

A

Project Report

On

# **Real Time Chat Messaging Application**

Submitted in partial fulfilment of the requirement for the 6<sup>th</sup> semester

**Bachelor of Computer Science**

Submitted by

**Bhupendra Singh (2261145)**

**Himanshu Singh (226172)**

**Jitendra Harbola (2261289)**

**Manish Kumar Dani (2361010)**

Under the guidance of

**Mr. Anubhav Bewerwal**

**Asisstant Professor**

**Department of CSE**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS**

**SATTAL ROAD, P.O. BHOWALI,**

**DISTRICT- NAINITAL- 263132**

**2024-2025**

## **STUDENT'S DECLARATION**

We, **Bhupendra Singh, Himanshu Singh, Jitendra Harbola And Manish Kumar Dani** hereby declare the work, which is being presented in the project, entitled ‘ **Real Time Chat Messaging Application** ’ in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Anubhav Bewerwal.

The matter embodied in this project has not been submitted by me for the award of any other degree.

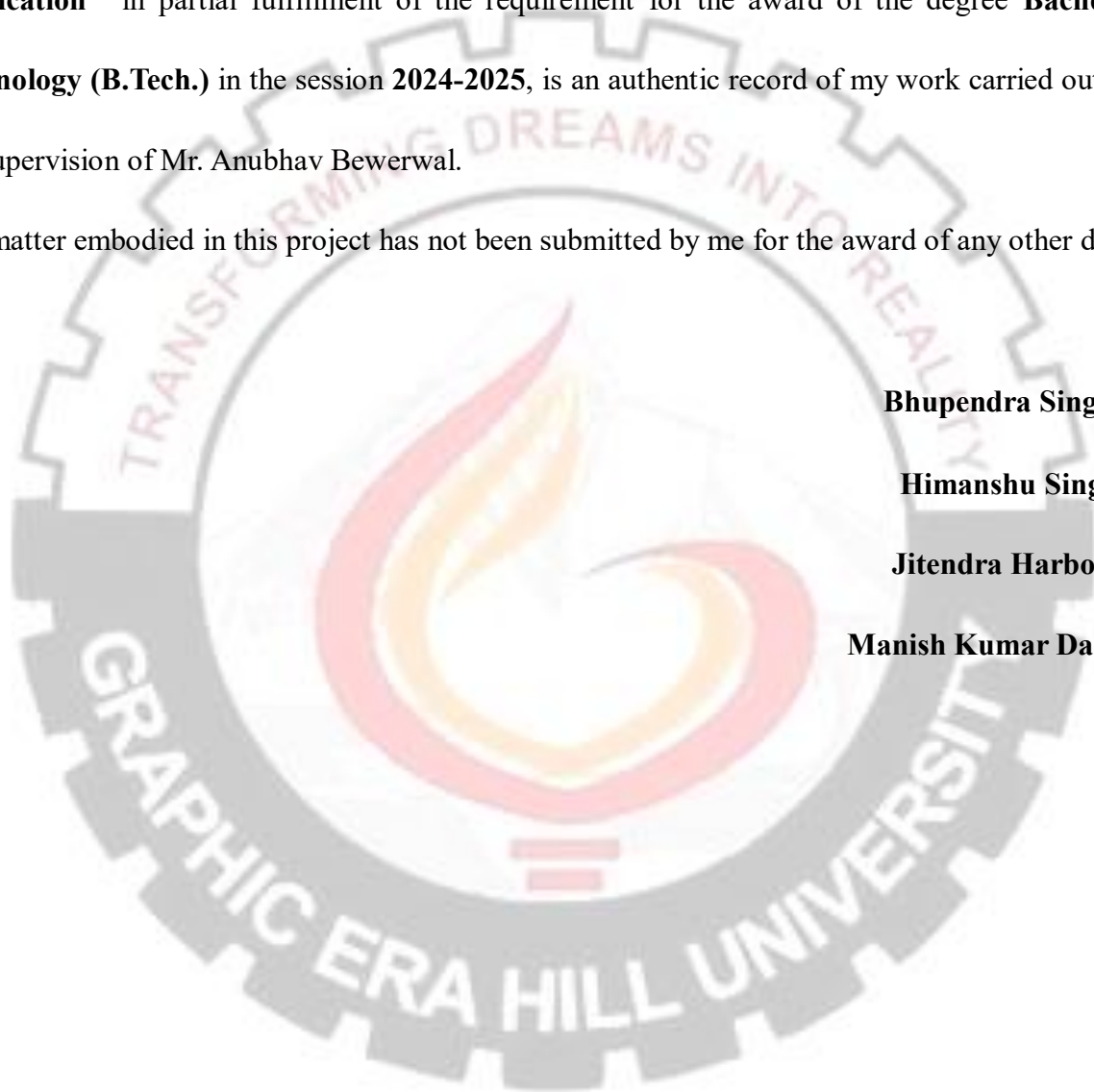
Date:

**Bhupendra Singh.**

**Himanshu Singh.**

**Jitendra Harbola.**

**Manish Kumar Dani.**



## **CERTIFICATE**

The project report entitled “**Real Time Chat Messaging Application**” being submitted by **Bhupendra Singh (2261145)**, **Himanshu Singh (2261272)**, **Jitendra Harbola (2261289)** and **Manish Kumar Dani (2361010)** to Graphic Era Hill University, Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of report.

**Mr. Anubhav Bewerwal**

**(Project Guide)**

**Dr. Ankur Singh Bisht**

**( HOD, CSE )**



## **ACKNOWLEDGEMENT**

We take immense pleasure in thanking the Honorable Director ‘**Prof. (Col.) Anil Nair (Retd.)**’, GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need.

We again want to extend thanks to our president ‘**Prof. (Dr.) Kamal Ghanshala**’ for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to ‘**Dr. Ankur Singh Bisht**’ (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide ‘**Mr. Anubhav Bewerwal**’ (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

**Bhupendra Singh**

**Himanshu Singh**

**Jitendra Singh**

**Manish Kumar Dani**

## **ABSTRACT**

This project is a modern real-time chat application built with **Next.js 13 (App Router)**, **React**, **TypeScript**, and **TailwindCSS**. It allows users to chat instantly without refreshing the page, thanks to **Upstash Redis** and its **Pub/Sub system**. Messages are delivered and shown in real-time for a smooth and fast user experience.

The app supports **one-on-one messaging** and is built to handle many users at once, without slowing down. It also includes a complete **friendship system**—users can send, accept, or decline friend requests and manage their friend list easily. All user relationships are securely managed on the backend to prevent any misuse.

**Redis** is used as the main database because of its fast in-memory performance, which is great for features like **real-time messaging** and **showing who's online**.

The user interface is designed using **TailwindCSS**, making it clean, responsive, and mobile-friendly. Whether on desktop or phone, the app looks and works great. To keep things secure, only logged-in users can access chat rooms, profiles, or friend features. **Google Sign-In via NextAuth.js** makes logging in simple and safe—no need for passwords.

Using **Upstash Redis** (a serverless and scalable database), the app doesn't need complex backend setup. It's cost-effective, handles traffic automatically, and works fast around the globe.

This setup makes the chat app ready for real-world use. It can be deployed easily on platforms like **Vercel**—perfect for personal use, startups, or any project needing a reliable real-time chat system.

# **TABLE OF CONTENTS**

Declaration.....	2
Certificate.....	3
Acknowledgement.....	4
Abstract.....	5
Table of Contents.....	6-7
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>8-12</b>
1. Prologue.....	8
2. Background and Motivations.....	8
3. Problem Statement.....	9
4. Objectives and Research Methodology.....	9
5. Project Organization.....	10-12
<b>CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE</b>	
1. Hardware Requirements.....	13
2. Software Requirements.....	13-14
<b>CHAPTER 3 CODING OF FUNCTIONS... ..</b>	<b>15-17</b>
<b>CHAPTER 4 SNAPSHOT... ..</b>	<b>18-19</b>



<b>CHAPTER 5 LIMITATIONS (WITH PROJECT) .....</b>	<b>20-21</b>
<b>CHAPTER 6 ENHANCEMENTS... ..</b>	<b>22</b>
<b>CHAPTER 7 CONCLUSION .....</b>	<b>23</b>
<b>REFERENCES.....</b>	<b>24</b>



# **INTRODUCTION**

## **1. PROLOGUE**

In an era where real-time communication is central to digital interaction, this project aims to build a modern, full-stack chat application that delivers instant messaging with unmatched speed and responsiveness. Leveraging the power of Next.js 13, React, TypeScript, and Upstash Redis, the application is designed to provide users with a seamless and secure chatting experience. With features like real-time updates, a full friendship system, protected routes, and Google authentication, the app combines performance, scalability, and usability into a production-ready solution. This project not only showcases the capabilities of modern web technologies but also serves as a practical blueprint for building scalable, real-time applications for real-world use.

## **2. BACKGROUND AND MOTIVATIONS**

The motivation behind this project is to explore and implement a real-world, production-ready chat application that showcases the power of modern web technologies in delivering fast, scalable, and interactive user experiences. With the increasing demand for real-time communication across platforms, this project aims to bridge that need by building a performant chat system using Next.js 13, React, and Upstash Redis. The goal is to demonstrate how a full-stack application can be architected to support instant messaging, secure user authentication, and dynamic user interactions—all while maintaining clean code, responsive design, and efficient backend operations. Through this project, the objective is to deepen practical understanding of real-time systems, serverless architecture, and end-to-end web development, ultimately achieving a complete, functional solution that meets modern user expectations.



### 3. PROBLEM STATEMENT

The problem this project addresses is the lack of lightweight, real-time chat applications that are both scalable and easy to maintain using modern web technologies. Traditional chat systems often struggle with latency, complex backend setups, or outdated UI designs. This project aims to solve these challenges by building a real-time chat app that delivers instant communication, secure user management, and a responsive interface—all powered by a serverless and highly performant stack using Next.js 13, Upstash Redis, and TypeScript.

### 4. OBJECTIVES AND RESEARCH METHODOLOGY

- **OBJECTIVES OF THE PROJECT**

- To develop a real-time chat application** that enables instant communication between users with minimal latency using modern web technologies.
- To implement a secure and scalable friendship system** where users can send, accept, or reject friend requests, managing their social connections effectively.
- To integrate Upstash Redis** as the primary database to achieve lightning- fast data access and real-time messaging capabilities through its Pub/Sub feature.
- To build a clean, responsive user interface** using TailwindCSS that provides a smooth experience across all devices and screen sizes.
- To ensure route protection and user privacy** by restricting access to sensitive parts of the application through secure authentication and authorization mechanisms.

- **RESEARCH METHODOLOGY**

- 1. Requirement Analysis:**

Gather information about user needs and expectations through observation, surveys, or interviews.

- 2. Technology Selection:**

Choose suitable tools and technologies like Next.js 13, Upstash Redis, React, and TypeScript based on performance, scalability, and ease of integration.

- 3. System Design:**

Create wireframes, flowcharts, and architecture diagrams to plan how the application will function and interact.

- 4. Implementation:**

Develop the application by writing code for the frontend, backend, database, and real-time features.

- 5. Testing:**

Perform unit testing, integration testing, and user testing to check for errors, bugs, and usability issues.

- **PROJECT ORGANIZATION**

The project is organized into several key modules and phases to ensure smooth development, testing, and deployment of the real-time chat application.

- 1. Frontend Development**

- Built using **React** and **Next.js 13 (App Router)**.

- Handles the user interface, including chat screens, login pages, and friend list views.
- Styled with **TailwindCSS** for responsiveness and modern design.

## 2. Backend & API Layer

- Uses **Next.js API routes and server actions** to handle data operations and route protection.
- Manages authentication, friend request logic, message sending/receiving, and session handling.

## 3. Real-Time Messaging System

- Powered by **Upstash Redis Pub/Sub** for instant message delivery.
- Enables users to chat in real-time with live message updates.

## 4. Authentication Module

- Implements **Google Authentication** using **NextAuth.js**.
- Manages user sessions securely and handles login/logout functionality.

## 5. Database Integration

- Uses **Upstash Redis** as the main database for storing messages, users, and friend data.
- Offers ultra-fast querying and supports real-time data communication.

## 6. Database Integration

- Uses **Upstash Redis** as the main database for storing messages, users, and friend data.
- Offers ultra-fast querying and supports real-time data communication.

## **7. Route Protection & Security**

- Restricts access to sensitive pages (e.g., chat dashboard) based on authentication status.
- Middleware and API-level checks are used to secure data and user actions.

## **8. Testing & Debugging**

- Manual and automated testing to ensure smooth user experience and bug-free performance.
- Checks include authentication flows, real-time chat reliability, and UI responsiveness.

## **9. Deployment**

- Application is deployed using **Vercel** for frontend and backend hosting.
- Upstash Redis handles the real-time backend as a fully managed service.

## **10. Documentation & Maintenance**

- All code and system architecture are well-documented.
- Instructions for installation, setup, and contribution are included to support future improvements.

## **HARDWARE AND SOFTWARE REQUIREMENTS**

### **1. HARDWARE REQUIREMENTS**

To develop, test, and run the real-time chat application effectively, the following hardware requirements are recommended:

<b>Component</b>	<b>Specification(Minimum)</b>
Processor	Intel Core i3 6th Gen or Equivalent (x64)
RAM	4 GB
Storage	500 MB of free disk space
Display	1024 x 768 resolution or higher
Input Devices	Keyboard and Mouse
Architecture	64-bit (x64) processor architecture

### **2. SOFTWARE REQUIREMENT**

To build, run, and deploy the full-stack real-time chat application, the following software tools and platforms are required:

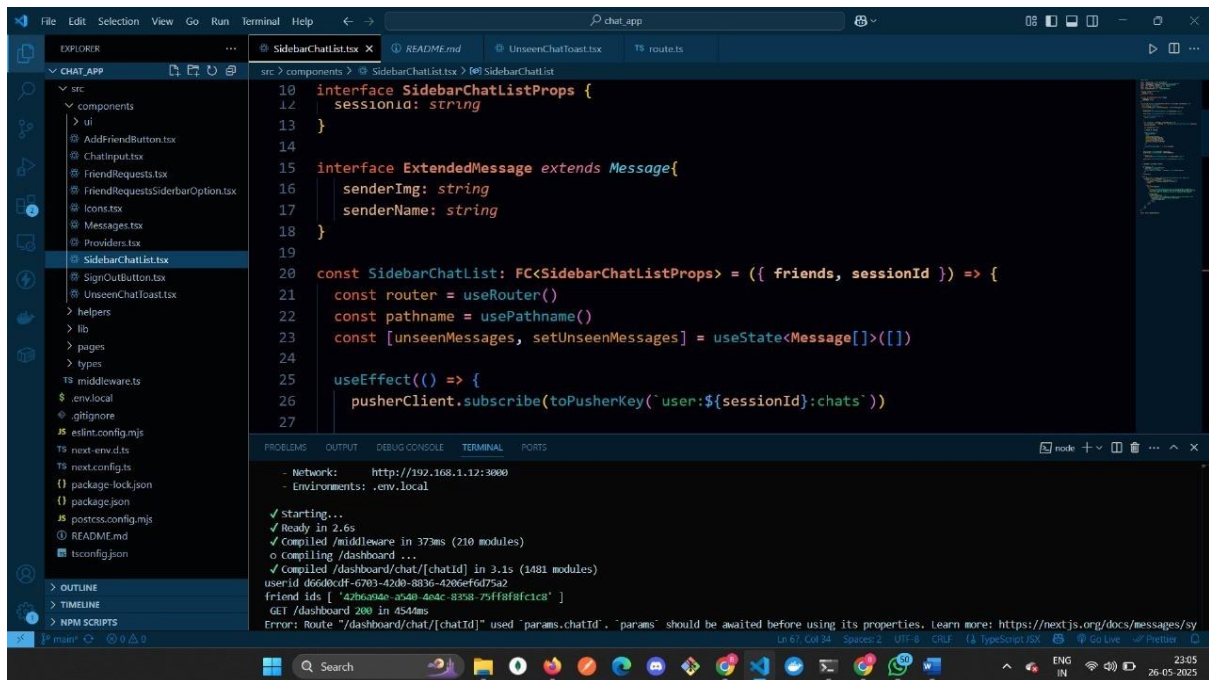
- **For Development**

- a. **Operating System:** Windows 10/11, macOS, or any Linux distribution (Ubuntu recommended)

- b. **Code Editor/IDE:** Visual Studio Code with useful extensions like Prettier, ESLint, and TailwindCSS IntelliSense
- c. **Runtime & Frameworks:** Node.js (v18 or above) – for running the Next.js application
- d. Next.js 13 (App Router) – for frontend and backend framework
- e. React – for building user interfaces
- f. TypeScript – for type-safe development
- g. TailwindCSS – for responsive and utility-first styling
- h. **Authentication:** NextAuth.js – for implementing Google Sign-In
- i. **Database & Real-Time Services:** Upstash Redis – managed, serverless Redis for storing data and handling real-time messaging
- j. **Browser:** Latest version of Google Chrome, Firefox, or Edge for development and testing
- **For Deployment**
  - a. **Hosting Platform:** Vercel – for deploying the Next.js app (frontend + backend)
  - b. **Environment Configuration:** .env.local file – for storing sensitive keys like Google OAuth credentials and Upstash Redis URL/token
  - c. These tools enabled efficient development , debugging, version tracking, and GUI rendering support for the custom shell project.



# CODE



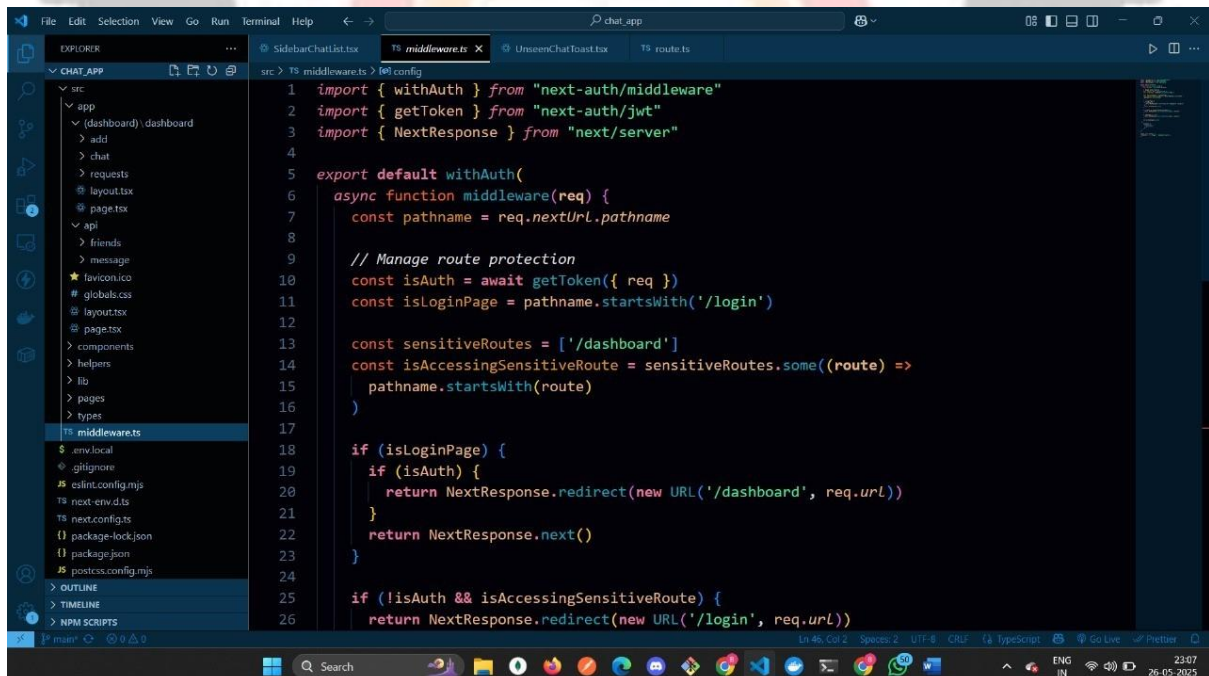
The screenshot shows the VS Code editor with the file `src > components > SidebarChatList.tsx` open. The file contains the following code:

```
10 interface SidebarChatListProps {
11   | sessionId: string
12 }
13
14
15 interface ExtendedMessage extends Message {
16   senderImg: string
17   senderName: string
18 }
19
20 const SidebarChatList: FC<SidebarChatListProps> = ({ friends, sessionId }) => {
21   const router = useRouter()
22   const pathname = usePathname()
23   const [unseenMessages, setUnseenMessages] = useState<Message[]>([])
24
25   useEffect(() => {
26     pusherClient.subscribe(toPusherKey(`user:${sessionId}:chats`))
27   })
28 }
```

The terminal output shows the following:

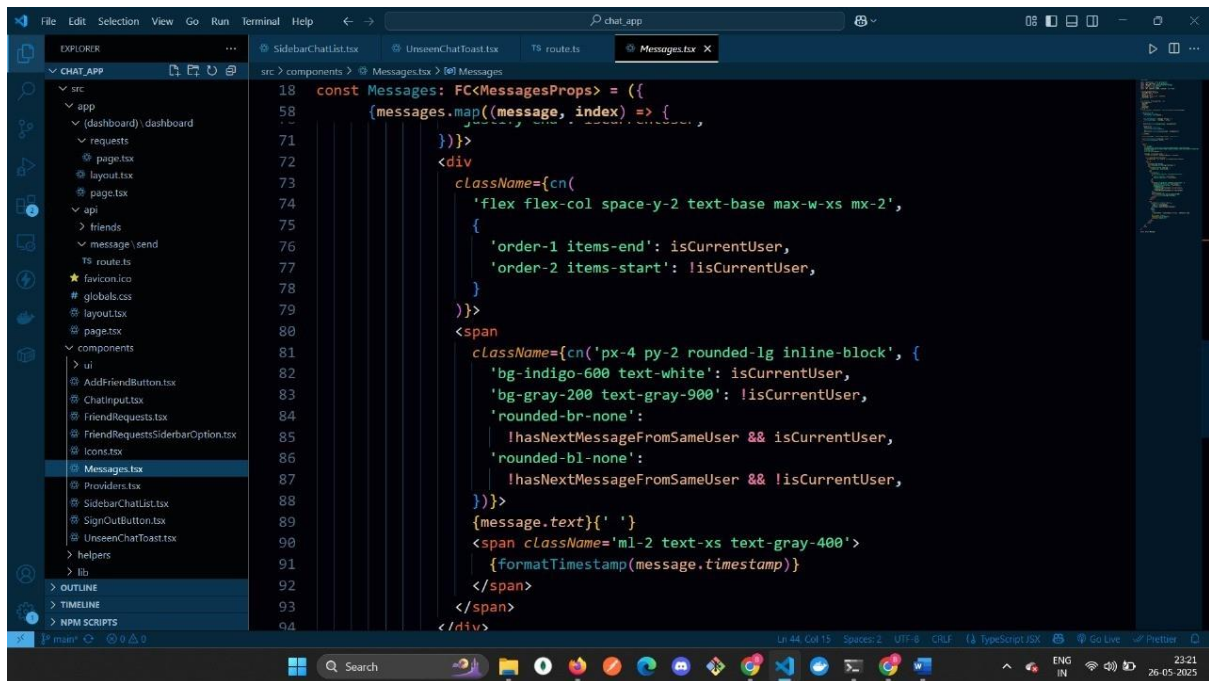
```
- Network: http://192.168.1.12:3000
- Environments: .env.local

Starting...
Ready in 2.6s
Compiled /middleware in 373ms (210 modules)
  o Compiling /dashboard ...
    ✓ Compiled /dashboard/chat/[chatId] in 3.1s (1481 modules)
    user id [ '60a0c01f-67b3-42d0-8836-4200ef6d75d2' ]
    friend id [ '42b0a9a-35d0-4a4c-833d-75f1a1f1c1c1' ]
    GET /dashboard/200 in 454ms
Error: Route "/dashboard/chat/[chatId]" used "params.chatId". "params" should be awaited before using its properties. Learn more: https://nextjs.org/docs/messages/sy
```

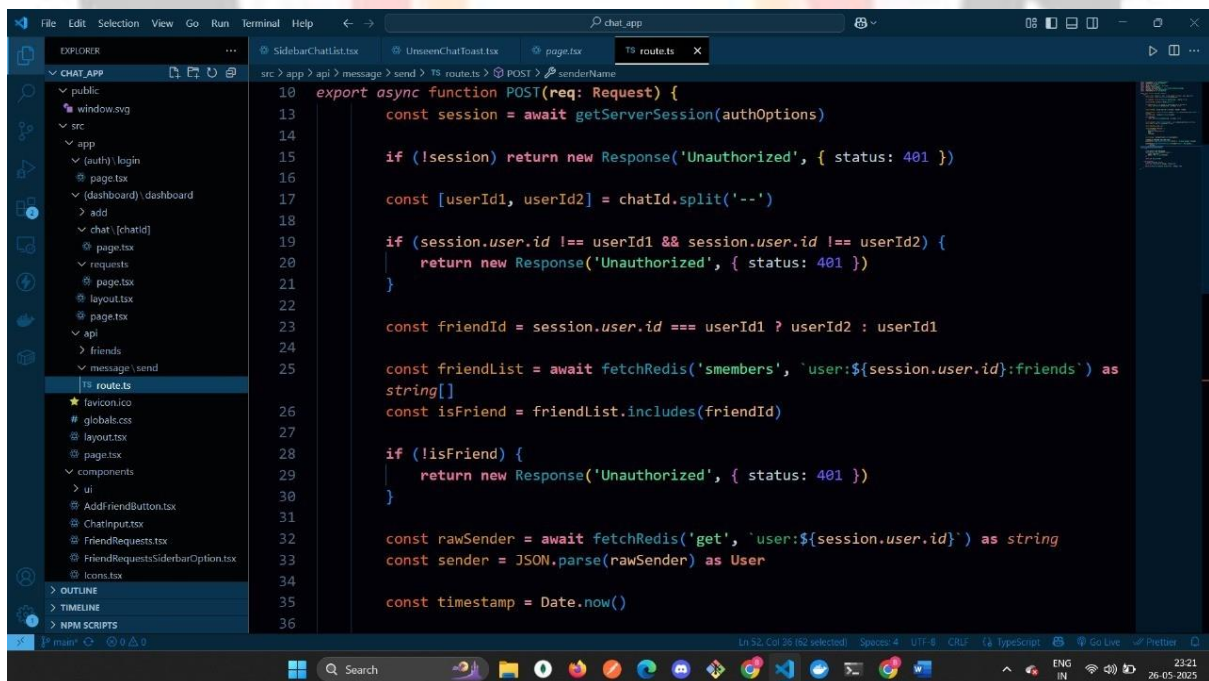


The screenshot shows the VS Code editor with the file `src > ts > middleware.ts` open. The file contains the following code:

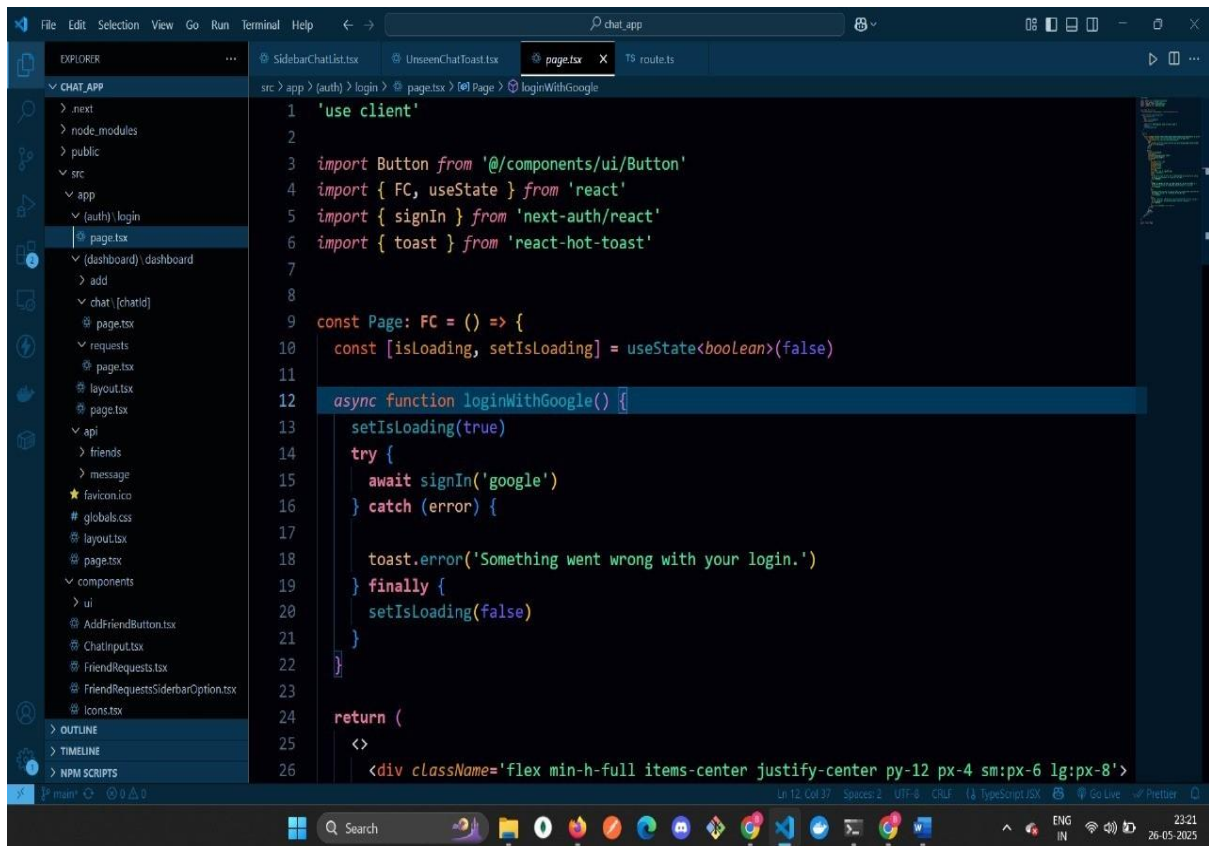
```
1 import { withAuth } from "next-auth/middleware"
2 import { getToken } from "next-auth/jwt"
3 import { NextResponse } from "next/server"
4
5 export default withAuth(
6   async function middleware(req) {
7     const pathname = req.nextURL.pathname
8
9     // Manage route protection
10    const isAuth = await getToken({ req })
11    const isLoginPage = pathname.startsWith('/login')
12
13    const sensitiveRoutes = ['/dashboard']
14    const isAccessingSensitiveRoute = sensitiveRoutes.some((route) =>
15      pathname.startsWith(route)
16    )
17
18    if (isLoginPage) {
19      if (isAuth) {
20        return NextResponse.redirect(new URL('/dashboard', req.url))
21      }
22      return NextResponse.next()
23    }
24
25    if (!isAuth && isAccessingSensitiveRoute) {
26      return NextResponse.redirect(new URL('/login', req.url))
27    }
28  }
29 )
```



```
18 const Messages: FC<MessagesProps> = ({
58   messages.map((message, index) => {
71   })))
72   <div
73     className={cn(
74       'flex flex-col space-y-2 text-base max-w-xs mx-2',
75       {
76         'order-1 items-end': isCurrentUser,
77         'order-2 items-start': !isCurrentUser,
78       }
79     )}>
80     <span
81       className={cn('px-4 py-2 rounded-lg inline-block', {
82         'bg-indigo-600 text-white': isCurrentUser,
83         'bg-gray-200 text-gray-900': !isCurrentUser,
84         'rounded-br-none':
85           !hasNextMessageFromSameUser && isCurrentUser,
86         'rounded-bl-none':
87           !hasNextMessageFromSameUser && !isCurrentUser,
88       })}>
89       {message.text}{' '}
90       <span className='ml-2 text-xs text-gray-400'>
91         {formatTimestamp(message.timestamp)}
92       </span>
93     </span>
94   </div>
```



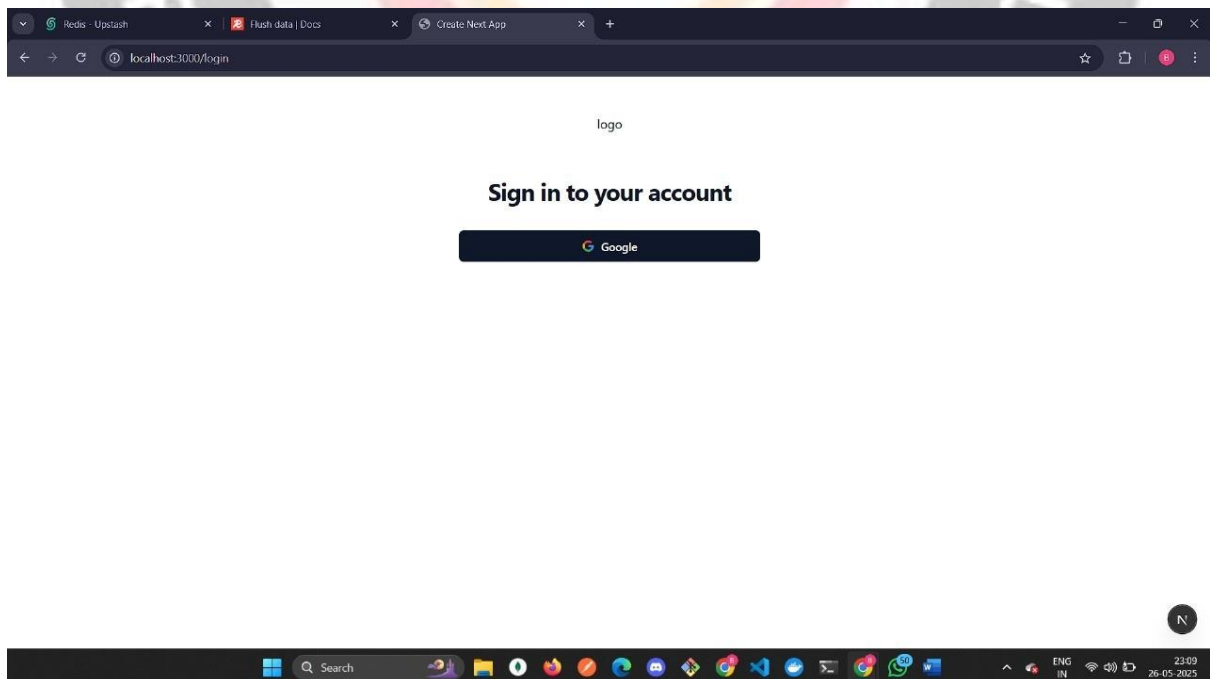
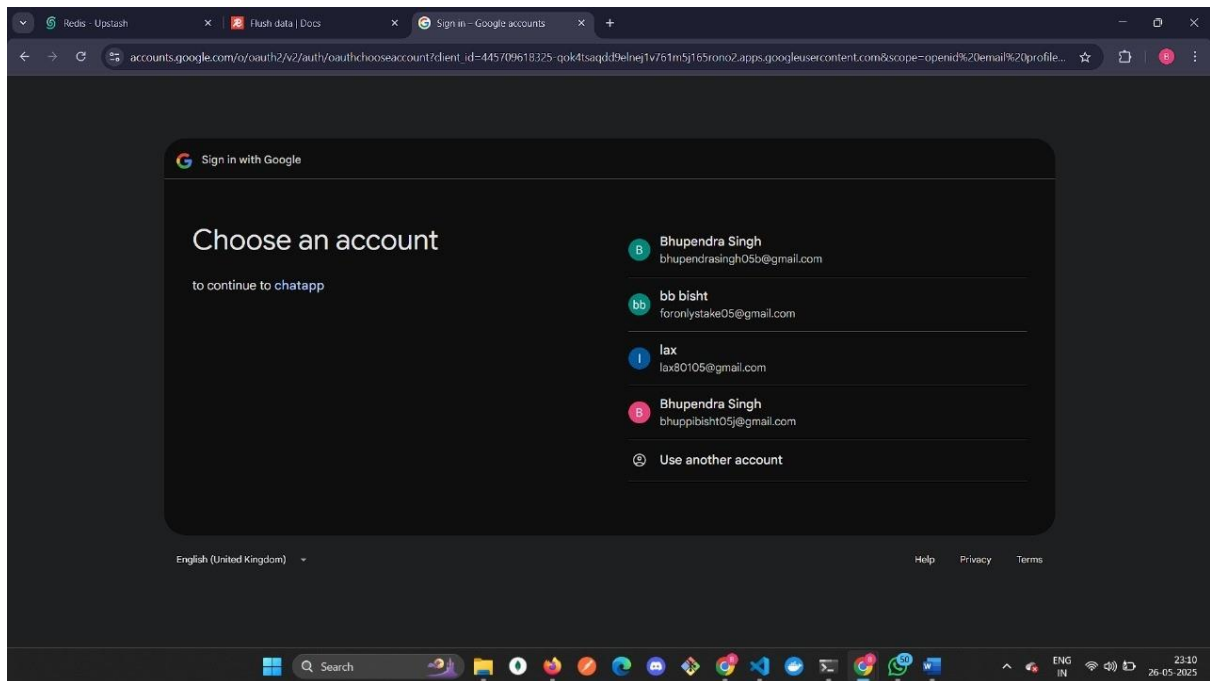
```
10 export async function POST(req: Request) {
13   const session = await getServerSession(authOptions)
14
15   if (!session) return new Response('Unauthorized', { status: 401 })
16
17   const [userId1, userId2] = chatId.split('--')
18
19   if (session.user.id !== userId1 && session.user.id !== userId2) {
20     return new Response('Unauthorized', { status: 401 })
21   }
22
23   const friendId = session.user.id === userId1 ? userId2 : userId1
24
25   const friendList = await fetchRedis('smembers', `user:${session.user.id}:friends`) as
26     string[]
27   const isFriend = friendList.includes(friendId)
28
29   if (!isFriend) {
30     return new Response('Unauthorized', { status: 401 })
31   }
32
33   const rawSender = await fetchRedis('get', `user:${session.user.id}`) as string
34   const sender = JSON.parse(rawSender) as User
35
36   const timestamp = Date.now()
```

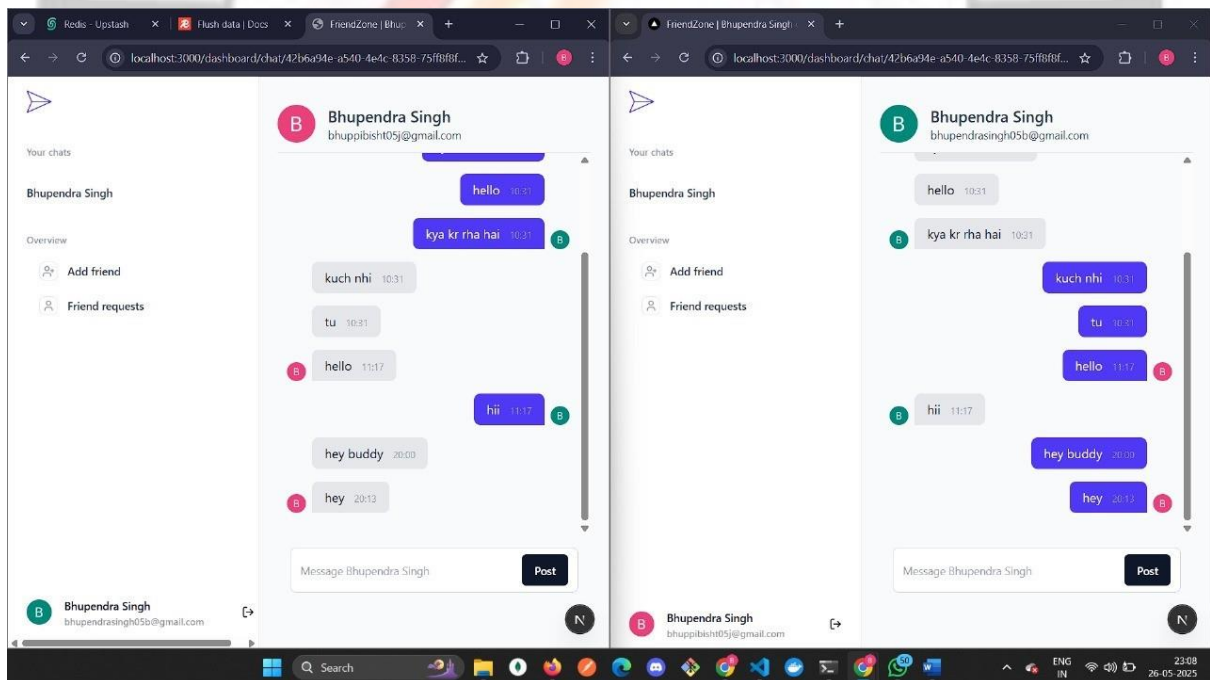
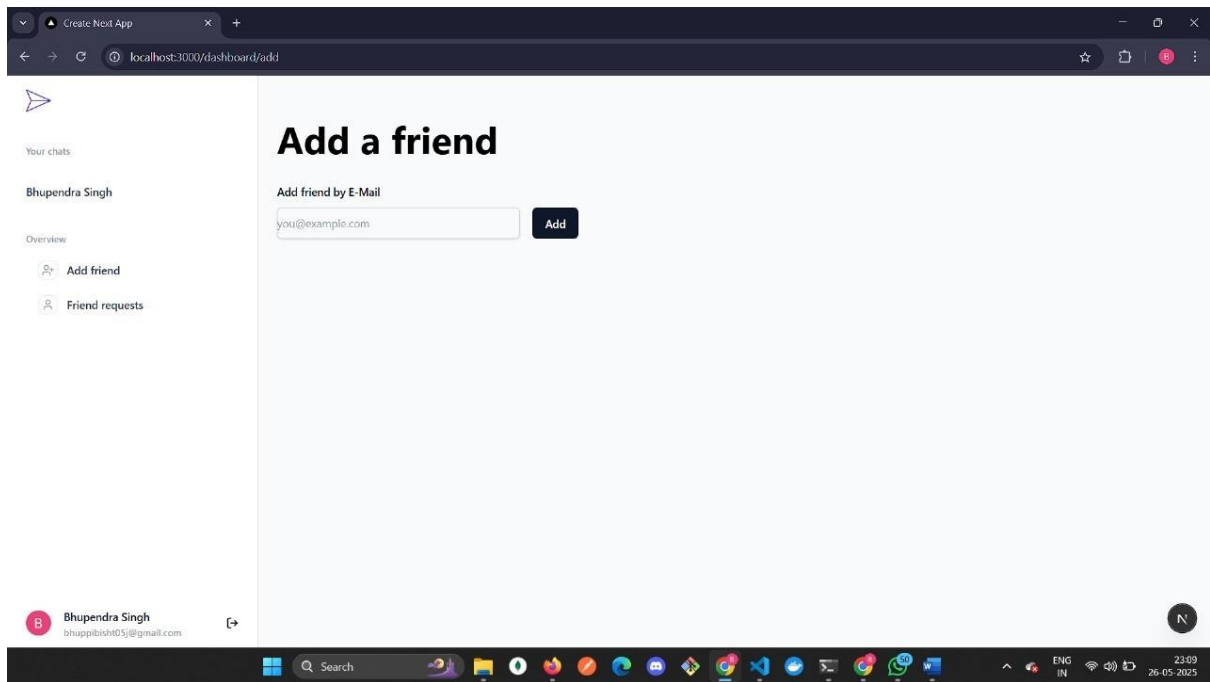


```
1 'use client'
2
3 import Button from '@components/ui/Button'
4 import { FC, useState } from 'react'
5 import { signIn } from 'next-auth/react'
6 import { toast } from 'react-hot-toast'
7
8
9 const Page: FC = () => {
10   const [isLoading, setIsLoading] = useState<boolean>(false)
11
12   async function loginWithGoogle() {
13     setIsLoading(true)
14     try {
15       await signIn('google')
16     } catch (error) {
17
18       toast.error('Something went wrong with your login.')
19     } finally {
20       setIsLoading(false)
21     }
22   }
23
24   return (
25     <>
26     <div className='flex min-h-full items-center justify-center py-12 px-4 sm:px-6 lg:px-8'>
```



# SNAPSHOTS







## **LIMITATIONS**

**Despite its strengths**, the project has some limitations. It currently supports only one-on-one messaging, with no group chat or media sharing features. The app relies on a stable internet connection and only offers Google Sign-In for authentication, limiting accessibility for users without Google accounts. Additionally, there is no offline support or admin panel for moderation, and messages are not end-to-end encrypted, which could raise privacy concerns. These limitations provide areas for future improvement and feature expansion.

### **1. LIMITED SCALABILITY FOR HIGH TRAFFIC**

While Upstash Redis supports real-time messaging efficiently, the free tier or lower plans may struggle under very high user loads without upgrading to a paid plan.

### **2. SINGLE CHAT TYPE (ONE-TO-ONE ONLY)**

The current system supports only one-on-one messaging. Group chats or channels are not implemented, limiting its use for teams or communities.

### **3. DEPENDENCY ON INTERNET CONNECTION**

Real-time messaging and authentication rely on a stable internet connection; poor connectivity may affect performance and message delivery.

### **4. LIMITED OFFLINE SUPPORT**

The application is web-based and doesn't support offline messaging or viewing, which could affect usability in low-connectivity areas.



## **5. NO FILE OR MEDIA SHARING**

The current version focuses on text messaging only. It does not support sending files, images, or voice messages.

## **6. AUTHENTICATION LIMITED TO GOOGLE**

The app currently uses only Google Sign-In for authentication. Users without a Google account cannot access the platform.

## **7. LACK OF ADMIN PANEL**

There is no admin dashboard for monitoring user activity, managing reports, or moderating content, which may be needed for large-scale use.

## **8. NO END-TO-END ENCRYPTION**

Messages are not end-to-end encrypted, which means they could be accessed on the server if compromised, raising potential privacy concerns.

## **ENHANCEMENTS**

**In the future**, the project can be enhanced by adding features such as group chats, media and file sharing, and multi-provider authentication options (like Facebook or email login). Implementing offline message storage and end-to-end encryption would improve reliability and user privacy. An admin dashboard could also be introduced for user moderation and analytics. These enhancements would make the application more robust, secure, and suitable for larger user bases or enterprise use.

### **1. Add Group Chat Functionality**

- Allow users to create and participate in group conversations.

### **2. Enable Media and File Sharing**

- Support sending images, documents, and audio files within chats.

### **3. Integrate Multiple Authentication Methods**

- Include options like email/password, Facebook, or GitHub login alongside Google.

### **4. Implement Offline Message Support**

- Let users read previous chats and queue messages even without internet connectivity.

### **5. Add End-to-End Encryption**

- Secure user messages by encrypting them from sender to receiver.

### **6. Develop an Admin Dashboard**

- Allow moderators to manage users, monitor activity, and handle reported content.

## CONCLUSION

**In conclusion**, this project successfully demonstrates the creation of a full-stack, real-time chat application that effectively combines modern web development technologies to meet the growing demand for instant, secure, and user-friendly communication platforms. By utilizing Next.js 13's powerful framework, React's dynamic interface capabilities, TypeScript's type safety, and Upstash Redis's blazing-fast in-memory data storage and Pub/Sub features, the app delivers seamless real-time messaging with minimal latency.

The integration of Google Authentication simplifies user login while enhancing security, and the friendship system adds an important social layer by allowing users to connect, manage friend requests, and control their chat networks. The use of TailwindCSS ensures the application is responsive and visually appealing across all devices. Route protection mechanisms safeguard user data and restrict access, maintaining privacy and security throughout the app.

While the current version focuses on one-on-one messaging without support for group chats, media sharing, or end-to-end encryption, it establishes a robust foundation with scalable architecture and maintainable codebase. This makes it easy to add more advanced features and improvements in the future. Moreover, the project highlights best practices in full-stack development, serverless architecture, and real-time data handling, providing valuable insights for developers looking to build scalable, production-ready applications.

Overall, this chat application not only meets its functional goals but also sets a roadmap for ongoing enhancements, making it well-suited for real-world deployment and adaptable to the evolving needs of its users.

## **REFERENCES**

1. Next.js Documentation – Official documentation for the Next.js framework covering routing, API routes, and deployment.
2. React Documentation – The official guide to building user interfaces with React.
3. TypeScript Handbook – Comprehensive resource for understanding TypeScript syntax and features.
4. Upstash Redis Documentation – Documentation for Upstash's serverless Redis service including Pub/Sub for real-time messaging.
5. NextAuth.js Documentation – Guide to implementing authentication in Next.js apps, including Google OAuth integration.
6. TailwindCSS Documentation – Utility-first CSS framework documentation for styling responsive user interfaces.