PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

6th Semester Project Report on

REAL-TIME CHAT SYSTEM

Submitted by

ALAN JOSEPH
PES2201800436

Jan – May, 2021

Under the Guidance of
Prof. Lavisha

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER APPLICATIONS
PROGRAM: BACHELOR OF COMPUTER APPLICATIONS

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER APPLICATIONS
PROGRAM – BACHELOR OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project entitled

REAL-TIME CHAT SYSTEM

is a bonafide work carried out by

ALAN JOSEPH(PES2201800436)

in partial fulfillment for the completion of 6th semester project work in the Program of Study BCA under rules and regulations of PES University, Bengaluru during the period Jan. 2021 – May 2021. The project report has been approved as it satisfies the 6th semester academic requirements in respect of project work.

Guide ***Chairperson***

Name and Signature of Examiners:

Examiner 1: *Examiner 2:* *Examiner 3:*

DECLARATION

I, **ALAN JOSEPH,** hereby declare that the project entitled, **"REAL-TIME CHAT SYSTEM "** is an original work done by us under the guidance of **Prof. Lavisha,** and is being submitted in partial fulfillment of the requirements for completion of 6th Semester course work in the Program of Study BCA. All corrections/suggestions indicated for internal assessment have been incorporated in the report**.**

Place:

Date:

ALAN JOSEPH

ACKNOWLEDGEMENT

I am personally thankful to my university for giving me the opportunity to do my project work. It has given me exposure and great knowledge.

I would like to express my special thanks of gratitude to my project guide **Prof. Lavisha** for her guidance and support in completing my project. I am very thankful for the motivation provided and her consent which lead to better design and implementation of my project.

I am also grateful to the Chairperson Prof. Dilip Kumar Maripuri and all professors for providing me the required information whenever I needed.

I would finally thank my family and friends who have constantly supported me in completion of my project in a better way.

ABSTRACT

This project aims to provide an instant messaging platform to users, with a responsive real-time interface. It aims to help users to communicate or interact with each other instantly with minimal delay. We also aim at providing all the common or rudimentary features available in commonly used messaging services or chat platforms.

Help or assistance is available to the users from the administrator who helps users with any issues they might face or approves actions performed by users and is overall responsible for managing the platform.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter-1

Introduction

A chat platform enables users to communicate with each other, or participate in discussions involving multiple users related to a particular topic. To manage the platform the admin account is responsible for creating new topics for users to participate in, approve the user requests to join various existing topics, deleting topics or banning users and has access to the Admin Logs. Various features are available to the regular user upon successful registration of an account, such as discovering new topics of discussion, finding friends to message, updating user profile image, notifications about actions/activities relevant to the user's interests, leaving topics, unfriend users, reporting users or topics to the admin.

Chapter-2

Background Study

2.1: Preliminary Investigation/Literature Survey

Slack is a widely used communication/ collaboration platform primarily used for business purposes, it has a responsive interface and updates the messages instantly. It provides features like allowing users to create groups where multiple users can participate together in discussions, direct messages to other users and sharing of files as well as indicating the online presence of other users.[1]

Telegram is another popular messaging service which operates real-time and has other features common with Slack. Both services provide certain privileges to an admin account who is in charge of maintaining the service as well as providing the required help to the users.[2]

Since the features of both services are quite intuitive to the average user, we will implement the common functionalities of such services to our project.

2.2: Technology Used

For the front-end we use HTML, CSS and the JavaScript Libraries React and Redux.

HTML (Hyper-text Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. CSS is the languages we use to style an HTML document.CSS describes how HTML elements should be displayed or appear in the webpage.

JavaScript is a scripting language used to create and control dynamic website content; it's one of the core technologies of web development and can be used on both the front-end and the back-end. React is a JavaScript library for building user interfaces; it is used to build single page applications. React allows us to create reusable UI components. Redux is an open-source JavaScript library used to manage application state.

2.3: Database used

Firebase Real-Time Database has been utilized for this project, it is used since it matches the requirement of providing a real-time experience for instant querying, creation, updating or deletion of data.

The Firebase Realtime Database is a cloud-hosted database, the data is synchronized in real-time to every connected client. All connected clients share one Realtime Database instances and automatically receive updates with the newest data.

Chapter-3

Project Overview

3.1: High level description of the project

Users are required to register a new account or login with an existing account to access the chat service, a username, e-mail and password are requested from the user for registration.

Upon successful login, users are able to view the topics previously joined or users they have befriended, discover new topics to participate or friends to engage in conversation with. The admin has to approve the join requests from the users for the user to be able to join the topic or the users will have to accept the friend requests received, for them to talk privately.

Various other common chat functionalities are provided to the user, such as setting a user avatar, uploading certain image formats along with the chat messages, searching the chat message history, further the users can view if their friends are online or if other users are currently typing a message.

The admin plays a central role in the maintenance and successful running of the chat system. Aside from approving topic join requests, they receive the user reports, ban users, delete uploaded images, or remove topics if violation/objectionable content is found.

3.2: System Requirements

3.2.1: Hardware Requirements

- Processor-Intel Pentium 4 processor
- Speed-133 MHZ (min)
- RAM-512 MB
- Hard Disc Capacity- 512 MB Minimum
- Monitor-VGA/SVAG colour monitor
- Keyboard-104 key
- Mouse- 2 buttons/3 buttons

3.2.2: Software Requirements

- Operating System- Linux/ Windows 7& Later Windows
- Software dependencies: Node.js, npm
- Web Browser (Chrome, Firefox, any modern popular web browser)
- Web Server: Apache
- Back End – Firebase

Chapter-4

Analysis

4.1: Problem Statement

4.1.1: Existing System

There may be many issues arising from the delay in communication, lack of awareness about the activities or actions undertaken by other users. The users are not notified about some of the actions and hence it may result in confusion or lack of updated knowledge.

However, the biggest issue seems to be the lack of active timely participation or of the admin/ staff to help the users with the problems they face, often there is a late response to user concerns or no action is taken against the reported offenders.

4.1.2: Proposed System

The new system focuses on reducing delays as much as possible between the users, every new message updates instantly along with notifying the time the message was sent, or displaying to users when other users are typing out a message, providing notifications about the activities taken by themselves, other users or the admin that may be of relevant interest to the user.

The user is also given the ability to report directly to the admin about issues faced either in the topic discussions or private messages. The admin is expected to act instantly upon receiving the reports.

4.2: Functional Requirements

Function	Description
FR01: User Registration	Provides a signup page for new users to create an account.
FR02: User Login	This form exists for users who already possess an account, the form also validates and loads to a different page if the user is recognized as an admin.
FR03: Discover Topics	This module helps users to find groups where multiple users can discuss topics.
FR04: Find Friends	This module helps users to find new friends to privately message outside of groups.
FR05: Report to Admin	This module enables users to report a topic or another user directly to the admin.
FR06: Chat Interface	The main chat body which contains all the sent and received messages.
FR07: Notifications	Module that contains received notifications
FR08: Leave Topic	This module allows users to leave a topic which they have previously joined.
FR09: Unfriend User	Module which allows users to unfriend the users present in their friends list.
FR10: Delete Topic	Module to facilitate deletion of an existing topic by the admin.

FR11: Ban User	Module to enable the user to punish an offending user permanently through a ban.
FR12: Admin Logs	Module which stores the admin logs. It contains log of certain admin or user actions along with user submitted reports.
FR13: Sign-out	This module provides logout functionality to both admin and users.

Table 4.1: Functional Requirement

4.3: Module Description

4.3.1: User Registration

The chat service requires users to register an account before being able to access any other functional features.

It requires a Username, email and password confirmation from the users.

4.3.2: User Login

After successful registration of an account, the user is logged in to the service where they can access the main interface and utilize the various features. The login not only validates the user account, but checks if the particular account is designated as an admin and loads the respective Admin interface accordingly.

4.3.3: Discover Topics

In this module, the user can view or search for the available topics by name and are able to view the list of topics along with its associated description. Upon trying to join a new topic, a request is sent to the admin for approval or rejection, allowing the user to subsequently participate in the topic.

4.3.4: Find Friends

In this module, functionality is provided to the users to look for other users and send them a friend request, for accepting or declining. Once a user accepts a friend request, both users are added into each other's friend list which then enables them to message each other privately.

4.3.5: Report to Admin

In the scenario where a user has faced an issue regarding a specific user or from a specific topic. The user is able to write a report or complaint about the incident, which asks for the type of incident and for a summarization of what occurred. The reports are delivered to the admin,

4.3.6: Chat Interface

This module contains all the sent and received messages between users either in topics or private messages. Users can send either text messages, emotes or upload image files. The message sent is recorded and displayed along with certain properties like the username, user avatar and the time the message was sent.

4.3.7: Notifications

It contains a log of actions or activity taken by the users, or admin that may be of relevant to the currently authenticated user along the time that the incident had occurred. It displays the notification count and is cleared once the notifications have been read by the user.

4.3.8: Leave Topic

It allows the regular users to quit the topics which they had previously joined with the approval of the admin. Users can leave the topics if they are disinterested or no longer want to participate in discussions related to the topic.

4.3.9: Unfriend User

It allows the user to unfriend users present in their friends list if either they no longer want to privately message the user or want to stop receiving the private messages from them.

4.3.10: Delete Topic

It is an admin module which enables the administrators to permanently delete a particular existing topic and hence, all the messages or data associated with that topic.

4.3.11: Ban User

The admin is able to complete ban a user account from the service when they are deemed to have committed a violation. Upon deletion all user data associated with the banned account is removed.

4.3.12 Admin Logs

The module operates in similar way to the user notifications; it logs the user actions, user- submitted reports or admin actions along with the time of the action.

4.3.13 Sign Out

The module allows the currently signed user or admin to quit the chat service's currently authenticated browser session, preparing it for a different login or re-authentication.

4.4: Non-Functional Requirements

4.4.1: Responsive Interface

The service provides a responsive modern interface similar to those implemented in chat services, once logged in all the activity of platform occurs and is contained within a single page web application.

4.4.2: Real-Time Information

All the data creation, updating, querying and deletion happens instantly from the database and the resultant output is accordingly displayed on the webpage.

4.4.3: Intuitive Design

The design is very user-friendly, and should be familiar to the average user who has utilized similar services in the past. New users should be able to quickly become familiarized with the basic usage.

4.4.4: User choice-driven

The user is given liberty to choose the topics to join which are of interest to them, the users they want to befriend, they can also leave the topics or unfriend the users as and when they please. Support is given to the user to file complaints or provide feedback to the admin.

4.4.5: Safety

The registration & login are well secured, further the admin acts as a safeguard against users with ill-intent or the posting of malicious/obscene content.

Chapter-5

Design

5.1: Diagrams

5.1.1: Data Flow diagram

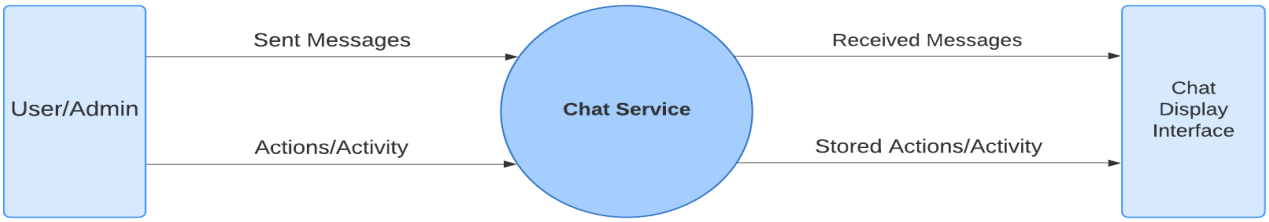

Fig 5.1: Zero Level DFD

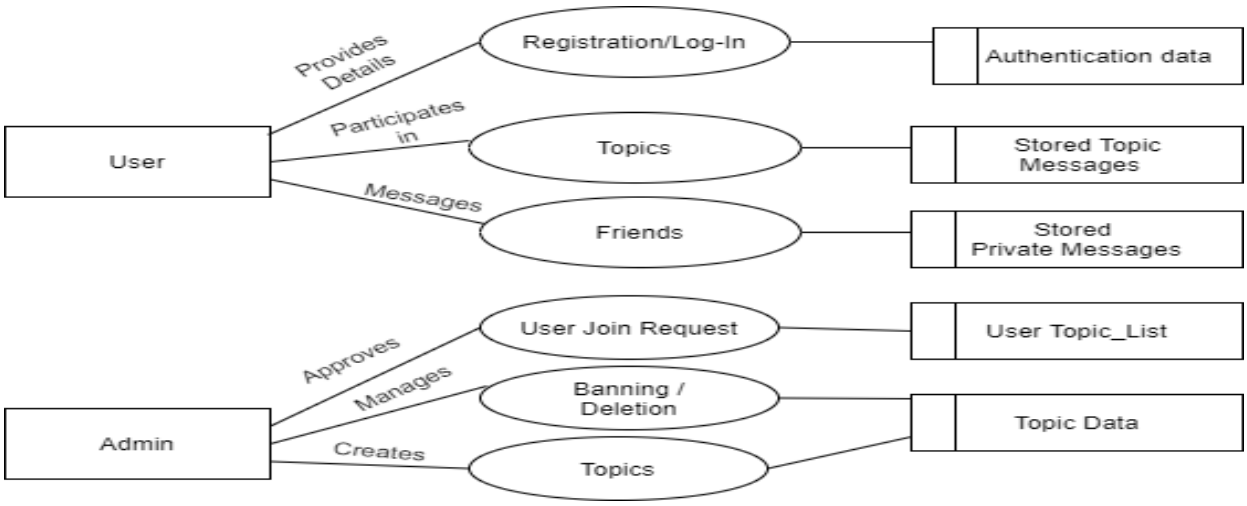

Fig 5.2: Level-one DFD

5.1.2 Use Case Diagram

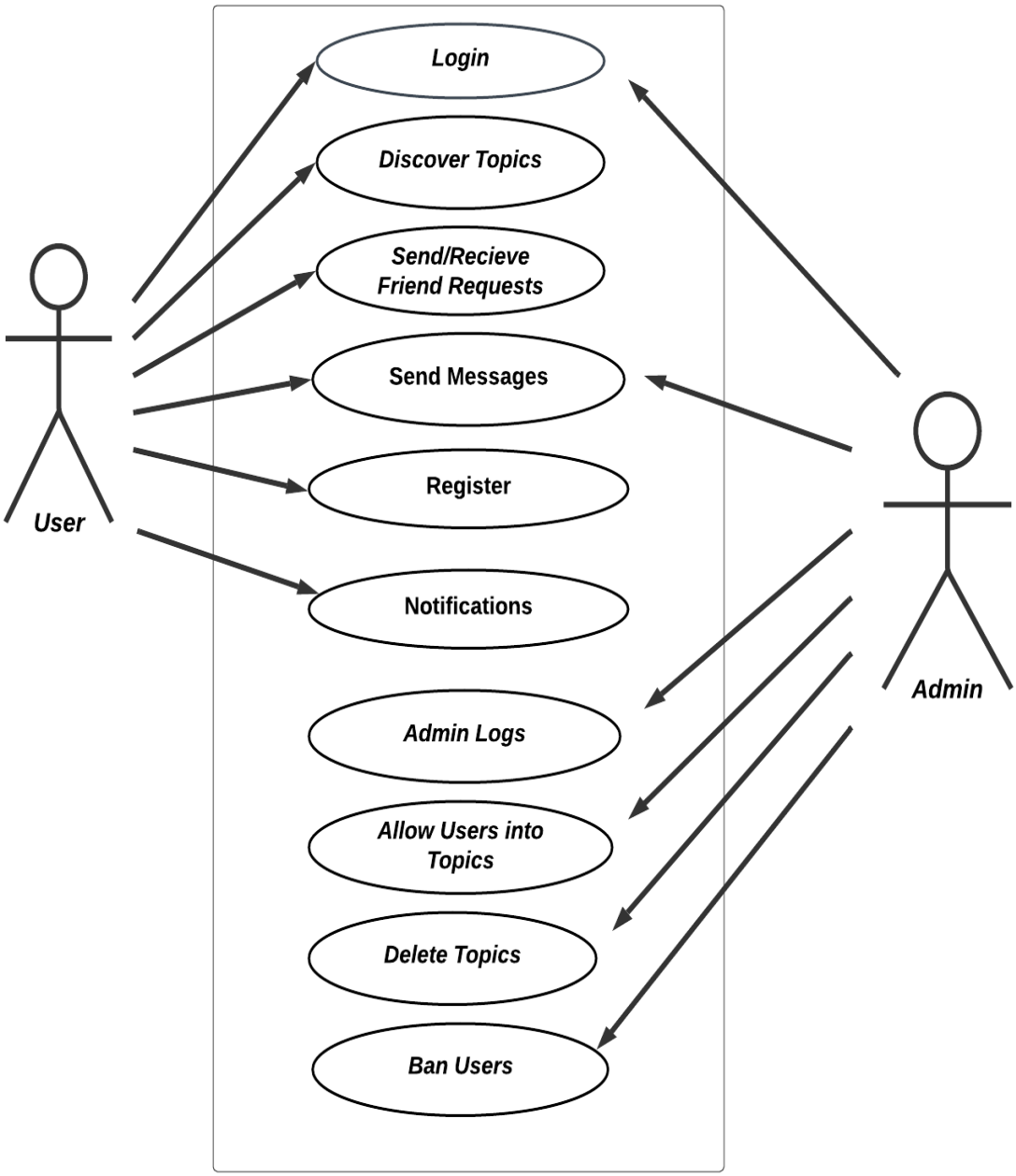

Fig 5.3: Use Case Diagram

5.1.3: E-R Diagram

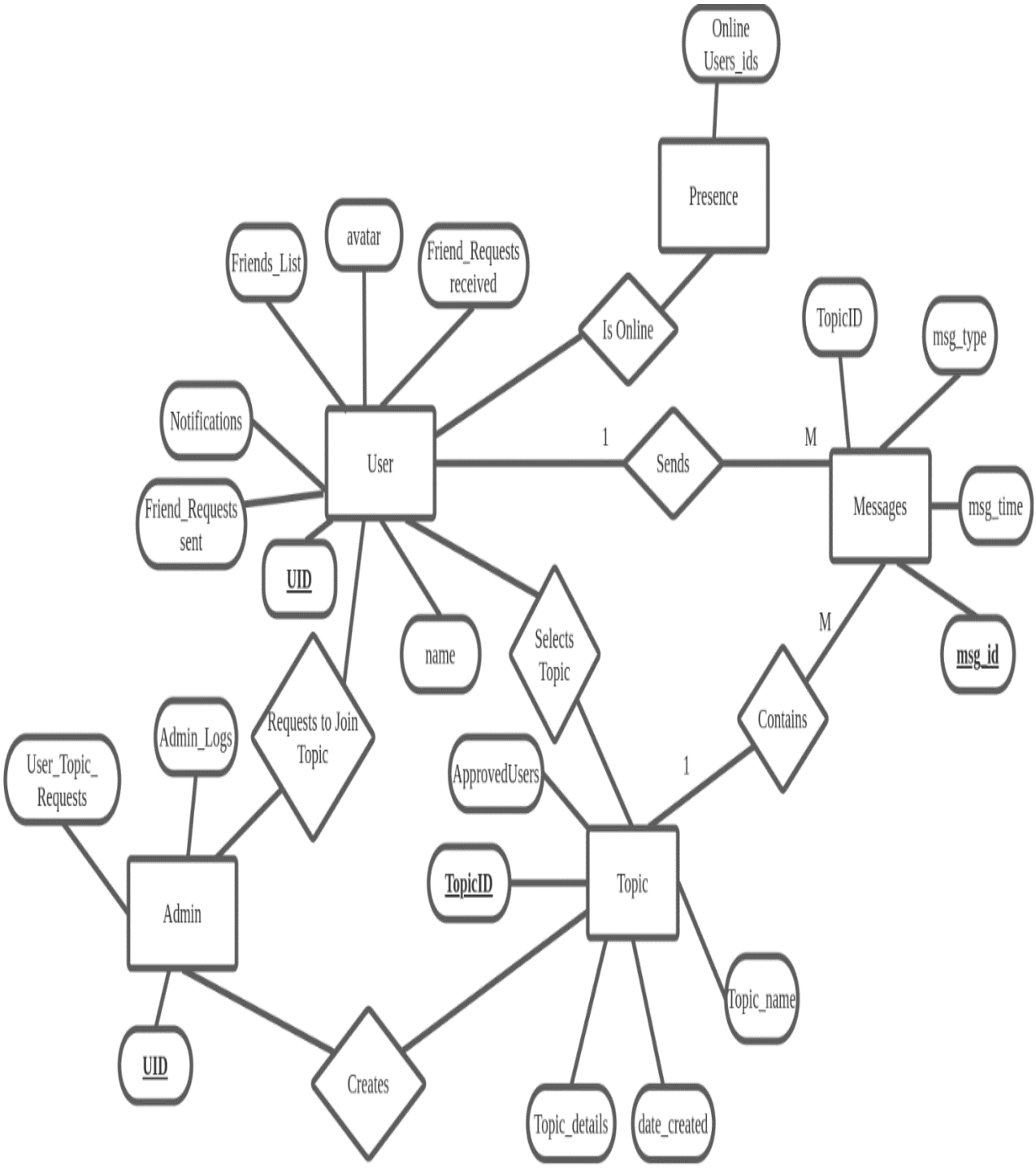

Fig 5.4: E-R Diagram

5.2: Interface Design

5.2.1: Registration Form

Fig 5.5: Registration Form

5.2.2: Login Form

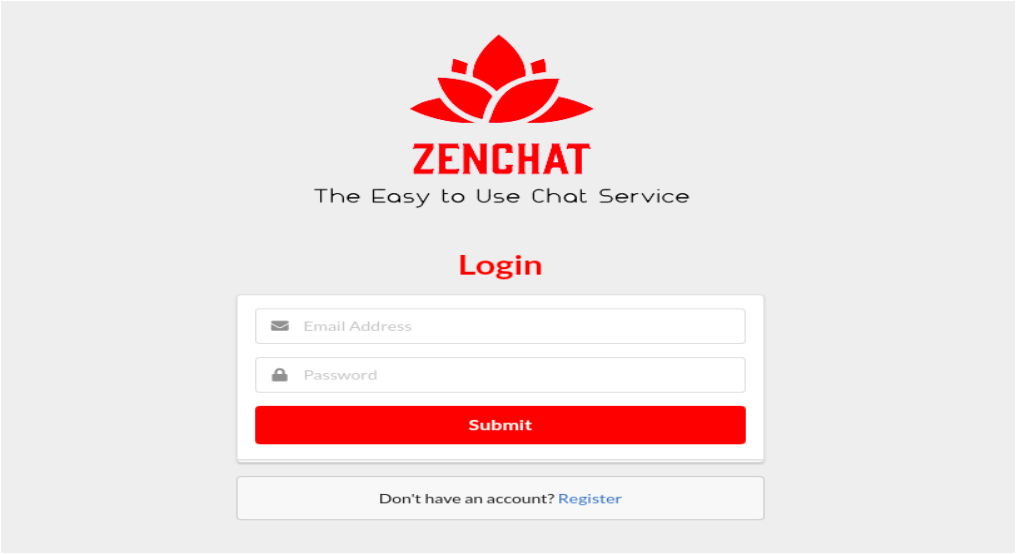

Fig 5.6: User Login Form

5.2.3: Admin Page

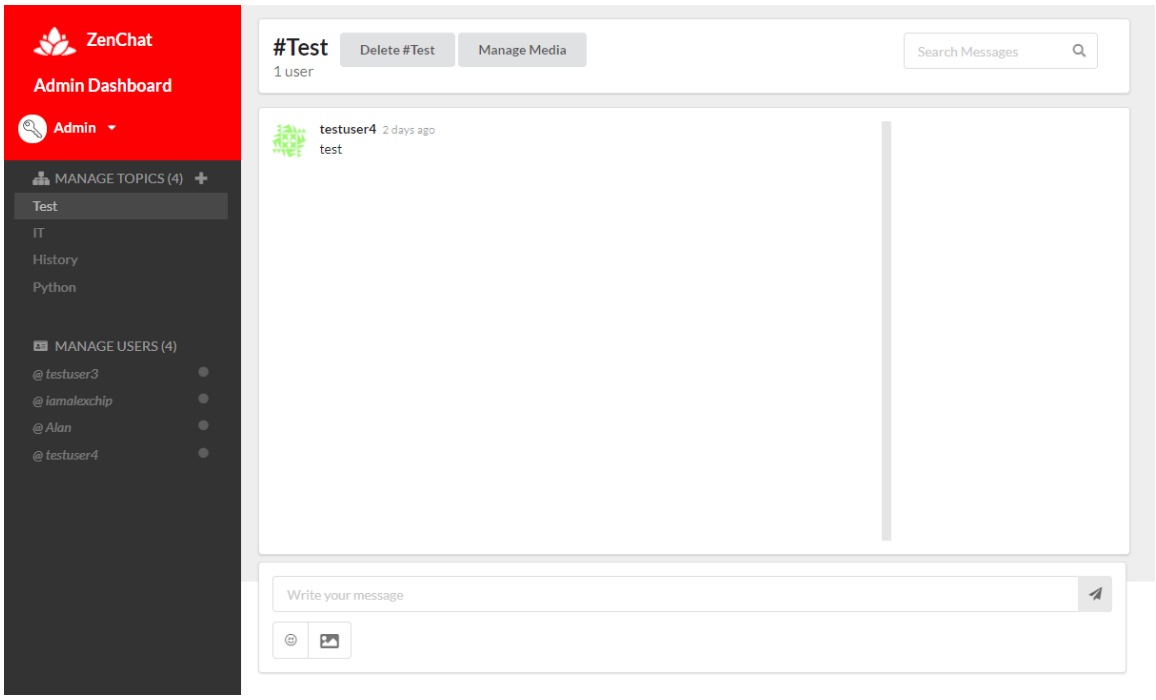

Fig 5.7: Admin Page

5.2.4: Adding a Topic

Fig 5.8: Add Topic Form

5.2.5: After Successfulcreation of a Topic

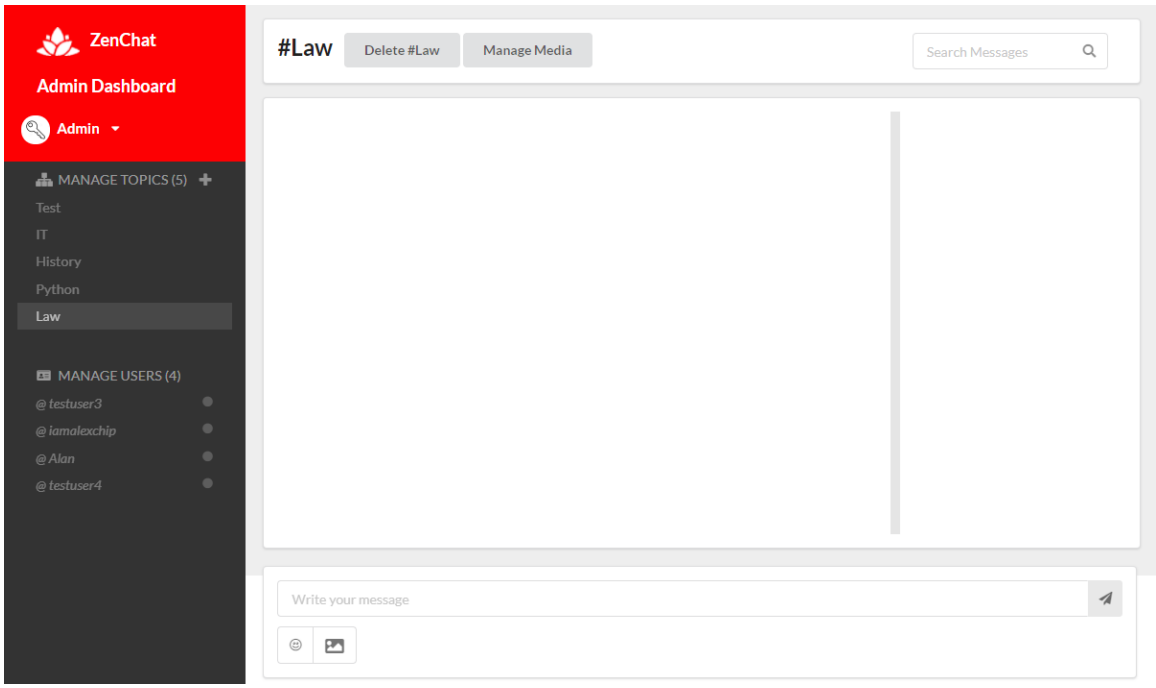

Fig 5.9: SuccessfulTopic Creation by Admin

5.2.6: User Login

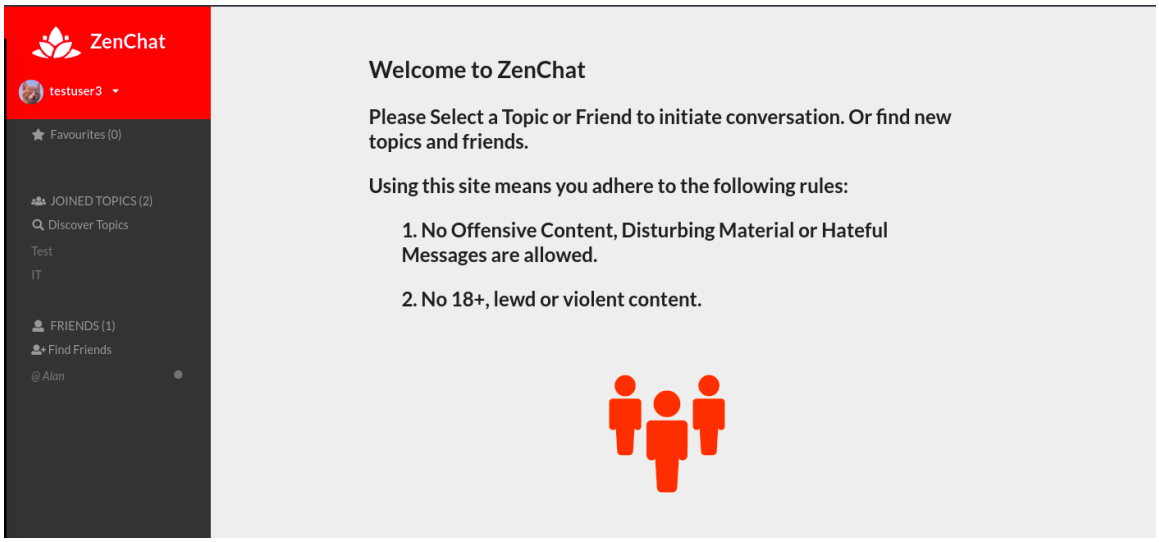

Fig 5.10: User Page after Successful Login

5.2.7: Approval of user join requests

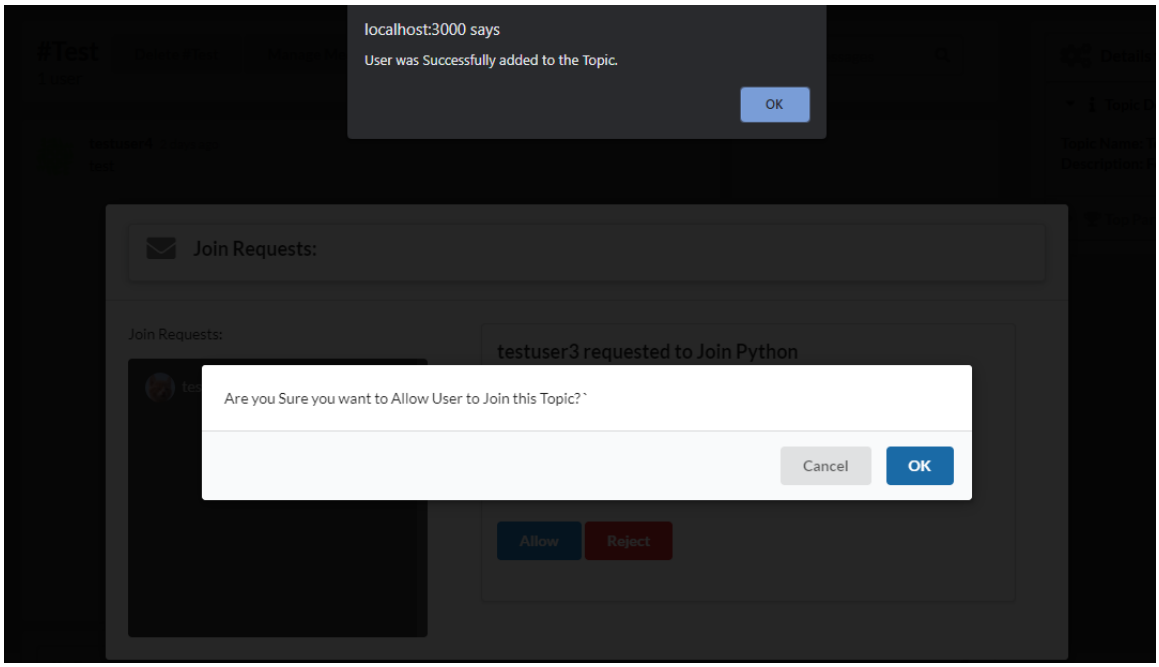

Fig 5.11: Admin approval of user join requests

5.2.8: User Notification after request approval

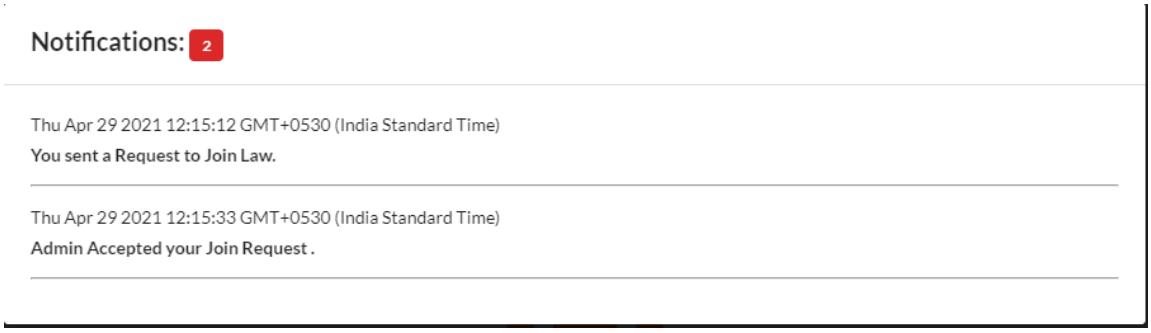

Fig 5.12: Notification after request approval

5.2.9: Topic Deletion by Admin

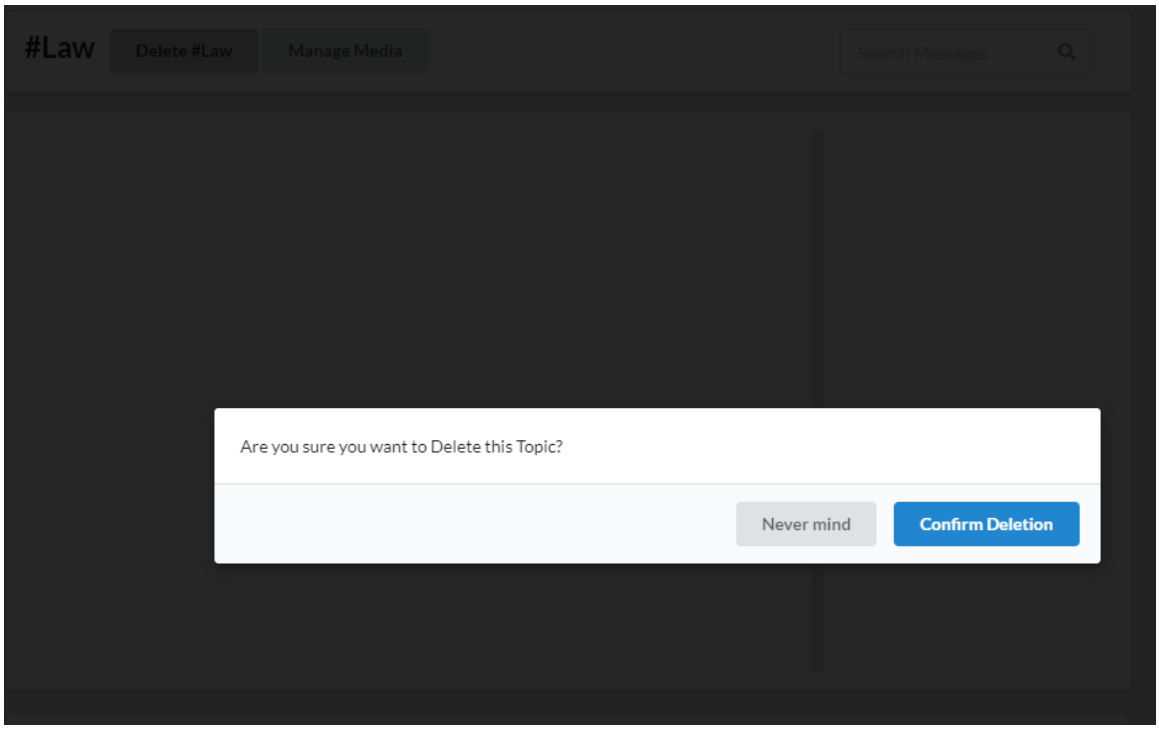

Fig 5.13: Topic Deletion by Admin

5.2.10: Admin Log after Topic Deletion

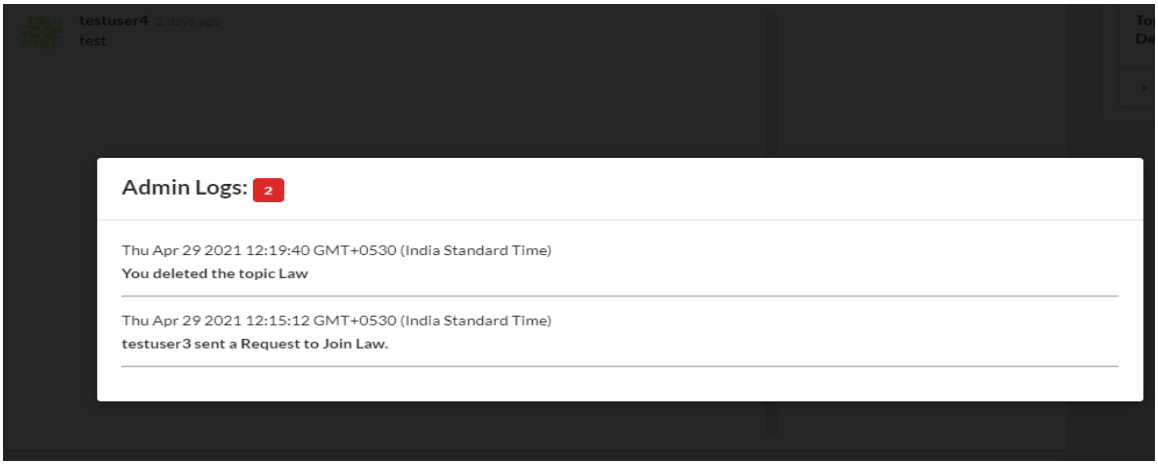

Fig 5.14: Admin Log after Topic Deletion

5.2.11: Displaying All Available Topics

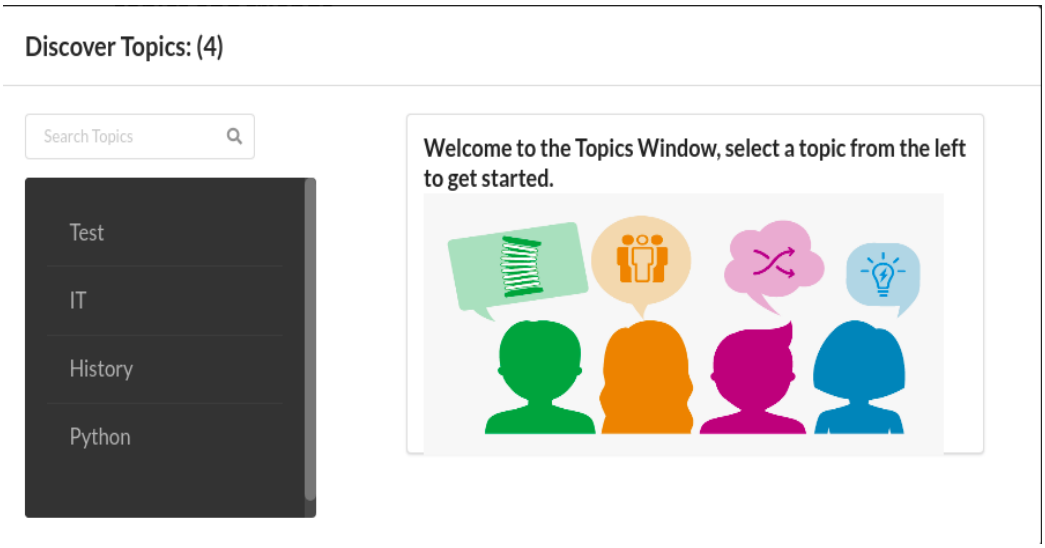

Fig 5.15: Discovering New Topics

5.2.12: Displaying User List

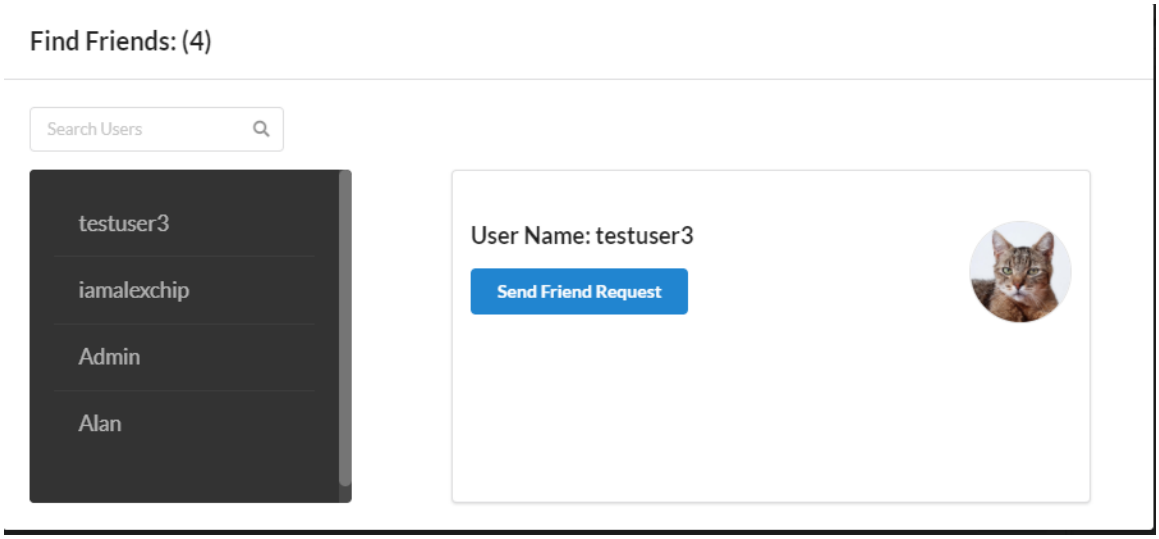

Fig 5.16: Displaying User List

5.2.13: Searching for a Specific Topic

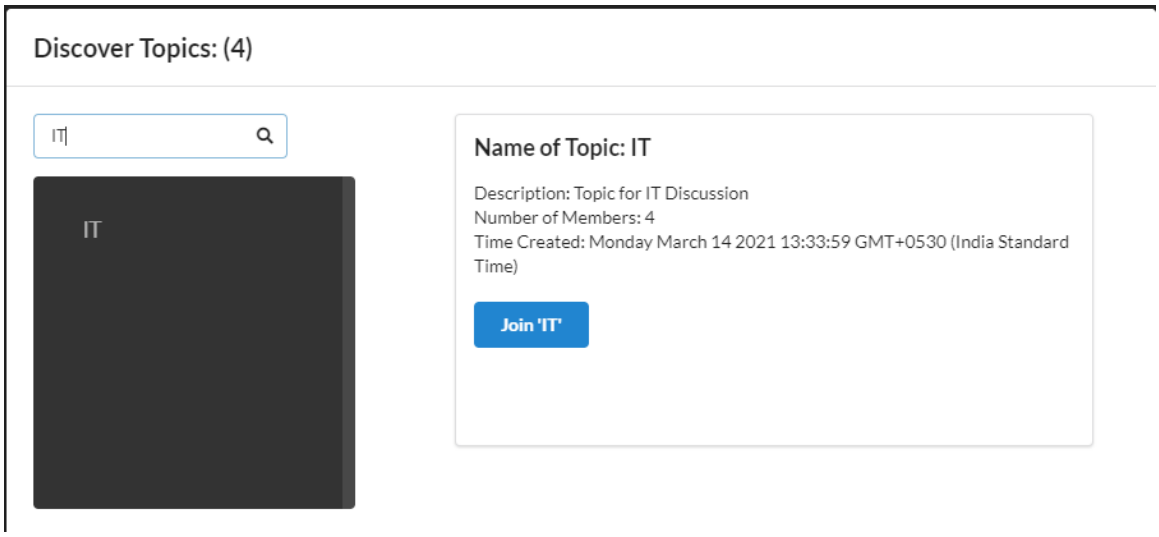

Fig 5.17: Searching for a Specific Topic

5.2.14: Changing User Avatar

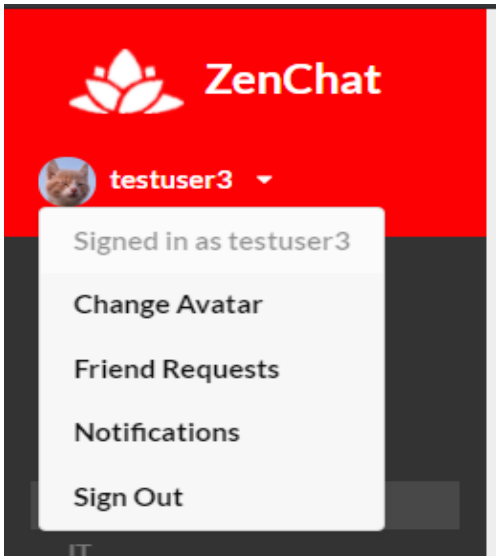

Fig 5.18: Changing User Avatar

5.2.15: Cropping User Avatar

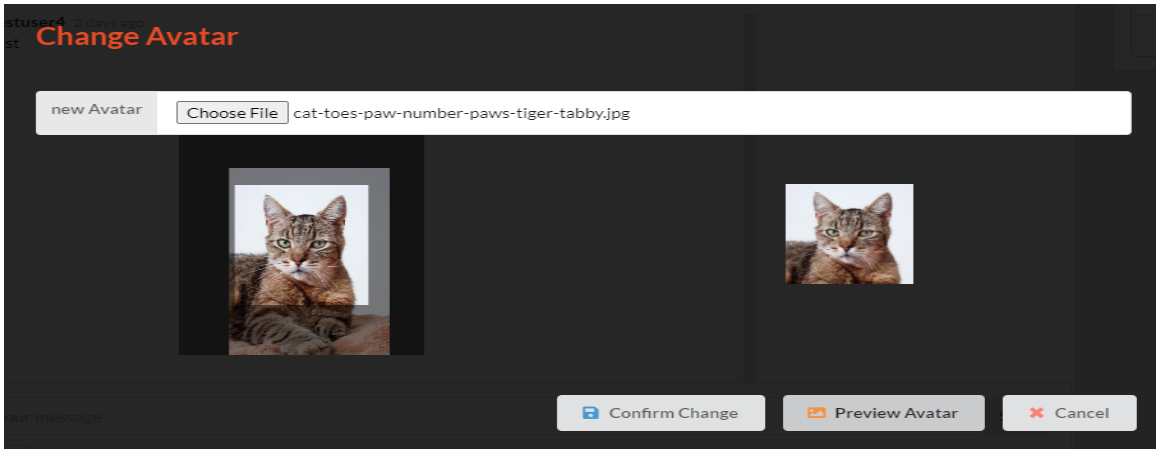

Fig 5.19: Cropping User Avatar

5.2.16: Successful Change of User Avatar

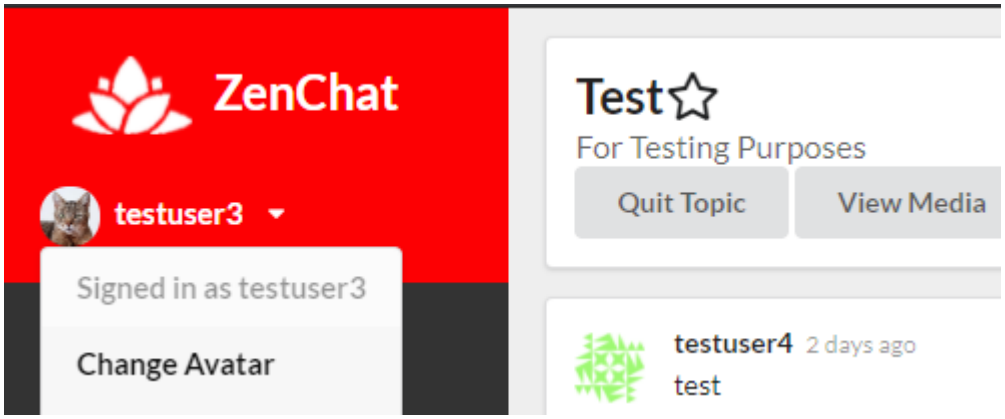

Fig 5.20: Successful Change of User Avatar

5.2.17: Reporting incidents to the Admin

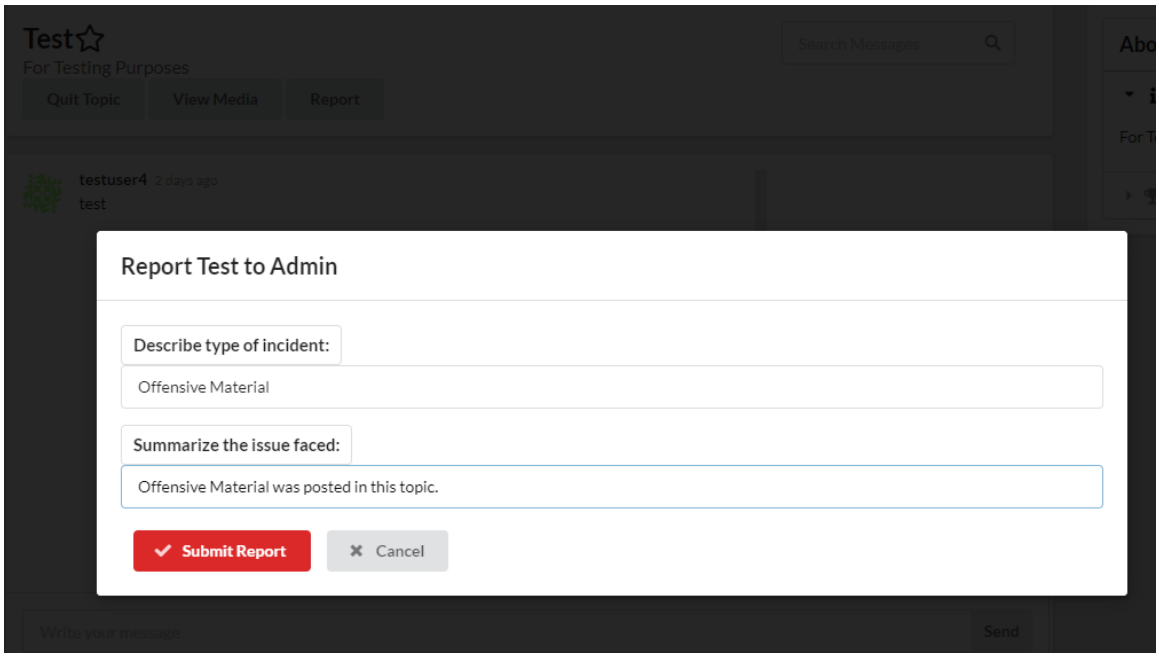

Fig 5.21: Reporting incidents to the Admin

5.2.18: Admin Logs after receiving a report from a user

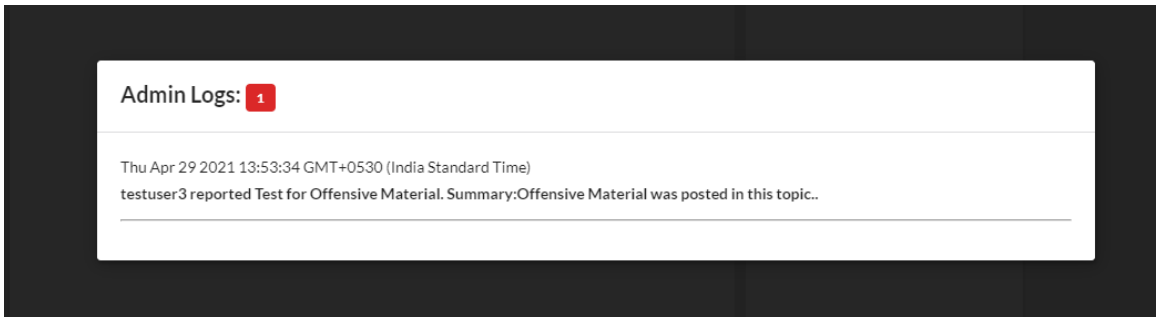

Fig 5.22: Admin Logs after receiving a report from a user

5.2.19: Participating in a Topic

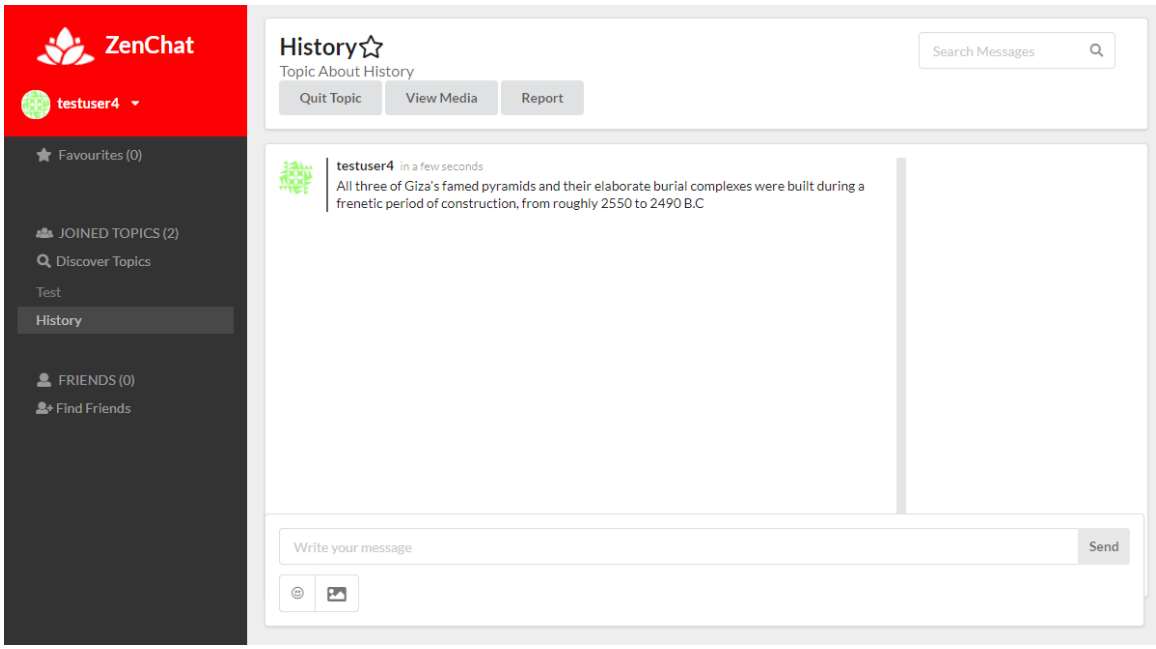

Fig 5.23: Participating in a Topic

5.2.20: User typing indicator

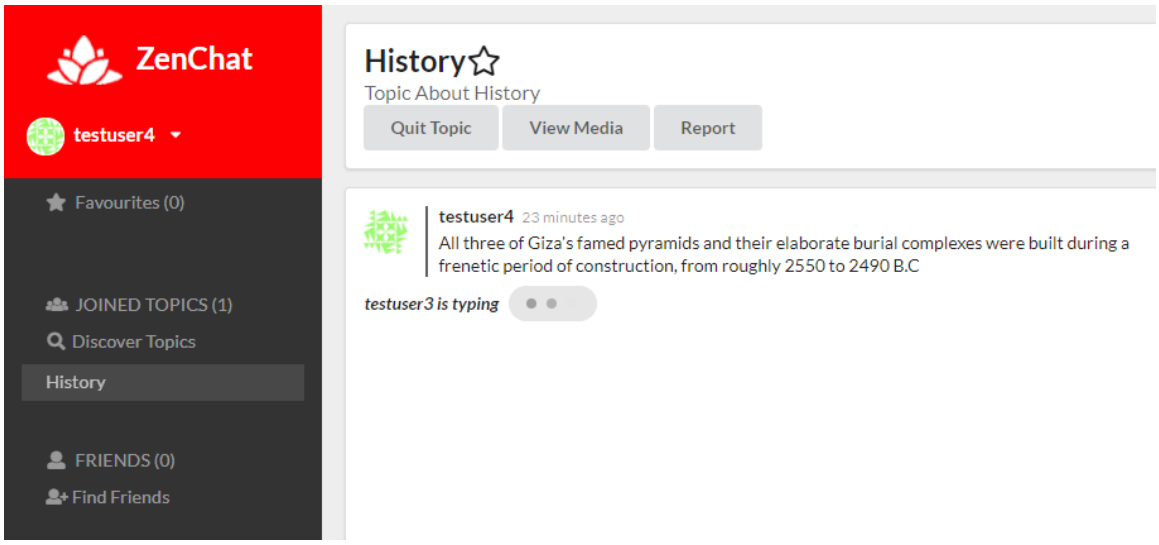

Fig 5.24: User typing indicator

5.2.21: User Uploading Image Form

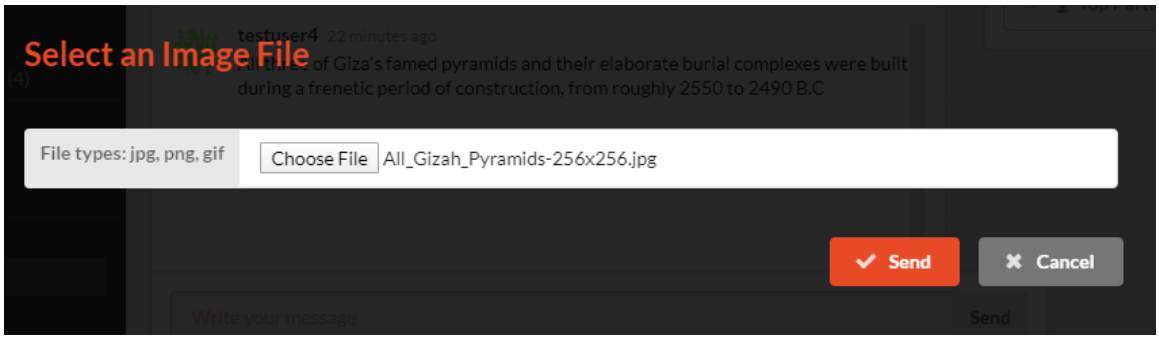

Fig 5.25: User Uploading Image in Messages

5.2.22: User Uploaded Image

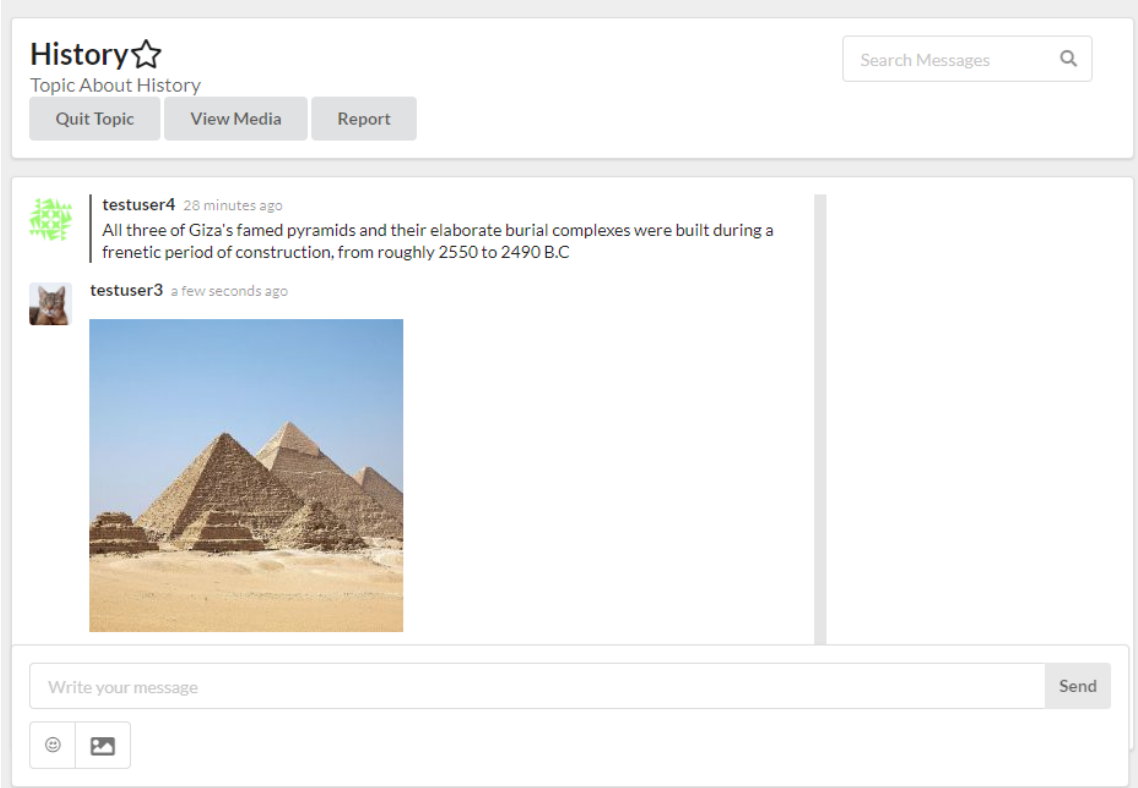

Fig 5.26: User Uploaded Image

5.2.23: User Leaving a Topic

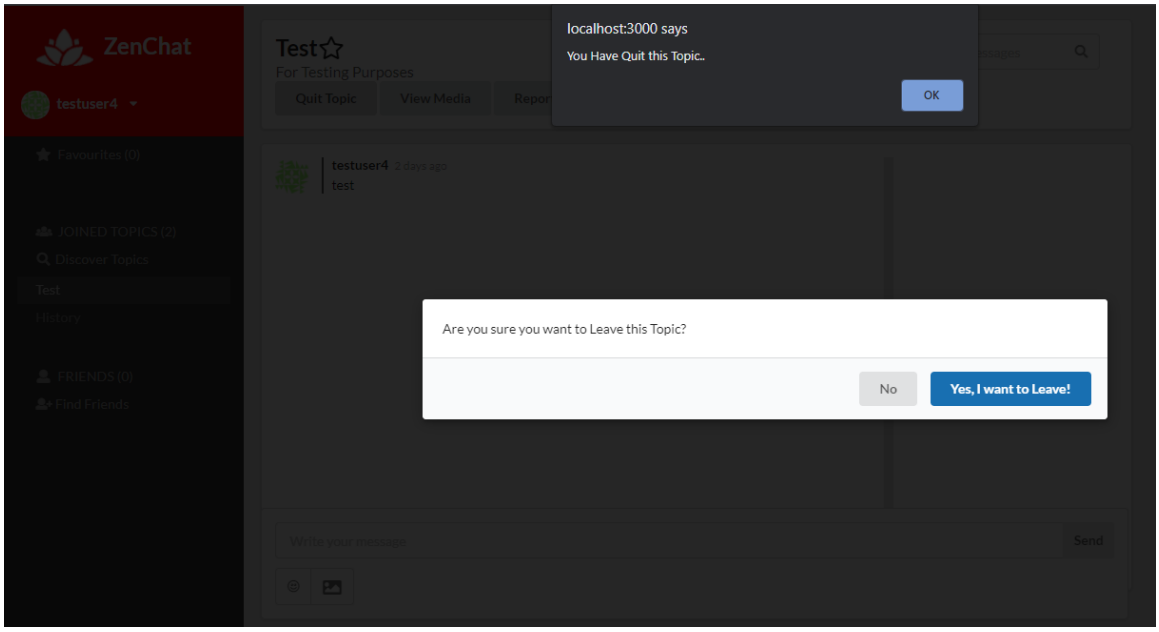

Fig 5.27: User Leaving a Topic

5.2.24: Joined Topics List after User Leaves a Topic

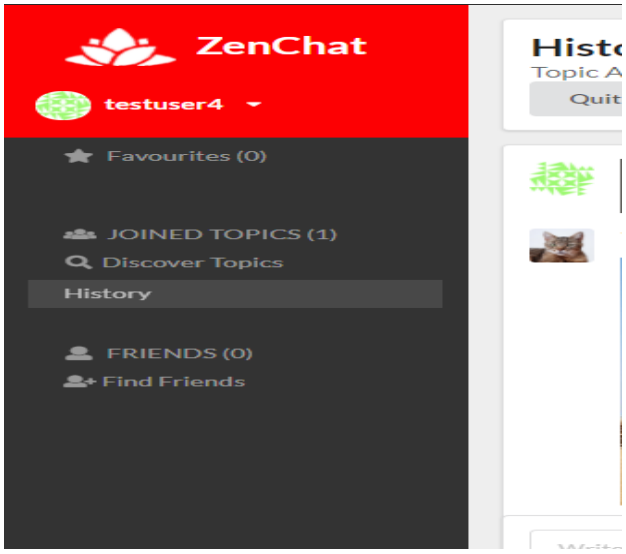

Fig 5.28: Joined Topics List after User Leaves Topic

5.2.25: Admin Manage Media Form

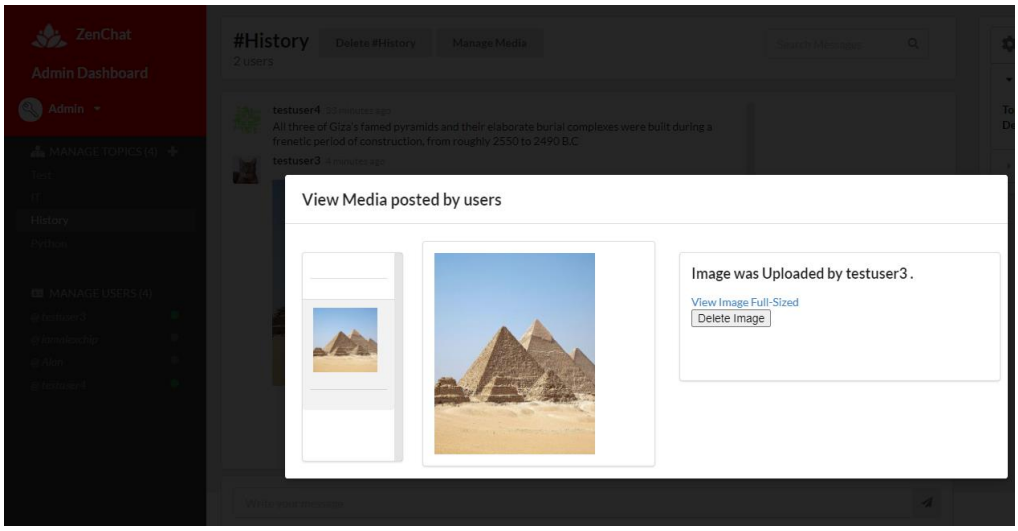

Fig 5.29: Admin options for images uploaded byuser

5.2.26: Admin Sign-Out

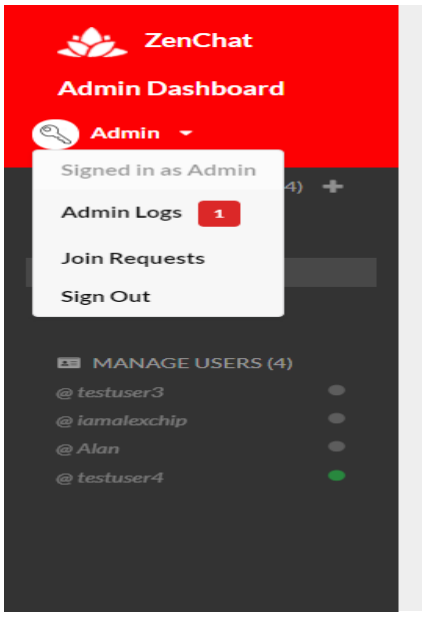

Fig 5.30: Admin Sign-Out

5.2.27: User Sign-Out

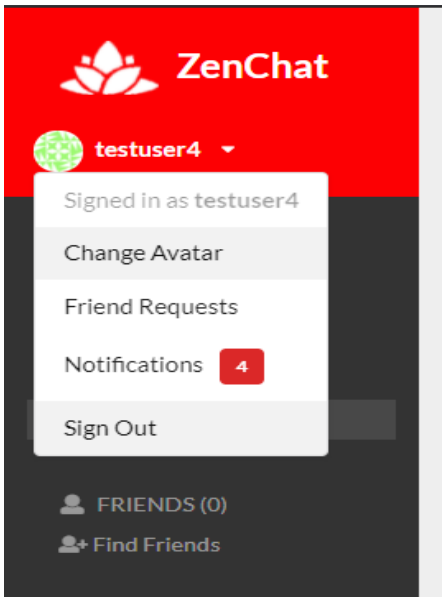

Fig 5.31: User Sign-Out

5.3 Table Structure

5.3.1 User Credentials Table: This table contains the authentication credentials of all the users.

Identifier	Providers	Created	Signed In	User UID
testuser3@gmail.com		Mar 6, 2021	Apr 28, 2021	1BJxumTKQDQmBWcb3ePOpYTli...
gracejoseph1236@gmail.com		Feb 21, 2021	Feb 21, 2021	6KQeV04IGESCj1FKS5xxBdU1faF3
iamalexchip@gmail.com		Apr 3, 2021	Apr 3, 2021	ObEwMSkLI1P2jdnMJ0zROcOgT9...
dave@test.net		Apr 23, 2021	Apr 23, 2021	Vu9jgsgPxOMoNV867X59BltoAGo1
admin@admin.net		Mar 9, 2021	Apr 28, 2021	fvTCEnVcpoOY8bqoPCaUdbig44S2
alanjsebas@gmail.com		Mar 31, 2021	Apr 25, 2021	jwt28DFS2YQ6fHHA83ybJiHRxzD3
pavajeb811@leonvero.com		Mar 15, 2021	Apr 27, 2021	qCp8h87gm5RwQvcVM39JyEFjnC...

Table 5.1: User Credentials Table

5.3.2 User properties: It stores the data associated with the specific user uid.

Table 5.2: User properties table

5.3.3 Topic table: This table contains the datawithin topic.

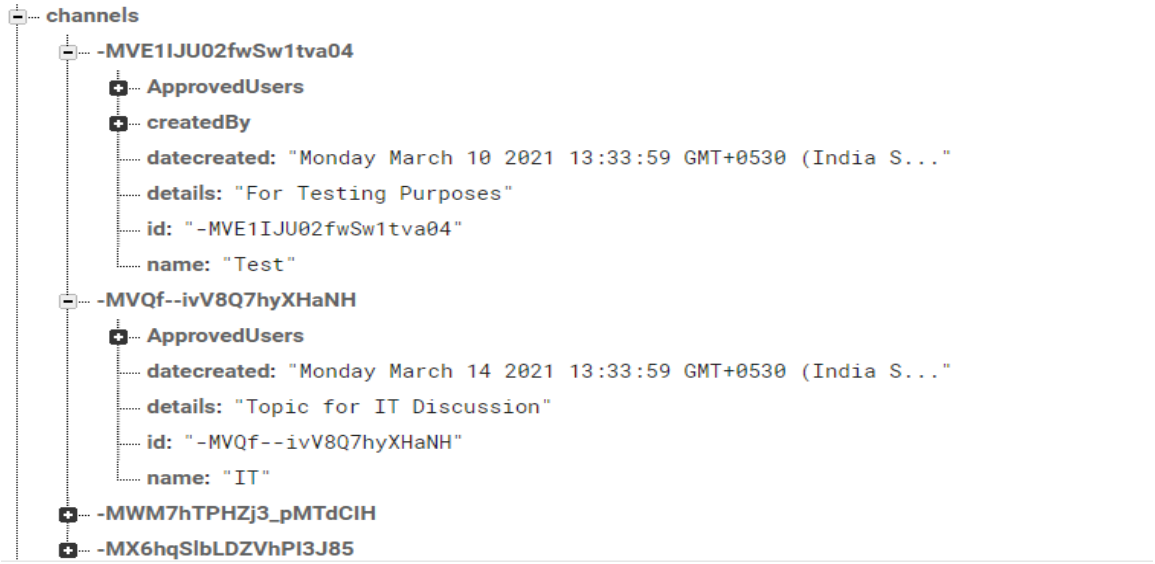

Table 5.3: Topic details table

5.3.4 Messages Table: This table contains the information of message, data &content about the messages and where they belong.

```
messages
  -MVE1IJU02fwSw1tva04
    -MZH5ZQ2XtrcVwPVWtev
      content: "test"
      timestamp: 1619506382870
      user
        avatar: "http://gravatar.com/avatar/34beffb3b31ecd4f3d69..."
        id: "qCp8h87gm5RwQvcVM39JyEFjnC83"
        name: "testuser4"
  -MVQf--ivV8Q7hyXHaNH
    -MZ0lkgOSDOH5oegplRF
      content: "Computers"
      timestamp: 1619232492749
      user
        avatar: "http://gravatar.com/avatar/34beffb3b31ecd4f3d69..."
        id: "qCp8h87gm5RwQvcVM39JyEFjnC83"
        name: "testuser4"
```


Table 5.4: Messages table

5.3.5 Private Messages Table: It contains the data about the private messages.


```
privateMessages
  -MVE1IJU02fwSw1tva04
    1BJxumTKQDQmBWcb3ePOpYTIi962
      -MYz4FEVn8s4470geody
        content: "test"
        timestamp: 1619187271506
        user
  1BJxumTKQDQmBWcb3ePOpYTIi962
    jwt28DFS2YQ6fHHA83ybJiHRxzD3
      -MYz4EF5rZZtEd9OGkZJ
        content: "test"
        timestamp: 1619187267426
        user
    qCp8h87gm5RwQvcVM39JyEFjnC83
```


Table 5.5: Privatemessages table

Chapter 6

Implementation and Integration

6.1Registration Code

```
import React from "react";
import firebase from "../../firebase";
import md5 from "md5";
import logo from "./logo.png";
import { Grid, Form,  Segment,Button,Header,Message} from "semanticui-react";
import { Link } from "react-router-dom";

class Register extends React.Component {
  state = {
    username: "",
    email: "",
    password: "",
passwordConfirmation: "",
    errors: [],
    loading: false,
usersRef: firebase.database().ref("users")
  };

isFormValid = () => {
    let errors = [];
    let error;

    if (this.isFormEmpty(this.state)) {
      error = { message: "Fill in all fields" };
this.setState({ errors: errors.concat(error) });
      return false;
    } else if (!this.isPasswordValid(this.state)) {
      error = { message: "Password is invalid" };
this.setState({ errors: errors.concat(error) });
      return false;
    } else {
      return true;
    }
  };

isFormEmpty = ({ username, email, password, passwordConfirmation }) => {
    return (
!username.length ||
!email.length ||
!password.length ||
!passwordConfirmation.length
```

```
    );
  };

isPasswordValid = ({ password, passwordConfirmation }) => {
    if (password.length< 6 || passwordConfirmation.length< 6) {
      return false;
    } else if (password !== passwordConfirmation) {
      return false;
    } else {
      return true;
    }
  };

displayErrors = errors =>
errors.map((error, i) =><p key={i}>{error.message}</p>);

handleChange = event => {
this.setState({ [event.target.name]: event.target.value });
  };

handleSubmit = event => {
event.preventDefault();
    if (this.isFormValid()) {
this.setState({ errors: [], loading: true });
      firebase
.auth()
.createUserWithEmailAndPassword(this.state.email, this.state.password)
.then(createdUser => {
          console.log(createdUser);
createdUser.user
.updateProfile({
displayName: this.state.username,
photoURL: `http://gravatar.com/avatar/${md5(
createdUser.user.email
            )}?d=identicon`
          })
.then(() => {
this.saveUser(createdUser).then(() => {
console.log("user saved");
            });
          })
.catch(err => {
console.error(err);
this.setState({
              errors: this.state.errors.concat(err),
              loading: false
```

```
            });
          });
        })
.catch(err => {
console.error(err);
this.setState({
          errors: this.state.errors.concat(err),
          loading: false
         });
        });
    }
  };

saveUser = createdUser => {
    return this.state.usersRef.child(createdUser.user.uid).set({
      name: createdUser.user.displayName,
      avatar: createdUser.user.photoURL
    });
  };

handleInputError = (errors, inputName) => {
    return errors.some(error =>error.message.toLowerCase().includes(inputName))
      ? "error"
      : "";
  };

render() {
const {
      username,
      email,
      password,
passwordConfirmation,
      errors,
      loading
    } = this.state;

    return (
<Grid textAlign="center" verticalAlign="middle" className="app">
<Grid.Column style={{ maxWidth: 450 }}>
<br></br>
<imgsrc={logo} alt="Logo" />
<Header as="h1" color="orange" textAlign="center">
          Register for ZenChat
</Header>
<Form onSubmit={this.handleSubmit} size="large">
<Segment stacked>
```

```
<Form.Input
            fluid
            name="username"
            icon="user"
iconPosition="left"
            placeholder="Username"
onChange={this.handleChange}
            value={username}
            type="text"
           />

<Form.Input
            fluid
            name="email"
            icon="mail"
iconPosition="left"
            placeholder="Email Address"
onChange={this.handleChange}
            value={email}
className={this.handleInputError(errors, "email")}
            type="email"
           />

<Form.Input
            fluid
            name="password"
            icon="lock"
iconPosition="left"
            placeholder="Password"
onChange={this.handleChange}
            value={password}
className={this.handleInputError(errors, "password")}
            type="password"
           />

<Form.Input
            fluid
            name="passwordConfirmation"
            icon="repeat"
iconPosition="left"
            placeholder="Password Confirmation"
onChange={this.handleChange}
            value={passwordConfirmation}
className={this.handleInputError(errors, "password")}
            type="password"
           />
```

```
<Button
            disabled={loading}
className={loading ? "loading" : ""}
color="orange"
            fluid
            size="large"
>
            Submit
</Button>
</Segment>
</Form>
        {errors.length> 0 && (
<Message error>
<h3>Error</h3>
           {this.displayErrors(errors)}
</Message>
        )}
<Message>
          Already a user? <Link to="/login">Login</Link>
</Message>
</Grid.Column>
</Grid>
   );
  }
}
export default Register
```

6.2 Login Code

```
import React from "react";
import firebase from "../../firebase";
import logo from "./logo.png";
import {Grid,Form,Segment,Button, Header,Message} from "semantic-ui-react";
import { Link } from "react-router-dom";
class Login extends React.Component {
  state = {
    email: "",
    password: "",
    errors: [],
    loading: false
  };
displayErrors = errors =>
errors.map((error, i) =><p key={i}>{error.message}</p>);

handleChange = event => {
this.setState({ [event.target.name]: event.target.value });
```

```
  };

handleSubmit = event => {
event.preventDefault();
    if (this.isFormValid(this.state)) {
this.setState({ errors: [], loading: true });
      firebase
.auth()
.signInWithEmailAndPassword(this.state.email, this.state.password)
.then(signedInUser => {
          console.log(signedInUser);
        })
.catch(err => {
console.error(err);
this.setState({
            errors: this.state.errors.concat(err),
            loading: false
          });
        });
    }
  };



isFormValid = ({ email, password }) => email && password;

handleInputError = (errors, inputName) => {
    return errors.some(error =>error.message.toLowerCase().includes(inputName))
      ? "error"
      : "";
  };

render() {
const{ email, password, errors, loading } = this.state;

    return (
<Grid textAlign="center" verticalAlign="middle" className="app">
<Grid.Column style={{ maxWidth: 450 }}>
<imgsrc={logo} alt="Logo" />
<Header as="h1" icon color="orange" textAlign="center">
            Login
</Header>
<Form onSubmit={this.handleSubmit} size="large">
<Segment stacked>
<Form.Input
                fluid
```

```
            name="email"
            icon="mail"
iconPosition="left"
            placeholder="Email Address"
onChange={this.handleChange}
            value={email}
className={this.handleInputError(errors, "email")}
            type="email"
           />

<Form.Input
            fluid
            name="password"
            icon="lock"
iconPosition="left"
            placeholder="Password"
onChange={this.handleChange}
            value={password}
className={this.handleInputError(errors, "password")}
            type="password"
           />

<Button
            disabled={loading}
className={loading ? "loading" : ""}
color="orange"
            fluid
            size="large"
>
            Submit
</Button>
</Segment>
</Form>
        {errors.length> 0 && (
<Message error>
<h3>Error</h3>
           {this.displayErrors(errors)}
</Message>
        )}
<Message>
          Don't have an account? <Link to="/register">Register</Link>
</Message>
</Grid.Column>
</Grid>
   );
  }
```

```
}
export default Login;
```

6.3 Code for Main Interface

```
import React from "react";
import { Grid, Segment } from "semantic-ui-react";
import "./App.css";
import { connect } from "react-redux";
import SidePanel from "./SidePanel/SidePanel";
import Messages from "./Messages/Messages";
import MetaPanel from "./MetaPanel/MetaPanel";
const App = ({ currentUser, currentChannel, isPrivateChannel, userPosts }) => (
<Grid columns="equal" className="app" style={{ background: "#eee" }}>
<ColorPanel />
<SidePanel
    key={currentUser&&currentUser.uid}
currentUser={currentUser}
  />
<Grid.Column style={{ marginLeft: 160 }}>
{!currentChannel&&<Segment basic><div style={{ marginLeft: 160, marginTop:30
}}><h1>Welcome to ZenChat</h1>
<h2>Please Select a Topic or Friend to initiate conversation. Or find new topics and
friends. </h2>
<h2>Using this site means you adhere to the following rules:
<ul>1. No Offensive Content, Disturbing Material or Hateful Messages are allowed.</ul>
<ol>2. No 18+, lewd or violent content.</ol></h2>
<center><img
src="https://welzijnskompas.nl/app/uploads/2017/02/social_icon.png"></img></center>
</div></Segment>}
<Messages
    key={currentChannel&& currentChannel.id}
currentChannel={currentChannel}
currentUser={currentUser}
isPrivateChannel={isPrivateChannel}

    />
</Grid.Column>
<Grid.Column width={4}>
<MetaPanel
    key={currentChannel&& currentChannel.id}
userPosts={userPosts}
currentChannel={currentChannel}
isPrivateChannel={isPrivateChannel}
    />
</Grid.Column>
</Grid>
```

```
);



constmapStateToProps = state => ({
currentUser: state.user.currentUser,
currentChannel: state.channel.currentChannel,
isPrivateChannel: state.channel.isPrivateChannel,
userPosts: state.channel.userPosts
});

export default connect(mapStateToProps)(App);
```

6.4 Code for SidePanel

```
import React from "react";
import UserPanel from "./UserPanel";
import Channels from "./Channels";
import DirectMessages from "./DirectMessages";
import JoinedTopics from "./JoinedTopics";
import Starred from "./Starred";
import { Menu } from "semantic-ui-react";

class SidePanel extends React.Component {
render() {
const{ currentUser } = this.props;

   return (
<Menu
      size="large"
      inverted
      fixed="left"
      vertical
      style={{ background: "#333333", fontSize: "1.2rem" }}
>
<UserPanelcurrentUser={currentUser} />
<Starred currentUser={currentUser}/>
<JoinedTopicscurrentUser={currentUser}/>
<DirectMessagescurrentUser={currentUser} />

</Menu>
   );
  }
}

export default SidePanel;
```

6.5 Code for Displaying &Messaging Friends

```
import React from "react";
import firebase from "../../firebase";
import {connect} from "react-redux";
import {setCurrentChannel,setPrivateChannel} from "../../actions";
import { Menu, Icon, Button } from "semantic-ui-react";
import AddFriends from "./Discover/AddFriends";

class DirectMessages extends React.Component {
  state = {
activeChannel: "",
    user: this.props.currentUser,
    users: [],
usersRef: firebase.database().ref("users"),
FriendsList: firebase.database().ref("/users/" + firebase.auth().currentUser.uid +
"/Friend_List"),
connectedRef: firebase.database().ref(".info/connected"),
presenceRef: firebase.database().ref("presence"),
loadedFriends: [],
    friends: []
  };

componentDidMount() {
    if (this.state.user) {
this.addListeners(this.state.user.uid);
    }
  }

addListeners = currentUserUid => {
    let loadedUsers = [];
this.state.FriendsList.on("child_added", snap => {
      if (currentUserUid !== snap.key) {
        let user = snap.val();
        user["uid"] = snap.key;
        user["status"] = "offline";
loadedUsers.push(user);
this.setState({ users: loadedUsers });
      }

    }

    );

this.state.FriendsList.on("child_removed", snap => {
constFriendToRemove = { id: snap.key, ...snap.val() };
constfilteredFriends = this.state.users.filter(user =>{
```

```
      return user.uid != FriendToRemove.id;
    });
this.setState({users:filteredFriends});
window.location.reload();
  });


this.state.connectedRef.on("value", snap => {
      if (snap.val() === true) {
const ref = this.state.presenceRef.child(currentUserUid);
ref.set(true);
ref.onDisconnect().remove(err => {
          if (err !== null) {
console.error(err);
          }
        });
      }
    });

this.state.presenceRef.on("child_added", snap => {
      if (currentUserUid !== snap.key) {
this.addStatusToUser(snap.key);
      }
    });

this.state.presenceRef.on("child_removed", snap => {
      if (currentUserUid !== snap.key) {
this.addStatusToUser(snap.key, false);
      }
    });
  };


addStatusToUser = (userId, connected = true) => {
constupdatedUsers = this.state.users.reduce((acc, user) => {
      if (user.uid === userId) {
        user["status"] = `${connected ? "online" : "offline"}`;
      }
      return acc.concat(user);
    }, []);
this.setState({ users: updatedUsers });
  };

isUserOnline = user =>user.status === "online";
```

```
changeChannel = (user) =>{
constchannelId = this.getChannelId(user.uid);
constchannelData={
      id: channelId,
      name: user.name
    };
this.props.setCurrentChannel(channelData);
this.props.setPrivateChannel(true);
this.setActiveChannel(user.uid);
  }

getChannelId = userId =>{
constcurrentUserId = this.state.user.uid;
    return userId<currentUserId ?
     `${userId}/${currentUserId}`:`${currentUserId}/${userId}`;

  }

setActiveChannel = userId =>{
this.setState({activeChannel: userId});
  }

render() {
const{ users,activeChannel } = this.state;
const{ currentUser } = this.props;


    return (
<Menu.MenuclassName="menu">
<Menu.Item>
<span>
<Icon name="user" /> FRIENDS

</span>{" "}
         ({users.length})


</Menu.Item>
<span><AddFriendscurrentUser={currentUser}/></span>
        {users.map(user => (
<Menu.Item
           key={user.uid}
           active={user.uid === activeChannel}
onClick={() =>this.changeChannel(user)}
           style={{ opacity: 0.7, fontStyle: "italic" }}
```

```
>
<Icon
            name="small circle"
color={this.isUserOnline(user) ? "green" : "gray"}
          />
          @ {user.name}
</Menu.Item>
       ))}
</Menu.Menu>
    );
  }
}

export default connect(null,{setCurrentChannel,setPrivateChannel})(DirectMessages);
```

6.6User Panel Code

```
import React from "react";
import firebase from "../../firebase";
import {connect} from "react-redux";
import logo from "./lo.png";
import Requests from "./Requests";
import Notifications from "./Notifications";
import { Grid, Header, Dropdown, Image, Modal, Input, Button, Icon } from "semantic-
ui-react";
import AvatarEditor from "react-avatar-editor";
class UserPanel extends React.Component {
  state = {
    user: this.props.currentUser,
    modal: false,
previewImage: '',
croppedImage: '',
    blob: null,
IsSubmitReq: false,
uploadCroppedImage: '',
storageRef: firebase.storage().ref(),
userRef: firebase.auth().currentUser,
usersRef: firebase.database().ref('users'),
    metadata: {
contentType:"image/jpg"
    }

  };
openModal= () =>this.setState({modal:true});
closeModal= () =>this.setState({modal:false});
componentDidMount(){
```

```
this.setState({ user: this.props.currentUser});
  }
dropdownOptions = () => [
    {
      key: "user",
      text: (
<span>
          Signed in as <strong>{this.state.user.displayName}</strong>
</span>
      ),
      disabled: true
    },
    {
      key: "avatar",
      text: <span onClick={this.openModal}><div>Change Avatar</div></span>
    },
    {
      key: "frenReq",
      text: <Requests></Requests>

    },
    {
      key: "frenReq",
      text: <span><Notifications></Notifications></span>

    },
    {
      key: "signout",
      text: <span onClick={this.handleSignout}><div>Sign Out</div></span>
    }


  ];

uploadCroppedImage= () =>{
const {storageRef,userRef,blob,metadata}= this.state;

storageRef
.child(`avatars/users/${userRef.uid}`)
.put(blob,metadata)
.then(snap =>{
snap.ref.getDownloadURL().then(downloadURL => {
this.setState({uploadCroppedImage:downloadURL},()=>
this.changeAvatar()
        );
        });
```

```
      });
  };

changeAvatar= () =>{
this.state.userRef
.updateProfile({
photoURL: this.state.uploadCroppedImage
    })
.then(() => {
console.log('PhotoURL updated');
this.closeModal();
    })
.catch(err=>{
      console.log(err);
    });
this.state.usersRef
.child(this.state.user.uid)
.update({avatar:this.state.uploadCroppedImage})
.then(()=>{
console.log('User Avatar  updated');
    })
.catch(err => {
      console.log(err);
    });
  };

handleChange= event =>{
const file =event.target.files[0];
const reader= new FileReader();

    if (file){
reader.readAsDataURL(file);
reader.addEventListener('load', () =>{
this.setState({ previewImage:reader.result});
      });
  }
};

SubmitFrenReq = () => {
this.setState(prevState=>({
IsSubmitReq: !prevState.IsSubmitReq
  }), () =>this.subFriend());
console.log("Request Sent");

}
```

```
subFriend = () =>{
  if(this.state.IsSubmitReq){
console.log("Friend  Request Sent");
}else{
console.log("Request Not Sent");
  }
}


handleCropImage = () => {
    if(this.AvatarEditor){
this.AvatarEditor.getImageScaledToCanvas().toBlob(blob => {
        let imageURL = URL.createObjectURL(blob);
this.setState({
croppedImage: imageURL,
          blob
        });
      });
    }

  };

handleSignout = () => {
    firebase
.auth()
.signOut()
.then(() => console.log("signed out!"));
window.location.reload();
  };

render() {
const{ user, modal, previewImage, croppedImage} = this.state;

    return (
<Grid style={{ background: "#fd0003" }}>
<Grid.Column>
<Grid.Row style={{ padding: "1.2em", margin: 0 }}>
            {/* App Header */}
<Header inverted floated="left" as="h2">
<imgsrc={logo} align="top" alt="Inverted Logo"/>
<Header.Content>ZenChat</Header.Content>
</Header>
</Grid.Row>
          {/* User Dropdown  */}
<br></br>
<Header style={{ padding: "0.25em" }} as="h4" inverted>
```

```
<Dropdown
            trigger={
<span>
<Image src={user.photoURL} spaced="right" avatar/>
              {user.displayName}
</span>}
            options={this.dropdownOptions()}
           />
</Header>
         {/*Change User Avatar Modal*/}
<Modal basic open={modal} onClose={this.closeModal}>
<Modal.Header>
<h2 class="ui orange header">
            Change Avatar
</h2>
</Modal.Header>
<Modal.Content>
<Input
onChange={this.handleChange}
            fluid
            type="file"
            label="new Avatar"
            name="previewImage"
            />
<Grid centered stackable columns={2}>
<Grid.Rowcentered>
<Grid.ColumnclassName="uicenter aligned grid">
                  {previewImage&&(
<AvatarEditor
                    ref={node =>(this.AvatarEditor = node)}
                    image={previewImage}
                    width={120}
                    height={120}
                    border={50}
                    scale={1.2}
                    />
                  )}
                  {/*Image Preview*/}
</Grid.Column>
<Grid.Column>
                  {croppedImage&&(
<Image
                    style={{margin: '3.5em auto'}}
                    width={100}
                    height={100}
src={croppedImage}
```

```
                    />
                  )}
</Grid.Column>
</Grid.Row>
</Grid>
</Modal.Content>
<Modal.Actions>
            {croppedImage&&<Button onClick={this.uploadCroppedImage}>
<Icon name="save" color="blue"/>Confirm Change
</Button>}
<Button onClick={this.handleCropImage}>
<Icon name="image" color="orange" inverted/>Preview Avatar
</Button>
<Button onClick={this.closeModal}>
<Icon name="remove" color="red" inverted/>Cancel
</Button>
</Modal.Actions>
</Modal>
</Grid.Column>
</Grid>
    );
  }
}
constmapStateToProps = state => ({
currentUser: state.user.currentUser
})
export default connect(mapStateToProps)(UserPanel);
```

6.7 Code for Notifications

```
import React from "react";
import firebase from "../../firebase";
import  {Icon, Modal, Label, Feed } from "semantic-ui-react";

class Notifications extends React.Component {
    state = {
        user: [],
        open: false,
NotificationsRef: firebase.database().ref("/users/" + firebase.auth().currentUser.uid +
"/Notifications"),
        Notifications: []
      };

componentDidMount(){
this.addListeners();
      }
```

```
     //Load Notifications for Displaying Unread
addListeners=() =>  {
       let loadednotifs = [];
this.state.NotificationsRef.on("child_added", snap => {
       let notifs = snap.val();
notifs["id"]=snap.key;
loadednotifs.push(notifs);
this.setState({ Notifications: loadednotifs })
       })

      //Remove Read Notifications
this.state.NotificationsRef.on("child_removed", snap => {
constNotificationsToRemove = { id: snap.key, ...snap.val() };
constFilteredNotifications = this.state.Notifications.filter(notifs=>{
         return notifs.id != NotificationsToRemove.id;
       });
this.setState({Notifications:FilteredNotifications});

     });
   }

displayNotifications = Notifications => (
Notifications.length>0  &&Notifications.map(notifs => (
<React.Fragment>
<Feed>
<Feed.Event>
                 {notifs.time}
</Feed.Event>
<Feed.Content>
<div class="ui middle aligned divided list">
<div class="item">
</div>
<div class="content">
<h5><b>{notifs.message}</b></h5>
<hr></hr>

</div>
</div>
<div class="item">
</div>
</Feed.Content>
</Feed>
</React.Fragment>

       ))
   )
```

```
onClose = () => {
this.state.NotificationsRef
.remove(err=> {
if(err !==null){
          console.log(console.error());
       }
       })
this.setState({open: false});
     }
NotificationsOpen=() =>this.setState({open: true});
render() {
const{ user, Notifications } = this.state;

       return (
<div>
<span onClick={this.NotificationsOpen}>Notifications {Notifications.length>0 &&
(<Label  color="red"  >{Notifications.length}</Label>)} </span>
<Modal open={this.state.open} onClose={this.onClose}>
<Modal.Header>Notifications: <Label  color="red"
>{Notifications.length}</Label></Modal.Header>
<Modal.Content>
<Modal.Description>
<span>
                          {this.displayNotifications(Notifications)}
</span>
</Modal.Description>
<React.Fragment>
</React.Fragment>
</Modal.Content>
</Modal>
</div>
       );
     }
   }
export default Notifications;
```

Chapter 7

Testing

7.1 Login Page

Case 1: Successful Login

Attribute	Value
Form	Login Form
Action	Validates the user credentials.
Expected results	If the right credentials are provided, the form accepts the data and the user is logged into the service.

Table 7.1: Successful Login

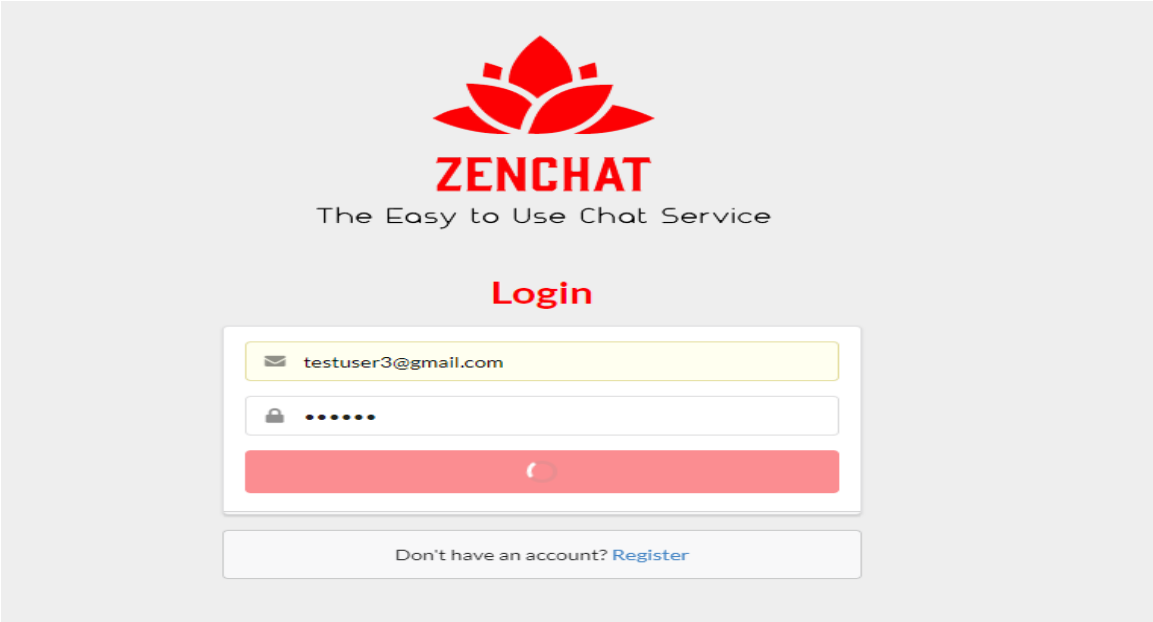

Fig 7.1 Successful Login

Case 2: Invalid Login

7.1.2 Invalid Login

Attribute	Value
Form	Login Form
Action	Validates the user credentials.
Expected results	If user provided credentials are invalid or do not match form requirements, the errors are displayed.

Table 7.2: Invalid Login

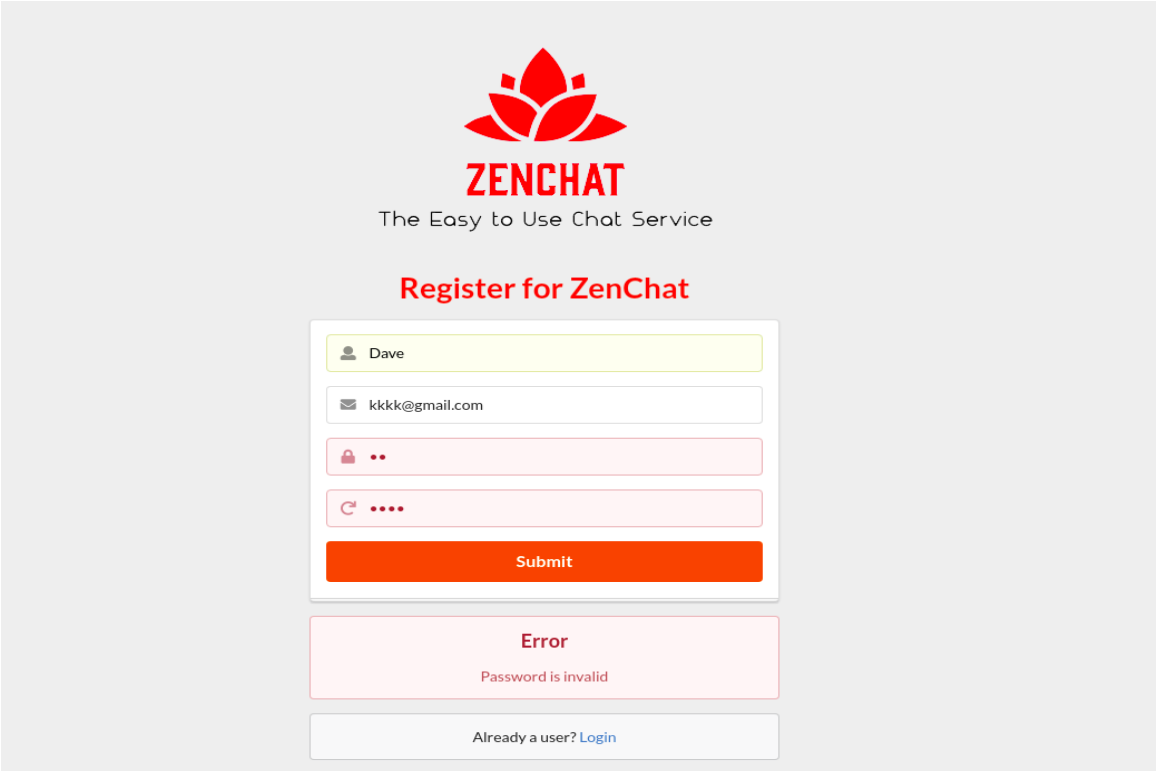

Fig 7.2 Invalid Login

7.2 New Account Opening

7.2.1 New Account Opening Error

Case 1: Empty Fields

Attribute	Value
Form	Registration Form
Action	Validates that some data is present in the fields.
Expected results	Points out to fill empty data.

Table 7.3: New Account opening Registration

Fig 7.3Invalid Account Opening

7.2.2 New Account Opening

Case 2: Email field invalid

Attribute	Value
Form	Registration Form

Action	Checks if mail id is entered is in proper format or not.
Expected results	Points out if email field does not match requirements.

Table 7.4: New Account opening Registration

Fig 7.4 Invalid Account Registration

7.3 Searching for a Topic

Case 1: Topic Search Results

Attribute	**Value**
Form	List of Topics
Action	Filtering topics whose name matches the user query.
Expected results	Displays specific results.

Table 7.5: Successful Searching Forum

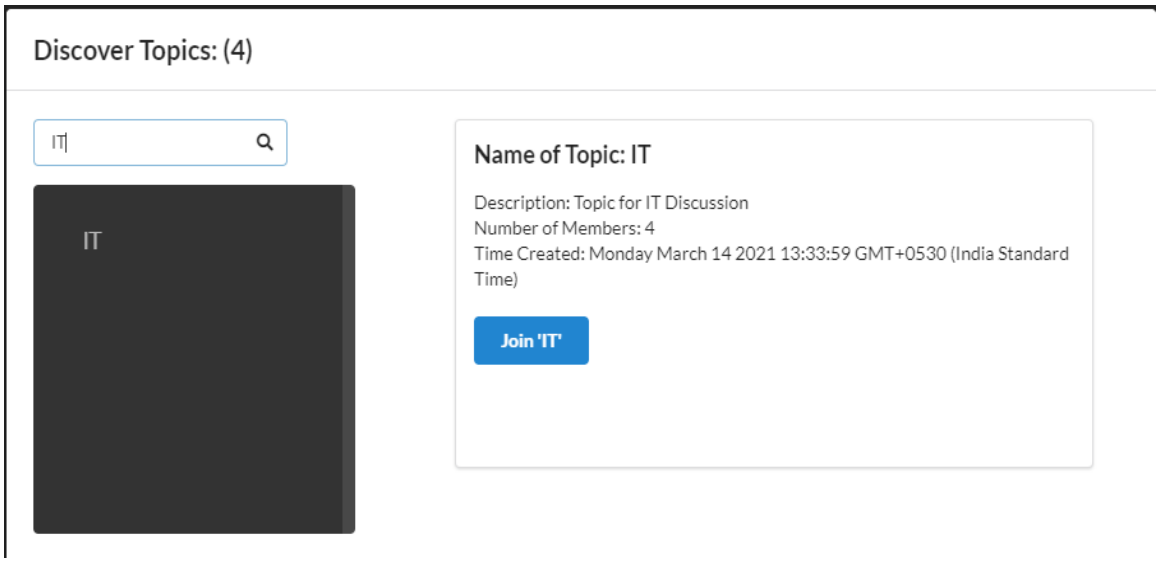

Fig 7.5 Displaying Topic Search Results

Case 2: No Results

Attribute	Value
Form	List of Topics
Action	Filtering topics whose name matches the user query.
Expected results	It displays no topics.

Table 7.6: Empty Search Results

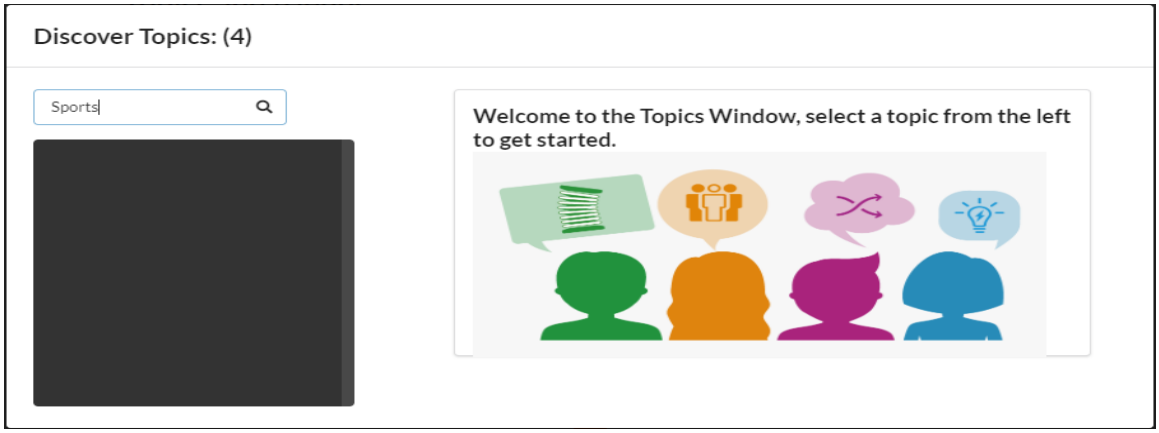

Fig 7.6: No Topics Displayed

7.4Message Input

Case 1: Message form contains some data

Attribute	Value
Form	Message Form
Action	Checks if the message form was given any
Expected results	Accepts the message input

Table 7.7: Message Form contains some data

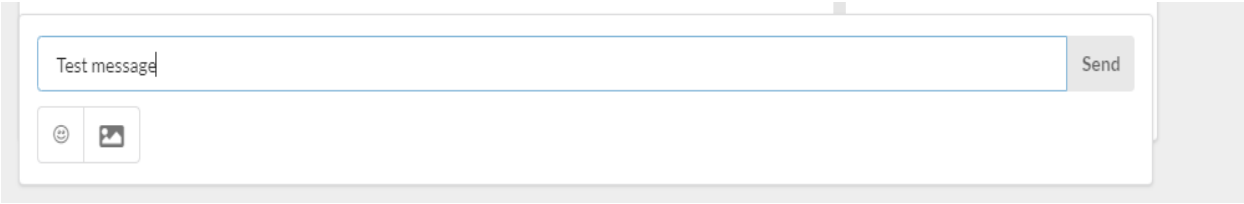

Fig 7.7: Message Form Accepts Input

Case 2: Message form contains no data

Attribute	Value
Form	Message Form
Action	Checks if the message form was given any
Expected results	Form does not accept the input when user tries to send

Table 7.8: Message Form contains no data

Fig 7.8: Message Form Invalid Input

Chapter-8
Conclusion

The main goal of this project is to provide instantaneous communication and interaction between users.

The project already implements most of the features or functionalities available in many of the commonly used existing communication platforms. It is possible to include additional features in the future such as individual message deletion, promotion of regular users to admin, more media file support (audio or video), voice or video call integration or more customization options for the users. The project can further, be hosted and deployed to the internet, allowing any users to access it remotely regardless of geographical locations or physical barriers.

To finalize and conclude, this project hopes to provide a user-friendly, pleasant experience of basic communication features involving real-time principles.

Bibliography

1 Slack.com

2 Telegram.org

3 Text books:

a. Marijn Haverbeke, "Eloquent JavaScript: A Modern Introduction to Programming", 3rd Edition, 2019.

b. Shama Hoque, "Full-Stack React Projects", 2nd Edition, Packt Publishing, 2020.

c. Harmeet Singh & Mayur Tanna, "Web Applications with React and Firebase", Packt Publishing, 2018.