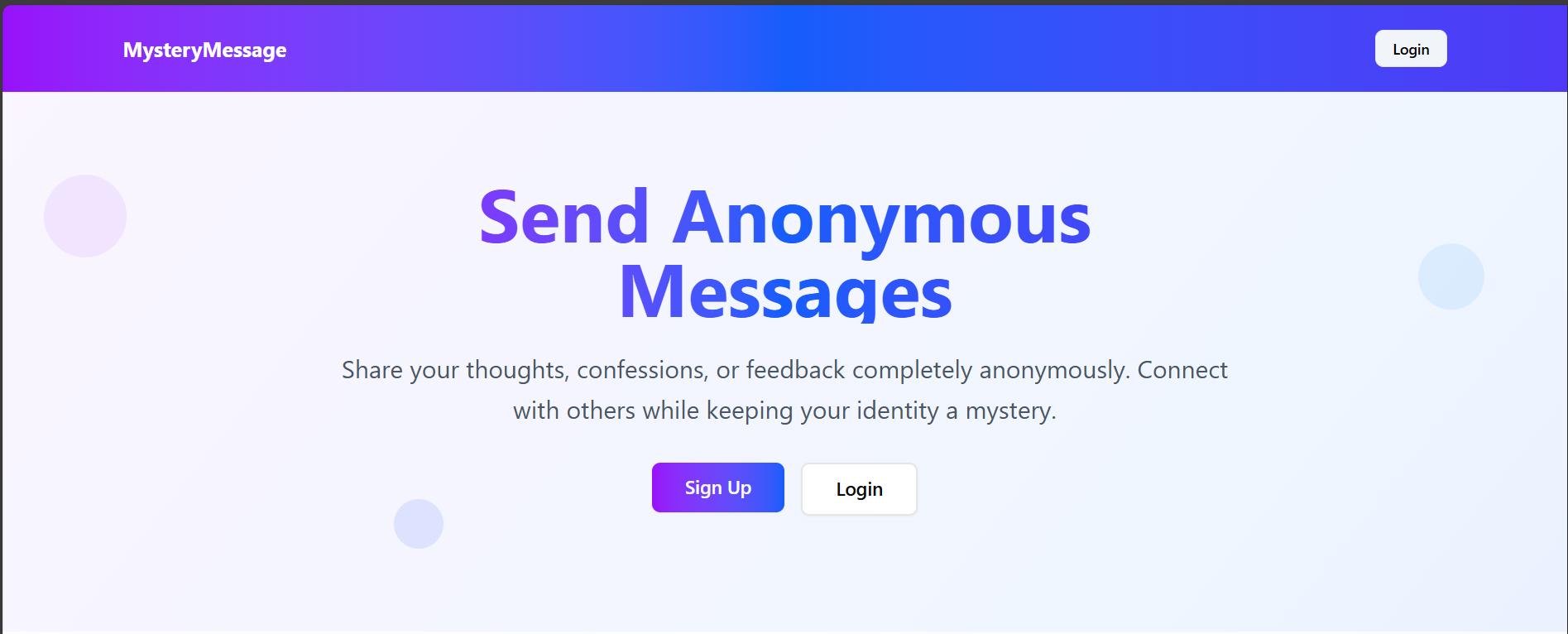
**Non-Functional Requirements:**

* **Performance**: Real-time updates with minimal latency.
* **Scalability**: Easily scales via Next.js API routes and MongoDB.
* **Security**: Data encryption, secure login, input validation (Zod).
* **Usability**: Simple, intuitive interface.
* **Reliability**: Handles connection drops, server errors gracefully.
* **Maintainability**: Modular code with ESLint and TypeScript checks.

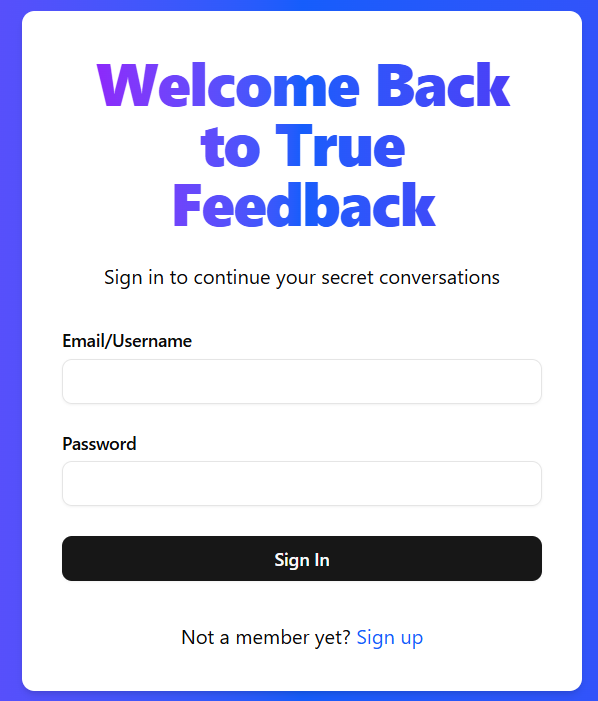
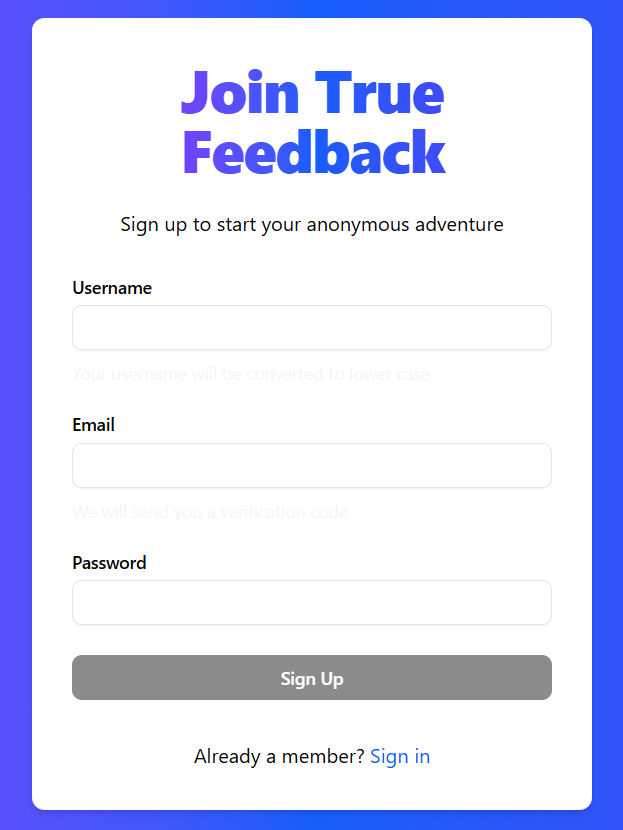
**Chapter 3: Implementation**

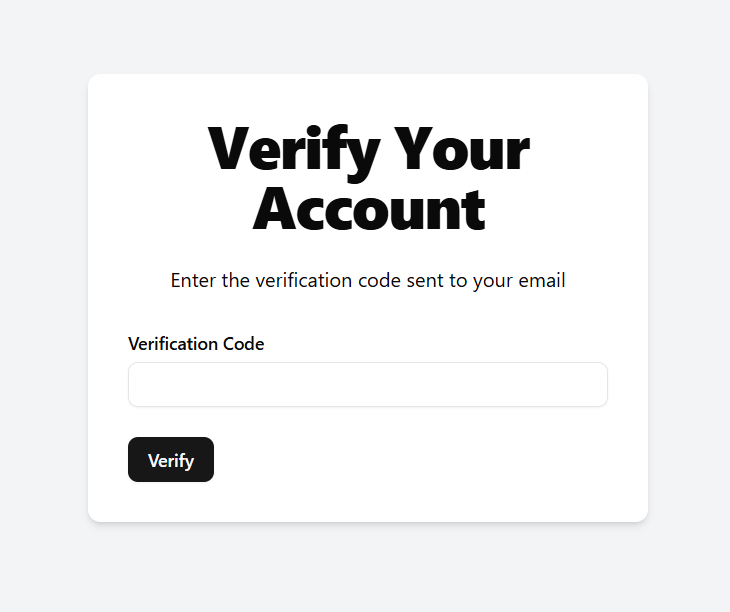
**3.1 Screenshots of the Running System:**

**3.1.1 Home page displaying project overview and call to action –**

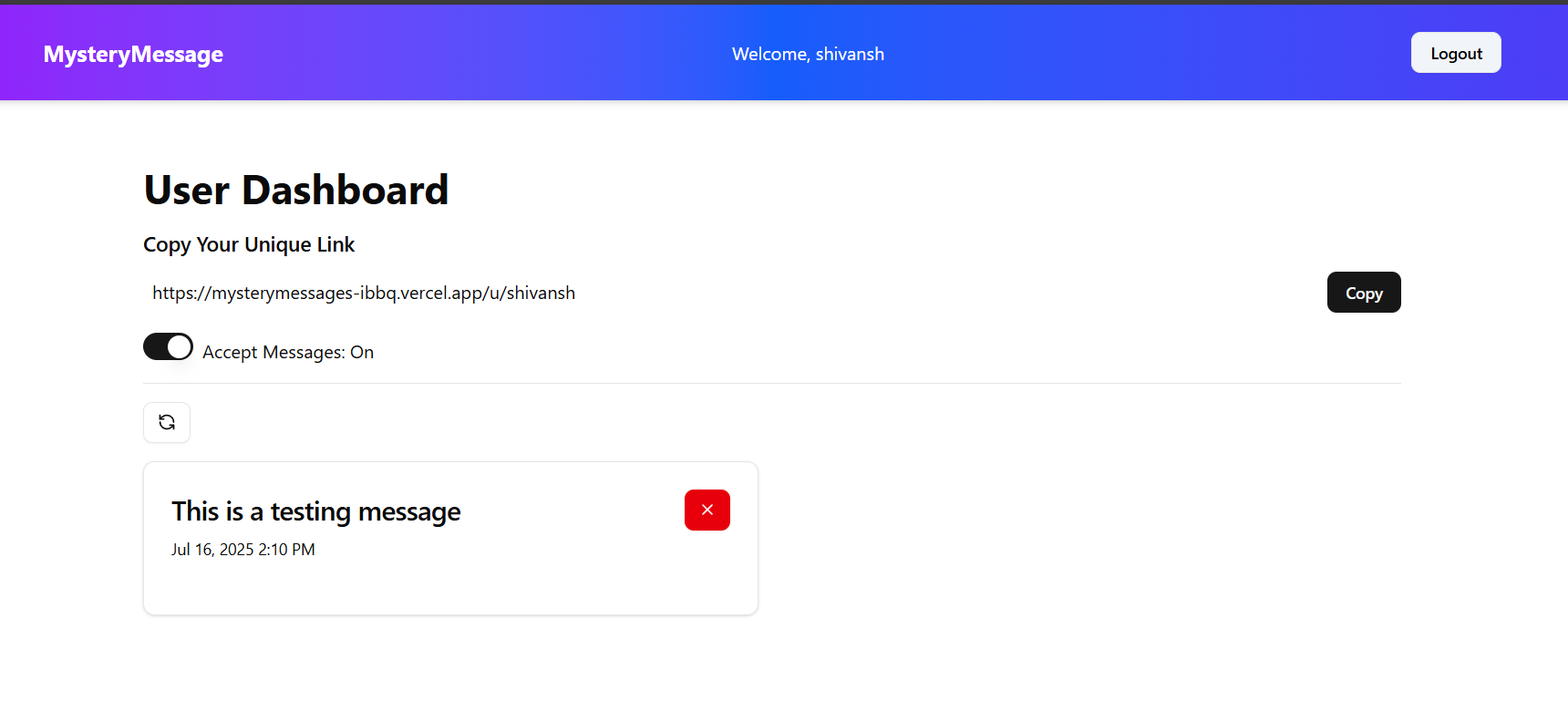
****

**3.1.2 Registration and login screen with email verification -**

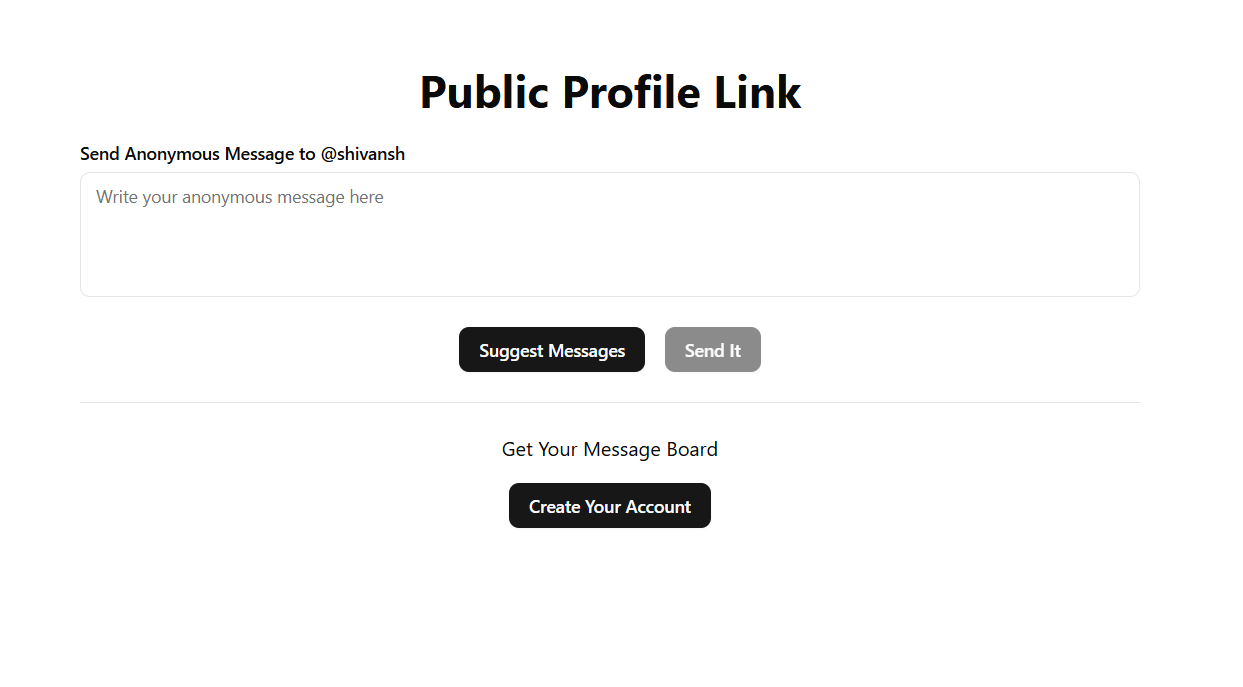
**** ****

****

**3.1.3 Message inbox showing received anonymous messages –**

****

**3.1.4 Compose screen for sending a message anonymously –**

****

**3.2 Code Snippets**

**Below are some representative code snippets from key parts of the application:**

**3.2.1 User Authentication (Next.js API Route) –**

export const authOptions: NextAuthOptions = {

  providers: [

    CredentialsProvider({

      id: "credentials",

      name: "Credentials",

      credentials: {

        email: { label: "Email", type: "text" },

        password: { label: "Password", type: "password" },

      },

      async authorize(credentials: any): Promise<any> {

        await dbConnect();

        try {

          const user = await UserModel.findOne({

            $or: [

              { email: credentials.identifier },

              { username: credentials.identifier },

            ],

          });

          if (!user) {

            throw new Error("No user found with this email");

          }

          if (!user.isVerified) {

            throw new Error("Please verify your account first.");

          }

          const isPasswordCorrect = await bcrypt.compare(

            credentials.password,

            user.password

          );

          if (isPasswordCorrect) {

            return user;

          } else {

            throw new Error("Invalid Password");

          }

        } catch (error: any) {

          throw new Error(error);

        }

      },

    }),

  ],

**3.2.2 Real-Time Messaging with MongoDB + Next.js API –**

    const newMessage = { content, createdAt: new Date() };

    user.messages.push(newMessage as Message);

    await user.save();

    return Response.json(

      {

        success: true,

        message: "Message sent successfully",

      },

      {

        status: 200,

      }

    );

**3.2.3 Zod Validation Example for Form Inputs –**

import { z } from "zod";

export const messageSchema = z.object({

  content: z

    .string()

    .trim()

    .min(10, { message: "Content must be of atleast 10 characters" })

    .max(300, { message: "Content must be of atmost 300 characters" }),

});

**3.3 Tools, Languages, Frameworks Used**

| **Category** | **Details** |
| --- | --- |
| **Programming Language** | **TypeScript** |
| **Frontend Framework** | **Next.js, React, Tailwind CSS** |
| **Backend & API** | **Next.js API Routes, Mongoose, RESTful APIs** |
| **Database** | **MongoDB (via Mongoose ODM)** |
| **Authentication** | **Custom auth, JWT/sessions, Resend email verification** |
| **Form Handling** | **React Hook Form + Zod** |
| **HTTP Client** | **Axios** |
| **Linting & Formatting** | **ESLint, Prettier** |
| **Version Control** | **Git (hosted on GitHub)** |

**3.4 Testing Strategy & Test Cases**

**Testing Strategy:**

* Manual testing for form validation, user flows, and messaging.
* Unit testing of utility functions.
* API route tests using mock requests.
* Basic integration testing on user registration to message delivery.

**References**

| **Type** | **Source** |
| --- | --- |
| **Documentation** | [Next.js Docs](https://nextjs.org/docs), [React Docs](https://reactjs.org/) |
| **Database** | [MongoDB Docs](https://docs.mongodb.com/) |
| **Form & Validation** | [React Hook Form](https://react-hook-form.com/), [Zod](https://zod.dev/) |
| **Authentication** | [Resend](https://resend.com/), [JWT Docs](https://jwt.io/) |
| **UI & Styling** | [Tailwind CSS](https://tailwindcss.com/), [Shadcn](https://ui.shadcn.com/docs) |
| **Code Quality** | [ESLint](https://eslint.org/), [Prettier](https://prettier.io/) |