**A SYNOPSIS ON**



**A Real Time Chat Messaging App**



**Submitted in partial fulfilment of the requirement for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**Computer Science & Engineering**

**Submitted by:**

**Bhupendra singh** **2261145**

**Jitendra harbola 2261289**

**Himanshu singh** **2261272**

**Manish kumar dani 2361010**

***Under the Guidance of***

***Anubhav Bewerwal***

***Designation***

**Project Team ID:  18.**



**Department of Computer Science & Engineering**

**Graphic Era Hill University, Bhimtal, Uttarakhand**

**March-2025**



**CANDIDATE’S DECLARATION**

I/We hereby certify that the work which is being presented in the Synopsis entitled **“Realtime chat messaging app”** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the Graphic Era Hill University, Bhimtalcampus and shall be carried out by the undersigned under the supervision of Mr **Anubhav Bewerwal, Designation**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

**Bhupendra singh**  2261145

**Jitendra harbola**  2261289

**Himanshu singh** 2261272

**Manish kumar dani**  2361010

The above mentioned students shall be working under the supervision of the undersigned on the **“A Real Time Chat Messaging App”**

Signature Signature

**Supervisor** **Head of the Department**

**Internal Evaluation (By DPRC Committee)**

**Status of the Synopsis:**  Accepted / Rejected

**Any Comments:**

**Name of the Committee Members: Signature with Date**

1.

2.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Description** | **Page No.** |
| Chapter 1 | Introduction and Problem Statement | 1-3 |
| Chapter 2 | Background/ Literature Survey | 4-6 |
| Chapter 3 | Objectives | 7 |
| Chapter 4 | Hardware and Software Requirements | 8 |
| Chapter 5 | Possible Approach/ Algorithms | 9-11 |
|  | References | 12 |

**Chapter 1**

1. **Introduction**

In today's digital-first world, communication systems have evolved beyond traditional modes of interaction. With the increasing dependence on instant connectivity—whether through social networks, collaborative platforms, or customer support systems—the demand for real-time messaging applications has grown rapidly. Users expect immediate delivery, live updates, seamless interface responsiveness, and uninterrupted interaction across devices.

This project is centered on the development of a **full-stack real-time chat messaging application**. The application uses modern technologies like **Next.js 13**, **Upstash Redis**, **TypeScript**, **TailwindCSS**, and **Google Authentication** to provide a secure, responsive, and scalable user experience. The goal is to simulate a real-world messaging system that can be deployed and used by actual users, offering features similar to commercial chat applications but built entirely from scratch.

### Background and Motivation

While several commercial solutions exist—such as WhatsApp, Slack, and Discord—these platforms often function as closed ecosystems with limited customization. Developers and startups frequently require flexible, lightweight messaging solutions that can be tailored to specific user needs. This project was conceived to fill that gap by demonstrating how a robust real-time chat system can be built using an open, modern tech stack.

The use of **Next.js 13** ensures that both client and server-side components can coexist seamlessly in one codebase, simplifying development and deployment. **Upstash Redis** brings high-speed real-time data handling using a publish-subscribe model, eliminating the need for manual polling. **TypeScript** improves code maintainability and reduces runtime errors through static type checking. **TailwindCSS** is used to design a modern and responsive user interface, while **Google Authentication** secures the application and simplifies user onboarding.

### Key Features of the Application

This chat messaging application offers a combination of functionality and performance, aimed at creating a user-centric experience. The key features include:

* **Real-Time Messaging:** Messages are delivered instantly using Redis’s pub/sub architecture, allowing smooth and continuous conversation between users.
* **Friendship Management System:** Users can send friend requests, accept or deny them, and maintain a list of approved contacts.
* **Authentication with Google:** Secure login and user identification are handled via Google OAuth, eliminating the need to manage passwords.
* **Protected Routes:** Pages like the chat dashboard or user settings are accessible only to authenticated users, ensuring privacy and data protection.
* **Responsive and Modern UI:** Designed with TailwindCSS, the user interface adapts well to all screen sizes and provides a fluid experience on both mobile and desktop platforms.
* **Serverless and Scalable Backend:** Upstash Redis, being serverless and low-latency, allows high performance and scalability with minimal server management.

### Relevance in the Modern Web Ecosystem

The web development landscape is continuously shifting toward real-time, reactive applications. From collaborative tools like Google Docs to streaming platforms and real-time dashboards, responsiveness and interactivity are paramount. Chat systems represent a critical use case for these technologies, integrating many elements of full-stack engineering—databases, authentication, UI design, and performance optimization.

This project also aligns well with current trends in full-stack development, such as:

* **Type-safe development** using TypeScript.
* **Server components and app routing** in Next.js 13.
* **Cloud-based databases** like Upstash that reduce infrastructure complexity.
* **Component-based design** and reusable UI patterns using TailwindCSS.

1. **Problem Statement**

The core problem addressed in this project is the development of a real-time messaging application that ensures high performance, responsiveness, and scalability—while also being easy to develop, deploy, and extend. Building such an application from scratch presents several challenges that must be addressed through both architectural design and technical implementation.

### General Problem

Traditional web applications often rely on request-response cycles that do not suit the demands of real-time interaction. Polling mechanisms can be inefficient and lead to delayed communication. Moreover, building a full-stack application that supports dynamic user interactions, manages multiple concurrent sessions, and secures private routes is a complex task that requires careful planning and execution.

### Specific Problem

The specific challenge is to build a **full-featured, full-stack chat messaging app** using modern tools and practices, with the following criteria:

* Enable **real-time, bi-directional communication** between authenticated users.
* Maintain **user relationships** through a friendship system.
* Ensure **data consistency** and instant synchronization across client sessions.
* Provide a **secure authentication mechanism** using OAuth (Google).
* Protect sensitive routes from unauthorized access.
* Maintain a clean and maintainable **component-based frontend**.
* Optimize for performance and **low-latency interactions**.

**"To design and implement a full stack real-time chat messaging application using Next.js 13, Upstash Redis, TypeScript, and TailwindCSS that allows users to communicate instantly, manage friendships, authenticate via Google, and securely access protected chat features in a performant, responsive, and scalable environment."**

This project will demonstrate the implementation of:

* Real-time message delivery using Redis pub/sub mechanisms.
* Friend management features for controlled interactions.
* Server-side and client-side integration using Next.js.
* Route protection and session handling for logged-in users.
* UI responsiveness with optimized state management.

**Chapter 2**

**Background/ Literature Survey**

In the present times, research work is actively ongoing in the domain of **real-time web communication**, **scalable messaging systems**, and **full-stack web development**. The evolution of technologies such as WebSockets, serverless databases, and cloud-native authentication mechanisms has led to a new wave of applications focused on instant and efficient data transmission. Real-time messaging platforms have become an integral part of daily communication in both personal and professional contexts.

This chapter reviews key technologies, existing tools, and previous works that are relevant to the development of a real-time chat messaging application. The review focuses on three major areas: (1) real-time communication technologies, (2) frontend and backend frameworks for web development, and (3) authentication and access control mechanisms.

### Real-Time Communication Technologies

One of the core challenges in real-time communication is minimizing the latency between message send and message receive events. Early solutions often relied on HTTP polling or long polling, which were resource-intensive and inefficient. With the introduction of **WebSockets**—a full-duplex communication channel over a single TCP connection—real-time communication became more practical and scalable for web applications.

**Socket.IO**, a popular JavaScript library built on top of WebSockets, further abstracts these complexities and provides fallbacks for environments where WebSockets are not supported. It has been widely used in real-time applications like chat systems, online games, and collaborative editing platforms.

Recent advances have led to the adoption of **serverless and event-driven models**, such as **Redis Pub/Sub**, which allow messages to be broadcast instantly to all subscribers. According to research by Zhang et al. (2021), Redis Pub/Sub can support thousands of concurrent subscribers with sub-millisecond latency, making it ideal for building highly responsive chat applications.

### Databases for Real-Time Applications

Traditional relational databases are often not suitable for real-time messaging due to their higher query latency and rigid schema structure. In contrast, **in-memory databases** such as **Redis** provide exceptional speed and flexibility. Redis supports advanced data structures like hashes, lists, and sets, and is optimized for high-throughput operations.

**Upstash Redis** brings the power of Redis into the serverless environment, allowing developers to use it without managing infrastructure. Its pay-per-request pricing model and native support for Pub/Sub make it ideal for scalable real-time apps. Studies by Upstash engineers (2022) show that their platform can handle millions of events per minute with consistent latency, thanks to their globally distributed architecture.

### Full Stack Web Development with Next.js

**Next.js** is a React-based framework that supports both static site generation and server-side rendering. With the release of **Next.js 13**, the framework introduced the **App Router**, **Server Components**, and **Edge Middleware**, allowing more flexibility and performance in full-stack development.

Researchers and developers alike have embraced Next.js for its ease of integration with APIs, flexible routing, and seamless deployment with platforms like Vercel. In particular, its hybrid rendering capabilities make it suitable for applications that require both real-time interactivity and SEO friendliness.

In the context of chat applications, Next.js enables tight integration between the client and server, allowing efficient API handling, secure authentication, and rapid page loading. By combining **Client Components** (for real-time interactivity) and **Server Components** (for performance and logic handling), developers can achieve an optimal balance of speed and functionality.

### Authentication and Security

Authentication is a critical component of any chat application, ensuring that only authorized users can access specific features or data. Among the many methods available, **OAuth 2.0** and **OpenID Connect** have emerged as the standards for secure, federated login systems.

**Google Authentication**, built on OAuth 2.0, provides a secure and user-friendly way to sign in users without managing passwords directly. According to Google’s security whitepapers, using OAuth reduces the risk of credential theft and streamlines user onboarding across platforms.

Modern frameworks like Next.js make it easy to integrate OAuth through libraries such as **NextAuth.js**, which supports multiple providers and protects routes through middleware and session management.

In this project, route protection ensures that users cannot access certain parts of the application (e.g., the chat dashboard or friend requests) unless they are properly authenticated. This aligns with best practices in modern web security and data privacy.

### Existing Systems and Limitations

Several popular chat applications have been studied to understand their architecture and user experience:

* **Slack** uses a hybrid system of WebSockets and HTTP APIs, allowing users to send messages, upload files, and receive notifications in real time.
* **WhatsApp Web** operates using end-to-end encrypted WebSocket connections, syncing with the mobile device.
* **Discord** offers a rich API and uses a microservices-based architecture with Redis for session management.

While these applications are powerful, they often come with limitations:

* Lack of customization for third-party developers.
* Heavy infrastructure requirements.
* Complex codebases that are difficult to study or replicate.

This project addresses these gaps by providing a simplified yet effective real-time chat solution using open tools that are accessible to developers and scalable in real-world use.

### Conclusion of Literature Survey

The development of real-time messaging systems has benefited significantly from advancements in web communication protocols, in-memory data storage, serverless computing, and full-stack frameworks. Through the integration of **Next.js 13**, **Upstash Redis**, **TypeScript**, and **Google Authentication**, this project aims to combine the best of these technologies into a compact, efficient, and fully functional real-time chat application.

This literature survey provides a solid foundation for the technical choices made during the development of the system. The next chapter will present the **Objectives and Scope** of the project, defining what the system aims to achieve and what constraints it operates under.

**Chapter 3**

**Objectives**

The proposed work aims to build a full-stack real-time chat messaging application that delivers secure, scalable, and instant communication for users. This system will be designed using modern web technologies with a focus on performance, user experience, and ease of deployment. The key objectives are as follows:

1. **To develop a responsive and interactive real-time chat application** using **Next.js 13** and **React**, providing a smooth and modern user interface. **TailwindCSS** will be used to ensure responsiveness across devices and consistency in design.
2. **To implement instant messaging functionality using Upstash Redis** with the **Publish/Subscribe (Pub/Sub)** model, allowing users to exchange messages in real time with minimal delay, even under high user load.
3. **To build a complete friendship management system**, where users can send, accept, or reject friend requests. This will enable private and secure one-on-one chats while maintaining proper access control and relationship tracking.
4. **To integrate secure user authentication using Google OAuth**, ensuring users can sign in safely using their existing Google accounts. Sensitive routes will be protected using server-side middleware to prevent unauthorized access.
5. **To deploy the application on a cloud platform like Vercel**, ensuring scalability, global accessibility, and high performance through edge deployment and serverless architecture.

These objectives collectively aim to deliver a robust, real-world-ready application that showcases the power of full-stack development with real-time capabilities.

**Chapter 4**

**Hardware and Software Requirements**

4.1 Hardware Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No** | **Name of the Hardware** | **Specification** |  |
| 1 | Processor | Intel Core i5 / AMD Ryzen 5 or higher |  |
| 2 | RAM | Minimum 8 GB (16 GB recommended) |  |
| 3 | Hard Disk / SSD | Minimum 256 GB SSD |  |
| 4 | Internet Connectivity | Stable connection with minimum 10 Mbps |  |
| 5 | Display | 13" or larger with 1080p resolution |  |

4.2 Software Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No** | **Name of the Hardware** | **Specification** |  |
| 1 | Operating System | Windows 10/11, macOS, or Linux (Ubuntu 20.04 or higher) |  |
| 2 | Code Editor | Visual Studio Code (with necessary extensions) |  |
| 3 | Programming Languages | JavaScript, TypeScript |  |
| 4 | Database | Upstash Redis(cloud-based) |  |
| 5 | Authentication Library | NextAuth.js(Google OAuth integration) |  |

**Chapter 5**

**Possible Approach/ Algorithms**

In this chapter, we describe the methodologies and system-level approaches used to implement the proposed real-time chat messaging application. The overall solution integrates multiple components—frontend UI, backend APIs, real-time message delivery, user authentication, and a database for state management. We adopt a modular and scalable architecture using modern web technologies such as **Next.js 13**, **Upstash Redis**, **TypeScript**, and **Google OAuth**.

### **Overall Approach**

The application is divided into the following functional layers:

* **Client Layer (Frontend):** Built using React and TailwindCSS within the Next.js 13 framework, this layer renders chat UI, authentication pages, and friend request workflows.
* **Server Layer (API & Logic):** Next.js API routes and server components handle authentication, friend request processing, and Redis integration.
* **Database Layer:** Upstash Redis is used as the primary data store. It supports Pub/Sub for real-time messaging and stores user session data, friend relationships, and message history.
* **Authentication Layer:** Google OAuth via NextAuth.js is used for secure and reliable login functionality.

### **Real-Time Messaging using Redis Pub/Sub**

To achieve real-time communication, the Publish/Subscribe model from Redis is used. When a user sends a message, the backend publishes the message to a specific channel. All users subscribed to that channel receive the message instantly.

#### **Pseudocode: Real-Time Messaging Flow**

Input: Sender ID, Receiver ID, Message Text

Begin

1. Authenticate sender via session token.

2. Format the message with metadata (timestamp, senderId, text).

3. Save the message in Redis list under chat:{senderId}:{receiverId}

4. Publish the message to Redis channel chat:{receiverId}

5. Frontend listener (client socket) receives message from channel

6. Display message on recipient's screen in real-time.

End

### **Friendship System Algorithm**

The application includes a friendship model that handles friend requests and relationships.

#### **Friendship Workflow**

1. User A sends a friend request to User B.
2. User B accepts or rejects the request.
3. If accepted, both users are added to each other’s friend list.
4. Only friends can initiate chats.

#### **Pseudocode: Friend Request Handling**

Input: senderId, receiverEmail

Begin

1. Fetch receiverId using email from Redis

2. Check if senderId already in receiver’s friend list

3. If not:

a. Add senderId to receiver's pendingRequests list

b. Notify receiver about new friend request

4. Await response (accept/reject)

5. If accepted:

a. Move senderId from pending to accepted list

b. Add receiverId to sender's accepted list

6. Else, remove request from pending

End

### **Google Authentication via NextAuth.js**

The authentication flow is handled using NextAuth.js with the Google provider. It allows users to log in securely and maintains user sessions on both client and server sides.

#### **Authentication Steps**

1. User clicks "Sign in with Google".
2. Google OAuth verifies identity and returns user profile.
3. NextAuth creates a session token and stores it.
4. On subsequent requests, the session is validated.

### **Protected Routes with Middleware**

To restrict access to sensitive pages (e.g., chat dashboard, friend requests), middleware is used to check for authenticated sessions before rendering protected routes.

#### **Pseudocode: Route Protection**

Middleware(request):

1. Check session token in cookies

2. If session is valid, allow access

3. Else, redirect to login page

### **Deployment Approach**

The final application is deployed on **Vercel**, which supports serverless functions, edge networks, and fast CI/CD integration. Redis is accessed via its Upstash cloud dashboard.

### **Summary of Algorithms Used**

| **Feature** | **Algorithm / Method Used** |
| --- | --- |
| Real-Time Messaging | Redis Pub/Sub |
| Friend Request Management | Custom logic via Redis + API routes |
| Authentication | OAuth 2.0 via Google (NextAuth.js) |
| Route Protection | Middleware-based access control |
| Deployment | CI/CD via Vercel + Redis Cloud |

**References**

[1] J. Moroney, “Building Real-Time Chat Applications with Redis Pub/Sub,” International Journal of Web Applications, vol. 11, no. 3, pp. 145–152, June 2020.  
(Example: Journal Paper)

[2] A. Hajhosseini and N. Jafari Navimipour, “A survey on technical characteristics of the modern web frameworks,” Proc. 24th Int. Conf. on Computer and Information Technology, December 2021, pp. 215–220.  
(Example: Conference Paper)

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, 3rd ed., The MIT Press, McGraw-Hill Book Company, 2009.  
(Example: Textbook)

[4] Upstash Documentation [Online]. Accessed on 2nd April 2025 [5] Vercel Documentation – Next.js Deployment Guide [Online]. Accessed on 2nd April 2025:

[6] NextAuth.js Authentication Library [Online]. Accessed on 2nd April 2025 [7] Tailwind CSS Official Docs [Online]. Accessed on 2nd April 2025: [8] React Documentation – Official React Guide [Online]. Accessed on 2nd April 2025

[9] Josh Tried Coding, Build and Deploy a Full Stack Realtime Chat App with Next.js 13, Redis, Google Auth, YouTube, 20 Aug. 2023. [Online]. Available: