3.2. Block Diagram of Proposed System.

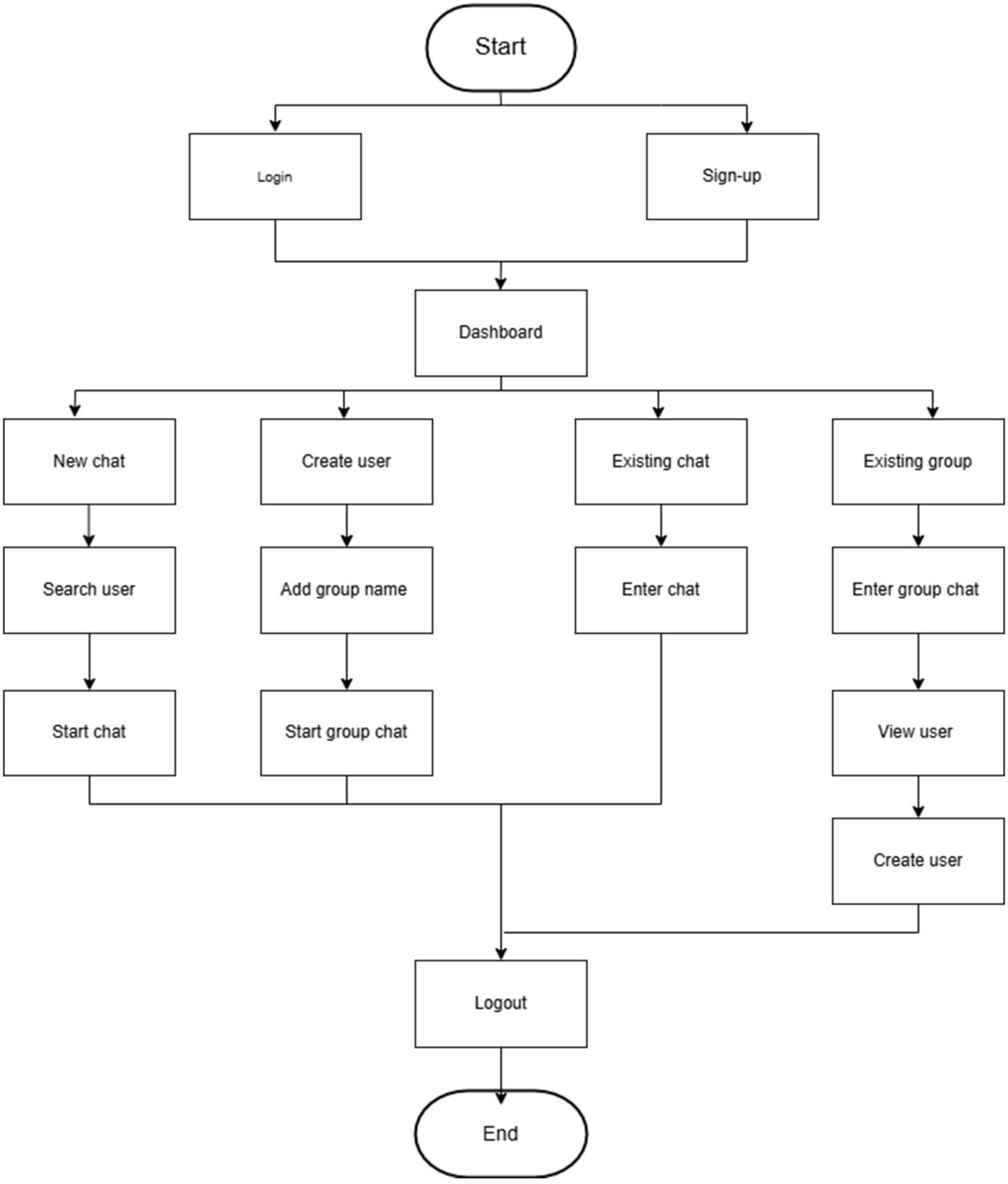

Figure 3.1. Block diagram of proposed system

3.3. Schema Representation.

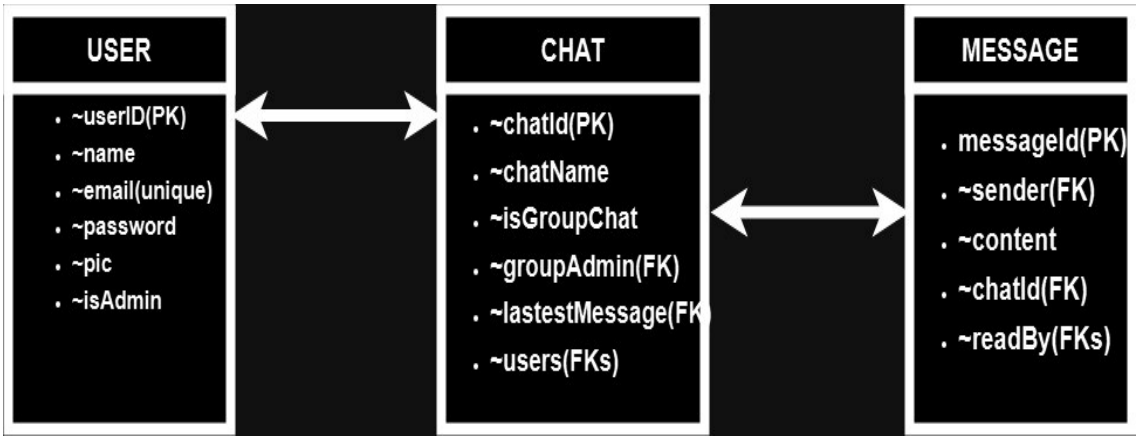

Figure 3.2. Schema Representation

Chapter 4

Technology Used

4.1. Mongo DB

MongoDB is used to store and manage data related to users, messages, and chat rooms in a flexible, scalable manner. Since it is a NoSQL database, it allows storing data in the form of documents (JSON-like objects), making it ideal for handling real-time, unstructured data such as chat messages.

4.2. Express.js

Express.js is used to handle the server-side logic, routing, and APIs for the chat application. It provides a lightweight framework to manage HTTP requests and build the RESTful APIs needed for handling user authentication, message storage, and chat functionality.

4.3. React.js

React is used to build the front-end of the chat application. It enables the creation of an interactive and dynamic user interface that allows users to send and receive messages in real-time, view notifications, and manage chat rooms.

4.4. Node.js

Node.js is used to build the server-side of the application. It serves as the runtime environment that executes JavaScript code on the server and enables communication between the front end (React) and the back end (Express, MongoDB). Additionally, Node.js is responsible for handling WebSocket connections to enable real-time chat functionality.

4.5. Socket.io

Socket.io is used to enable real-time, bi-directional communication between the client and the server. This library is essential for creating the core functionality of the chat application—sending and receiving messages in real time without refreshing the page

4.6. Requirement and Specification

4.6.1. Hardware

- **Processor:** A Pentium 4 processor or higher is recommended.
- **Memory:** It requires a minimum of 1MB of RAM, but it is recommended to have at least 2GB or more.
- **Hard drive:** Python requires a minimum of 4GB of free hard drive space.
- **Operating system:** Compatible with windows operating system.

4.6.2. Software

- Python IDE (Jupyter Notebook)
- Terminal (Anaconda prompt)
- Browser Google Chrome

4.6.3. Programming Language and libraries used

- MERN Stack Components: MongoDB, Express.js, React.js, Node.js
- WebSocket Library: Socket.io
- Node Package Manger
- Other Packages: Json Web Token, BCrypt, cookie, DotEnv
- Styling Components: Chakra Ui, Tailwind

Chapter 5

Result

5.1. Project Interface

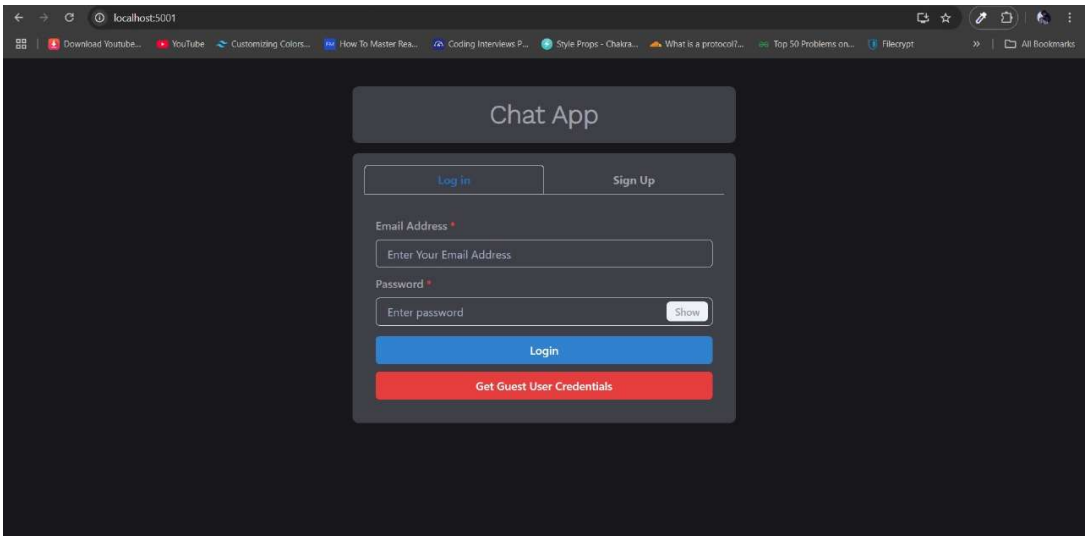

Fig 5.1: Login Page

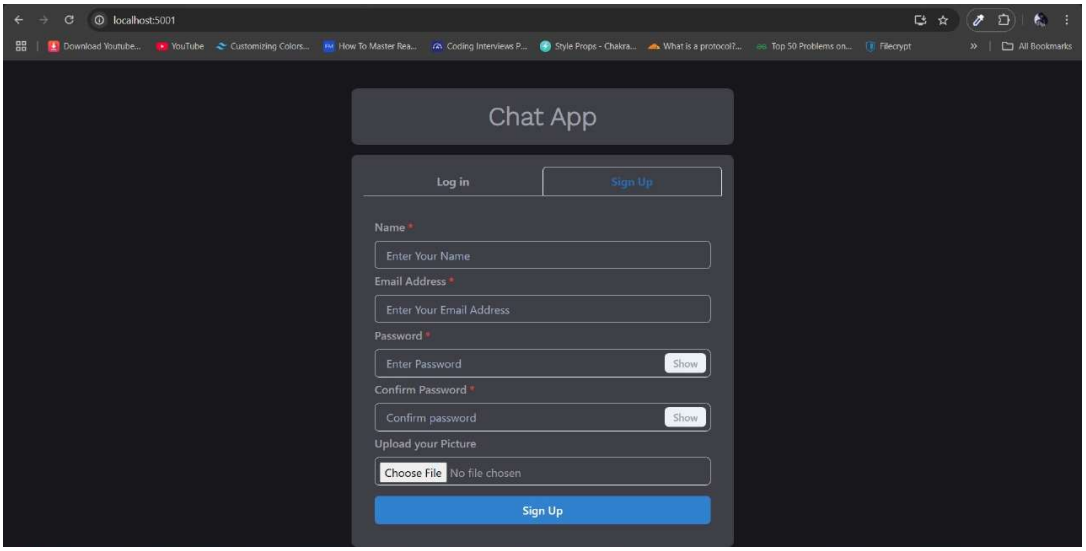

Fig 5.2: Sign Page

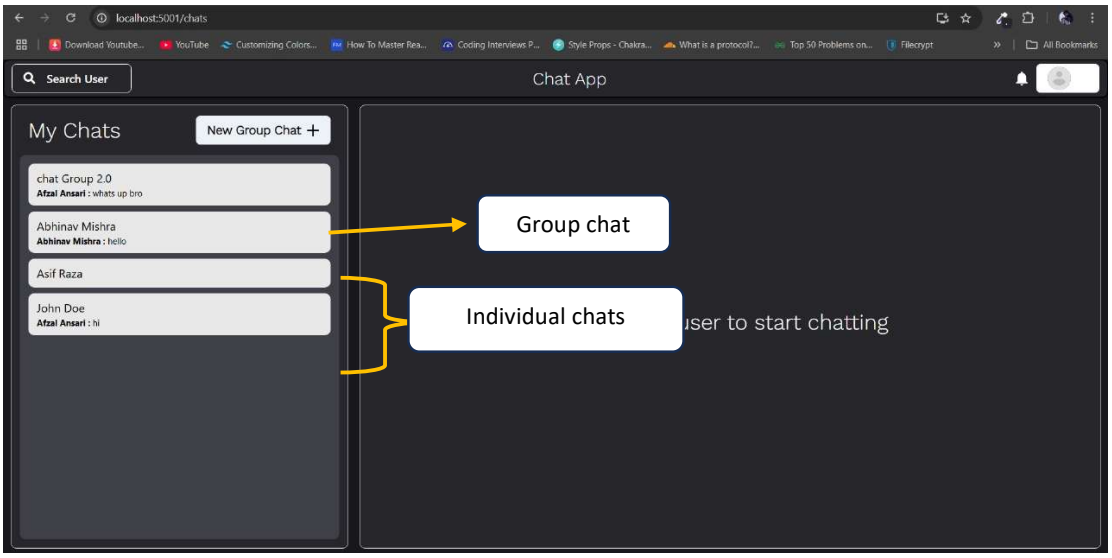

Fig 5.3: Dashboard Page

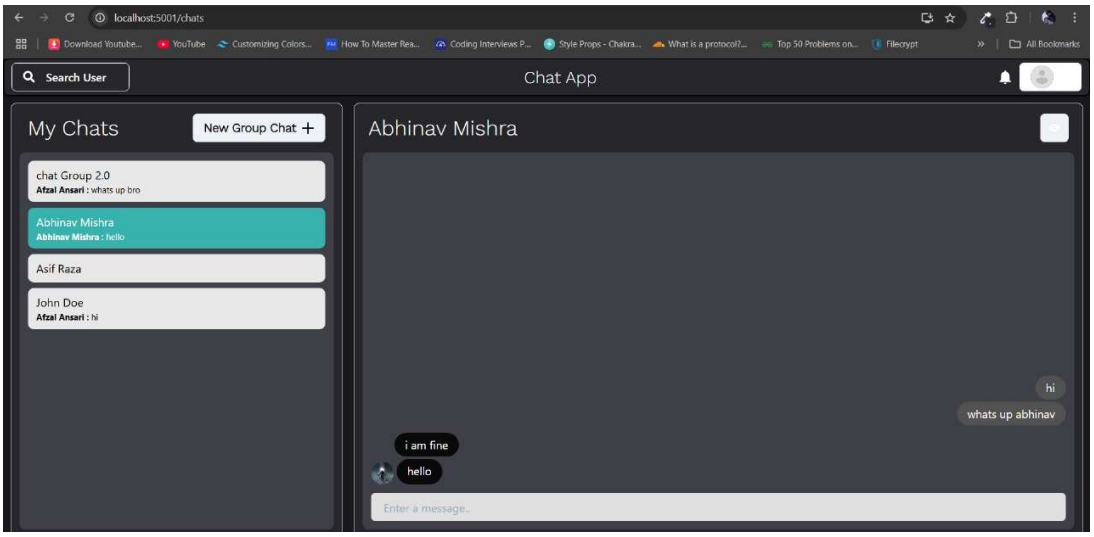

Fig 5.4: Existing Chats

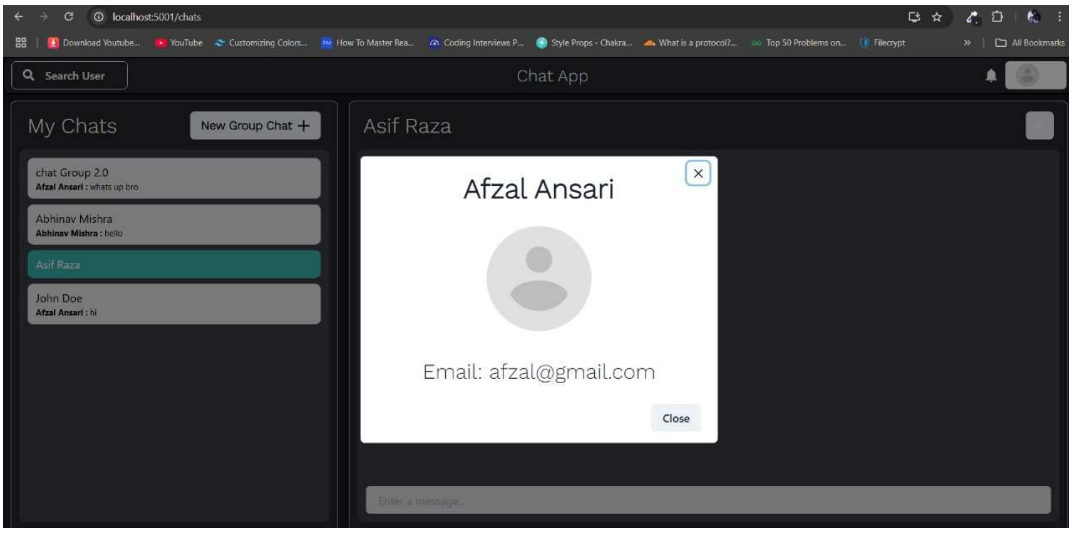

Fig 5.5: Sender's Profile

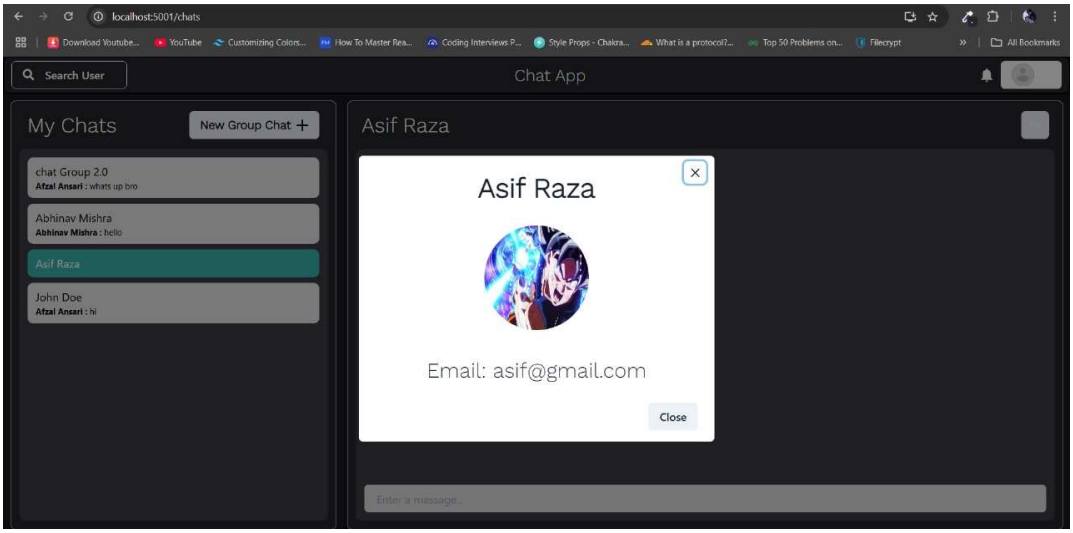

Fig 5.6: Receiver's Profile

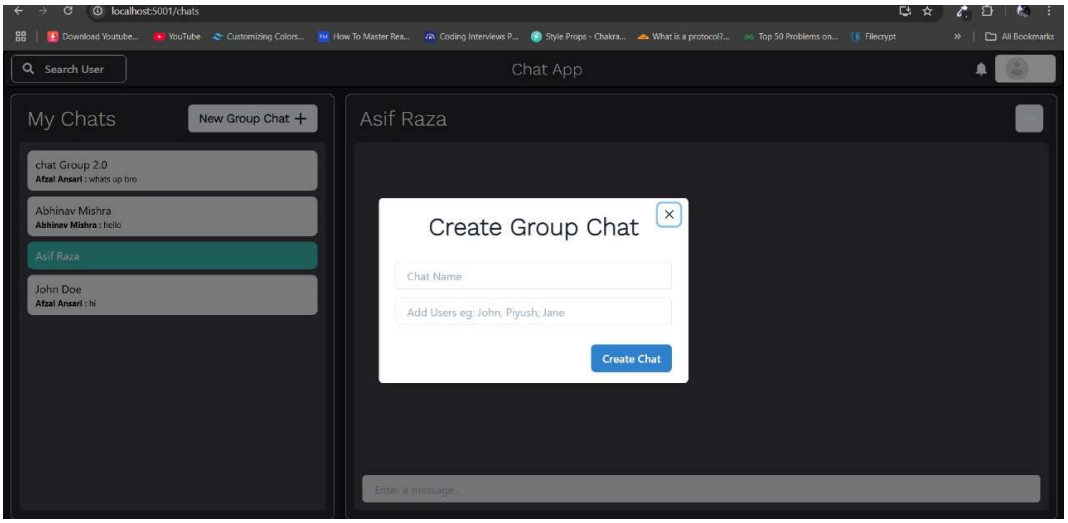

Figure 5.7: Option to create group chat

5.2. Conclusion

In conclusion, the development of a real-time chat application using the MERN stack and Socket.IO demonstrates the power of modern web technologies in addressing the need for instantaneous and reliable communication. The MERN stack's flexibility, scalability, and efficiency, combined with Socket.IO's real-time capabilities, enable the creation of robust and engaging communication platforms.

This project not only highlights the practical implementation of cutting-edge web development tools but also provides insights into overcoming common challenges in real-time application development, such as managing high user loads, ensuring data security, and delivering a superior user experience. By leveraging these technologies, the application has the potential to serve as a scalable solution for various communication needs, from personal interactions to enterprise-level collaboration.

The findings and outcomes of this project underscore the significance of adopting modern frameworks and libraries to stay aligned with industry demands. Future improvements could focus on enhancing scalability, integrating advanced features such as AI-powered chatbots, and optimizing performance to cater to even more complex use cases.

Appendix

Server File

```
const express = require("express");
const connectDB = require("./config/db");
const dotenv = require("dotenv");
const userRoutes = require("./routes/userRoutes");
const chatRoutes = require("./routes/chatRoutes");
const messageRoutes = require("./routes/messageRoutes");
const { notFound, errorHandler } = require("./middleware/errorMiddleware");
const path = require("path");

dotenv.config();

connectDB();
const app = express();

app.use(express.json()); // to accept json data

// app.get("/", (req, res) => {
//   res.send("API Running!");
// });

app.use("/api/user", userRoutes);
app.use("/api/chat", chatRoutes);
app.use("/api/message", messageRoutes);

// -------------------------deployment------------------------------

const __dirname1 = path.resolve();

if (process.env.NODE_ENV === "production") {
  app.use(express.static(path.join(__dirname1, "/frontend/build")));
```

```
  app.get("*", (req, res) =>
    res.sendFile(path.resolve(__dirname1, "frontend", "build", "index.html"))
  );
} else {
  app.get("/", (req, res) => {
    res.send("API is running..");
  });
}

// --------------------------deployment------------------------------

// Error Handling middlewares
app.use(notFound);
app.use(errorHandler);

const PORT = process.env.PORT;

const server = app.listen(
  PORT,
  console.log(`Server running on PORT ${PORT}...`.yellow.bold)
);

const io = require("socket.io")(server, {
  pingTimeout: 60000,
  cors: {
    origin: "http://localhost:3000",
    // credentials: true,
  },
});

io.on("connection", (socket) => {
```

```
  console.log("Connected to socket.io");
  socket.on("setup", (userData) => {
    socket.join(userData._id);
    socket.emit("connected");
  });

  socket.on("join chat", (room) => {
    socket.join(room);
    console.log("User Joined Room: " + room);
  });
  socket.on("typing", (room) => socket.in(room).emit("typing"));
  socket.on("stop typing", (room) => socket.in(room).emit("stop typing"));

  socket.on("new message", (newMessageRecieved) => {
    var chat = newMessageRecieved.chat;

    if (!chat.users) return console.log("chat.users not defined");

    chat.users.forEach((user) => {
      if (user._id == newMessageRecieved.sender._id) return;

      socket.in(user._id).emit("message recieved", newMessageRecieved);
    });
  });

  socket.off("setup", () => {
    console.log("USER DISCONNECTED");
    socket.leave(userData._id);
  });
});
```

DataBase Connection

```
const mongoose = require("mongoose");
const colors = require("colors");

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
  } catch (error) {
    console.error(`Error: ${error.message}`.red.bold);
    process.exit(1); // Exit with a non-zero status code to indicate an error
  }
};

module.exports = connectDB;
```

Database Schema

```
const asyncHandler = require("express-async-handler");
const Chat = require("../models/chatModel");
const User = require("../models/userModel");

//@description     Create or fetch One to One Chat
//@route           POST /api/chat/
//@access          Protected
const accessChat = asyncHandler(async (req, res) => {
  const { userId } = req.body;
```

```
if (!userId) {
  console.log("UserId param not sent with request");
  return res.sendStatus(400);
}

var isChat = await Chat.find({
  isGroupChat: false,
  $and: [
    { users: { $elemMatch: { $eq: req.user._id } } },
    { users: { $elemMatch: { $eq: userId } } },
  ],
})
  .populate("users", "-password")
  .populate("latestMessage");

isChat = await User.populate(isChat, {
  path: "latestMessage.sender",
  select: "name pic email",
});

if (isChat.length > 0) {
  res.send(isChat[0]);
} else {
  var chatData = {
    chatName: "sender",
    isGroupChat: false,
    users: [req.user._id, userId],
  };

  try {
    const createdChat = await Chat.create(chatData);
    const FullChat = await Chat.findOne({ _id: createdChat._id }).populate(
```

```
      "users",
      "-password"
    );
    res.status(200).json(FullChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
 }
});

//@description     Fetch all chats for a user
//@route           GET /api/chat/
//@access          Protected
const fetchChats = asyncHandler(async (req, res) => {
 try {
  Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
    .populate("users", "-password")
    .populate("groupAdmin", "-password")
    .populate("latestMessage")
    .sort({ updatedAt: -1 })
    .then(async (results) => {
      results = await User.populate(results, {
        path: "latestMessage.sender",
        select: "name pic email",
      });
      res.status(200).send(results);
    });
 } catch (error) {
  res.status(400);
  throw new Error(error.message);
 }
```

```
});

//@description     Create New Group Chat
//@route           POST /api/chat/group
//@access          Protected
const createGroupChat = asyncHandler(async (req, res) => {
 if (!req.body.users || !req.body.name) {
  return res.status(400).send({ message: "Please Fill all the feilds" });
 }

 var users = JSON.parse(req.body.users);

 if (users.length < 2) {
  return res
   .status(400)
   .send("More than 2 users are required to form a group chat");
 }

 users.push(req.user);

 try {
  const groupChat = await Chat.create({
   chatName: req.body.name,
   users: users,
   isGroupChat: true,
   groupAdmin: req.user,
  });

  const fullGroupChat = await Chat.findOne({ _id: groupChat._id })
   .populate("users", "-password")
   .populate("groupAdmin", "-password");
```

```
    res.status(200).json(fullGroupChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

// @desc    Rename Group
// @route   PUT /api/chat/rename
// @access  Protected
const renameGroup = asyncHandler(async (req, res) => {
  const { chatId, chatName } = req.body;

  const updatedChat = await Chat.findByIdAndUpdate(
    chatId,
    {
      chatName: chatName,
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!updatedChat) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(updatedChat);
  }
});
```

```
// @desc    Remove user from Group
// @route   PUT /api/chat/groupremove
// @access  Protected
const removeFromGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const removed = await Chat.findByIdAndUpdate(
    chatId,
    {
      $pull: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!removed) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(removed);
  }
});

// @desc    Add user to Group / Leave
// @route   PUT /api/chat/groupadd
// @access  Protected
```

```
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const added = await Chat.findByIdAndUpdate(
    chatId,
    {
      $push: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!added) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(added);
  }
});

module.exports = {
  accessChat,
  fetchChats,
  createGroupChat,
  renameGroup,
  addToGroup,
  removeFromGroup,
```

```
};
```

Message Schema

```
const mongoose = require("mongoose");

const messageSchema = mongoose.Schema(
 {
  sender: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
  content: { type: String, trim: true },
  chat: { type: mongoose.Schema.Types.ObjectId, ref: "Chat" },
  readBy: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
 },
 { timestamps: true }
);

const Message = mongoose.model("Message", messageSchema);
module.exports = Message;
```

User Schema

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");

const userSchema = mongoose.Schema(
 {
  name: { type: "String", required: true },
  email: { type: "String", unique: true, required: true },
  password: { type: "String", required: true },
  pic: {
   type: "String",
   required: true,
   default:
     "https://icon-library.com/images/anonymous-avatar-icon/anonymous-avatar-icon-25.jpg",
```

```
    },
    isAdmin: {
      type: Boolean,
      required: true,
      default: false,
    },
  },
  { timestaps: true }
);

userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

userSchema.pre("save", async function (next) {
  if (!this.isModified) {
    next();
  }

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

const User = mongoose.model("User", userSchema);

module.exports = User;
```

Routes

Chat Routes

```
const express = require("express");
const {
  accessChat,
```

```
  fetchChats,
  createGroupChat,
  removeFromGroup,
  addToGroup,
  renameGroup,
} = require("../controllers/chatControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/").post(protect, accessChat);
router.route("/").get(protect, fetchChats);
router.route("/group").post(protect, createGroupChat);
router.route("/rename").put(protect, renameGroup);
router.route("/groupremove").put(protect, removeFromGroup);
router.route("/groupadd").put(protect, addToGroup);

module.exports = router;
```

Message Routes

```
const express = require("express");
const {
  allMessages,
  sendMessage,
} = require("../controllers/messageControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/:chatId").get(protect, allMessages);
router.route("/").post(protect, sendMessage);
```

```
module.exports = router;
```

User Routes

```
const express = require("express");
const {
  registerUser,
  authUser,
  allUsers,
} = require("../controllers/userControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/").get(protect, allUsers);
router.route("/").post(registerUser);
router.post("/login", authUser);

module.exports = router;
```

Middlewares

```
const jwt = require("jsonwebtoken");
const User = require("../models/userModel.js");
const asyncHandler = require("express-async-handler");

const protect = asyncHandler(async (req, res, next) => {
  let token;

  if (
    req.headers.authorization &&
```

```
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      token = req.headers.authorization.split(" ")[1];

      //decodes token id
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      req.user = await User.findById(decoded.id).select("-password");

      next();
    } catch (error) {
      res.status(401);
      throw new Error("Not authorized, token failed");
    }
  }

  if (!token) {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
});

module.exports = { protect };

Error Middle Ware
const notFound = (req, res, next) => {
  const error = new Error(`Not Found - ${req.originalUrl}`);
  res.status(404);
  next(error);
};
const errorHandler = (err, req, res, next) => {
```

```
  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
  res.status(statusCode);
  res.json({
    message: err.message,
    stack: process.env.NODE_ENV === "production" ? null : err.stack,
  });
};

module.exports = { notFound, errorHandler };
```

Controllers

```
const asyncHandler = require("express-async-handler");
const Chat = require("../models/chatModel");
const User = require("../models/userModel");

//@description     Create or fetch One to One Chat
//@route           POST /api/chat/
//@access          Protected
const accessChat = asyncHandler(async (req, res) => {
  const { userId } = req.body;

  if (!userId) {
    console.log("UserId param not sent with request");
    return res.sendStatus(400);
  }

  var isChat = await Chat.find({
    isGroupChat: false,
    $and: [
      { users: { $elemMatch: { $eq: req.user._id } } },
      { users: { $elemMatch: { $eq: userId } } },
```

```
    ],
  })
    .populate("users", "-password")
    .populate("latestMessage");

  isChat = await User.populate(isChat, {
    path: "latestMessage.sender",
    select: "name pic email",
  });

  if (isChat.length > 0) {
    res.send(isChat[0]);
  } else {
    var chatData = {
      chatName: "sender",
      isGroupChat: false,
      users: [req.user._id, userId],
    };

    try {
      const createdChat = await Chat.create(chatData);
      const FullChat = await Chat.findOne({ _id: createdChat._id }).populate(
        "users",
        "-password"
      );
      res.status(200).json(FullChat);
    } catch (error) {
      res.status(400);
      throw new Error(error.message);
    }
  }
});
```

```
//@description     Fetch all chats for a user
//@route           GET /api/chat/
//@access          Protected
const fetchChats = asyncHandler(async (req, res) => {
  try {
    Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
      .populate("users", "-password")
      .populate("groupAdmin", "-password")
      .populate("latestMessage")
      .sort({ updatedAt: -1 })
      .then(async (results) => {
        results = await User.populate(results, {
          path: "latestMessage.sender",
          select: "name pic email",
        });
        res.status(200).send(results);
      });
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

//@description     Create New Group Chat
//@route           POST /api/chat/group
//@access          Protected
const createGroupChat = asyncHandler(async (req, res) => {
  if (!req.body.users || !req.body.name) {
    return res.status(400).send({ message: "Please Fill all the feilds" });
  }
```

```
  var users = JSON.parse(req.body.users);

  if (users.length < 2) {
    return res
      .status(400)
      .send("More than 2 users are required to form a group chat");
  }

  users.push(req.user);

  try {
    const groupChat = await Chat.create({
      chatName: req.body.name,
      users: users,
      isGroupChat: true,
      groupAdmin: req.user,
    });

    const fullGroupChat = await Chat.findOne({ _id: groupChat._id })
      .populate("users", "-password")
      .populate("groupAdmin", "-password");

    res.status(200).json(fullGroupChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

// @desc    Rename Group
// @route   PUT /api/chat/rename
// @access  Protected
```

```
const renameGroup = asyncHandler(async (req, res) => {
  const { chatId, chatName } = req.body;

  const updatedChat = await Chat.findByIdAndUpdate(
    chatId,
    {
      chatName: chatName,
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!updatedChat) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(updatedChat);
  }
});

// @desc    Remove user from Group
// @route   PUT /api/chat/groupremove
// @access  Protected
const removeFromGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const removed = await Chat.findByIdAndUpdate(
```

```
    chatId,
    {
      $pull: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!removed) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(removed);
  }
});

// @desc    Add user to Group / Leave
// @route   PUT /api/chat/groupadd
// @access  Protected
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const added = await Chat.findByIdAndUpdate(
    chatId,
    {
      $push: { users: userId },
    },
```

```
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!added) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(added);
  }
});

module.exports = {
  accessChat,
  fetchChats,
  createGroupChat,
  renameGroup,
  addToGroup,
  removeFromGroup,
};
```

Message Controller

```
const asyncHandler = require("express-async-handler");
const Message = require("../models/messageModel");
const User = require("../models/userModel");
const Chat = require("../models/chatModel");

//@description     Get all Messages
//@route           GET /api/Message/:chatId
```

```
//@access          Protected
const allMessages = asyncHandler(async (req, res) => {
  try {
    const messages = await Message.find({ chat: req.params.chatId })
      .populate("sender", "name pic email")
      .populate("chat");
    res.json(messages);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

//@description     Create New Message
//@route           POST /api/Message/
//@access          Protected
const sendMessage = asyncHandler(async (req, res) => {
  const { content, chatId } = req.body;

  if (!content || !chatId) {
    console.log("Invalid data passed into request");
    return res.sendStatus(400);
  }

  var newMessage = {
    sender: req.user._id,
    content: content,
    chat: chatId,
  };

  try {
    var message = await Message.create(newMessage);
```

```
    message = await message.populate("sender", "name pic").execPopulate();
    message = await message.populate("chat").execPopulate();
    message = await User.populate(message, {
      path: "chat.users",
      select: "name pic email",
    });

    await Chat.findByIdAndUpdate(req.body.chatId, { latestMessage: message });

    res.json(message);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

module.exports = { allMessages, sendMessage };
```

User Controllers

```
const asyncHandler = require("express-async-handler");
const User = require("../models/userModel");
const generateToken = require("../config/generateToken");

//@description     Get or Search all users
//@route           GET /api/user?search=
//@access          Public
const allUsers = asyncHandler(async (req, res) => {
  const keyword = req.query.search
    ? {
        $or: [
```

```
        { name: { $regex: req.query.search, $options: "i" } },
        { email: { $regex: req.query.search, $options: "i" } },
      ],
    }
  : {};

 const users = await User.find(keyword).find({ _id: { $ne: req.user._id } });
 res.send(users);
});

//@description     Register new user
//@route           POST /api/user/
//@access          Public
const registerUser = asyncHandler(async (req, res) => {
 const { name, email, password, pic } = req.body;

 if (!name || !email || !password) {
  res.status(400);
  throw new Error("Please Enter all the Feilds");
 }

 const userExists = await User.findOne({ email });

 if (userExists) {
  res.status(400);
  throw new Error("User already exists");
 }

 const user = await User.create({
  name,
  email,
  password,
```

```
    pic,
  });

  if (user) {
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(400);
    throw new Error("User not found");
  }
});

//@description     Auth the user
//@route           POST /api/users/login
//@access          Public
const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
```

```
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(401);
    throw new Error("Invalid Email or Password");
  }
});

module.exports = { allUsers, registerUser, authUser };
```

Reference

[1]. Mr.Sachin Bansal, Siddarth Dutt Sharma, Shahil Kumar Jha Sakshi Tomar, Roopali Pandey, "Real Time Chat Application", International Journal of Creative Research Thoughts (IJCRT), ISSN: 2320-2882, Issue 3, Volume 11, 3 March 2023.

[2]. Prof.Nargis Shaikh, Prof. Junaid Mandviwala, Harsh Mali, "Chat-With-Me: A MERN-based real-time chat Application", International Journal of Research and Analytical Reviews (IJRAR), ISSN 2349-5138, Issue 2, Volume 10, April 2023.

[3]. Abhishek Bedare, Harsh Jaiswal, Nehalika Kantule, "Lifeline messenger Real-Time Chat Application: Using Mern Stack", International Research Journal of Modernization in Engineering Technology and Science, e-ISSN: 2582-5208, Issue:05,Volume:05, May-2023.

[4]. Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, "An Overview of Real-Time Chat Application", International Journal for Research Trends and Innovation, ISSN: 2456-3315, Issue 6, Volume 7, 2022.

Appendix

Server File

```
const express = require("express");
const connectDB = require("./config/db");
const dotenv = require("dotenv");
const userRoutes = require("./routes/userRoutes");
const chatRoutes = require("./routes/chatRoutes");
const messageRoutes = require("./routes/messageRoutes");
const { notFound, errorHandler } = require("./middleware/errorMiddleware");
const path = require("path");

dotenv.config();

connectDB();
const app = express();

app.use(express.json()); // to accept json data

// app.get("/", (req, res) => {
//   res.send("API Running!");
// });

app.use("/api/user", userRoutes);
app.use("/api/chat", chatRoutes);
app.use("/api/message", messageRoutes);

// -------------------------deployment------------------------------

const __dirname1 = path.resolve();

if (process.env.NODE_ENV === "production") {
  app.use(express.static(path.join(__dirname1, "/frontend/build")));
```

```
  app.get("*", (req, res) =>
    res.sendFile(path.resolve(__dirname1, "frontend", "build", "index.html"))
  );
} else {
  app.get("/", (req, res) => {
    res.send("API is running..");
  });
}

// --------------------------deployment------------------------------

// Error Handling middlewares
app.use(notFound);
app.use(errorHandler);

const PORT = process.env.PORT;

const server = app.listen(
  PORT,
  console.log(`Server running on PORT ${PORT}...`.yellow.bold)
);

const io = require("socket.io")(server, {
  pingTimeout: 60000,
  cors: {
    origin: "http://localhost:3000",
    // credentials: true,
  },
});

io.on("connection", (socket) => {
```

```
  console.log("Connected to socket.io");
  socket.on("setup", (userData) => {
    socket.join(userData._id);
    socket.emit("connected");
  });

  socket.on("join chat", (room) => {
    socket.join(room);
    console.log("User Joined Room: " + room);
  });
  socket.on("typing", (room) => socket.in(room).emit("typing"));
  socket.on("stop typing", (room) => socket.in(room).emit("stop typing"));

  socket.on("new message", (newMessageRecieved) => {
    var chat = newMessageRecieved.chat;

    if (!chat.users) return console.log("chat.users not defined");

    chat.users.forEach((user) => {
      if (user._id == newMessageRecieved.sender._id) return;

      socket.in(user._id).emit("message recieved", newMessageRecieved);
    });
  });

  socket.off("setup", () => {
    console.log("USER DISCONNECTED");
    socket.leave(userData._id);
  });
});
```

DataBase Connection

```
const mongoose = require("mongoose");
const colors = require("colors");

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
  } catch (error) {
    console.error(`Error: ${error.message}`.red.bold);
    process.exit(1); // Exit with a non-zero status code to indicate an error
  }
};

module.exports = connectDB;
```

Database Schema

```
const asyncHandler = require("express-async-handler");
const Chat = require("../models/chatModel");
const User = require("../models/userModel");

//@description     Create or fetch One to One Chat
//@route           POST /api/chat/
//@access          Protected
const accessChat = asyncHandler(async (req, res) => {
  const { userId } = req.body;
```

```
if (!userId) {
  console.log("UserId param not sent with request");
  return res.sendStatus(400);
}

var isChat = await Chat.find({
  isGroupChat: false,
  $and: [
    { users: { $elemMatch: { $eq: req.user._id } } },
    { users: { $elemMatch: { $eq: userId } } },
  ],
})
  .populate("users", "-password")
  .populate("latestMessage");

isChat = await User.populate(isChat, {
  path: "latestMessage.sender",
  select: "name pic email",
});

if (isChat.length > 0) {
  res.send(isChat[0]);
} else {
  var chatData = {
    chatName: "sender",
    isGroupChat: false,
    users: [req.user._id, userId],
  };

  try {
    const createdChat = await Chat.create(chatData);
    const FullChat = await Chat.findOne({ _id: createdChat._id }).populate(
```

```
      "users",
      "-password"
    );
    res.status(200).json(FullChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
 }
});

//@description     Fetch all chats for a user
//@route           GET /api/chat/
//@access          Protected
const fetchChats = asyncHandler(async (req, res) => {
 try {
  Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
    .populate("users", "-password")
    .populate("groupAdmin", "-password")
    .populate("latestMessage")
    .sort({ updatedAt: -1 })
    .then(async (results) => {
      results = await User.populate(results, {
        path: "latestMessage.sender",
        select: "name pic email",
      });
      res.status(200).send(results);
    });
 } catch (error) {
  res.status(400);
  throw new Error(error.message);
 }
```

```
});

//@description     Create New Group Chat
//@route           POST /api/chat/group
//@access          Protected
const createGroupChat = asyncHandler(async (req, res) => {
 if (!req.body.users || !req.body.name) {
  return res.status(400).send({ message: "Please Fill all the feilds" });
 }

 var users = JSON.parse(req.body.users);

 if (users.length < 2) {
  return res
   .status(400)
   .send("More than 2 users are required to form a group chat");
 }

 users.push(req.user);

 try {
  const groupChat = await Chat.create({
   chatName: req.body.name,
   users: users,
   isGroupChat: true,
   groupAdmin: req.user,
  });

  const fullGroupChat = await Chat.findOne({ _id: groupChat._id })
   .populate("users", "-password")
   .populate("groupAdmin", "-password");
```

```
    res.status(200).json(fullGroupChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

// @desc    Rename Group
// @route   PUT /api/chat/rename
// @access  Protected
const renameGroup = asyncHandler(async (req, res) => {
  const { chatId, chatName } = req.body;

  const updatedChat = await Chat.findByIdAndUpdate(
    chatId,
    {
      chatName: chatName,
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!updatedChat) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(updatedChat);
  }
});
```

```
// @desc    Remove user from Group
// @route   PUT /api/chat/groupremove
// @access  Protected
const removeFromGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const removed = await Chat.findByIdAndUpdate(
    chatId,
    {
      $pull: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!removed) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(removed);
  }
});

// @desc    Add user to Group / Leave
// @route   PUT /api/chat/groupadd
// @access  Protected
```

```
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const added = await Chat.findByIdAndUpdate(
    chatId,
    {
      $push: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!added) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(added);
  }
});

module.exports = {
  accessChat,
  fetchChats,
  createGroupChat,
  renameGroup,
  addToGroup,
  removeFromGroup,
```

```
};
```

Message Schema

```
const mongoose = require("mongoose");

const messageSchema = mongoose.Schema(
 {
  sender: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
  content: { type: String, trim: true },
  chat: { type: mongoose.Schema.Types.ObjectId, ref: "Chat" },
  readBy: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
 },
 { timestamps: true }
);

const Message = mongoose.model("Message", messageSchema);
module.exports = Message;
```

User Schema

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");

const userSchema = mongoose.Schema(
 {
  name: { type: "String", required: true },
  email: { type: "String", unique: true, required: true },
  password: { type: "String", required: true },
  pic: {
   type: "String",
   required: true,
   default:
     "https://icon-library.com/images/anonymous-avatar-icon/anonymous-avatar-icon-25.jpg",
```

```
    },
    isAdmin: {
      type: Boolean,
      required: true,
      default: false,
    },
  },
  { timestaps: true }
);

userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

userSchema.pre("save", async function (next) {
  if (!this.isModified) {
    next();
  }

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

const User = mongoose.model("User", userSchema);

module.exports = User;
```

Routes

Chat Routes

```
const express = require("express");
const {
  accessChat,
```

```
  fetchChats,
  createGroupChat,
  removeFromGroup,
  addToGroup,
  renameGroup,
} = require("../controllers/chatControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/").post(protect, accessChat);
router.route("/").get(protect, fetchChats);
router.route("/group").post(protect, createGroupChat);
router.route("/rename").put(protect, renameGroup);
router.route("/groupremove").put(protect, removeFromGroup);
router.route("/groupadd").put(protect, addToGroup);

module.exports = router;
```

Message Routes

```
const express = require("express");
const {
  allMessages,
  sendMessage,
} = require("../controllers/messageControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/:chatId").get(protect, allMessages);
router.route("/").post(protect, sendMessage);
```

```
module.exports = router;
```

User Routes

```
const express = require("express");
const {
  registerUser,
  authUser,
  allUsers,
} = require("../controllers/userControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/").get(protect, allUsers);
router.route("/").post(registerUser);
router.post("/login", authUser);

module.exports = router;
```

Middlewares

```
const jwt = require("jsonwebtoken");
const User = require("../models/userModel.js");
const asyncHandler = require("express-async-handler");

const protect = asyncHandler(async (req, res, next) => {
  let token;

  if (
    req.headers.authorization &&
```

```
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      token = req.headers.authorization.split(" ")[1];

      //decodes token id
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      req.user = await User.findById(decoded.id).select("-password");

      next();
    } catch (error) {
      res.status(401);
      throw new Error("Not authorized, token failed");
    }
  }

  if (!token) {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
});

module.exports = { protect };

Error Middle Ware
const notFound = (req, res, next) => {
  const error = new Error(`Not Found - ${req.originalUrl}`);
  res.status(404);
  next(error);
};
const errorHandler = (err, req, res, next) => {
```

```
  const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
  res.status(statusCode);
  res.json({
    message: err.message,
    stack: process.env.NODE_ENV === "production" ? null : err.stack,
  });
};

module.exports = { notFound, errorHandler };
```

Controllers

```
const asyncHandler = require("express-async-handler");
const Chat = require("../models/chatModel");
const User = require("../models/userModel");

//@description     Create or fetch One to One Chat
//@route           POST /api/chat/
//@access          Protected
const accessChat = asyncHandler(async (req, res) => {
  const { userId } = req.body;

  if (!userId) {
    console.log("UserId param not sent with request");
    return res.sendStatus(400);
  }

  var isChat = await Chat.find({
    isGroupChat: false,
    $and: [
      { users: { $elemMatch: { $eq: req.user._id } } },
      { users: { $elemMatch: { $eq: userId } } },
```

```
    ],
  })
    .populate("users", "-password")
    .populate("latestMessage");

  isChat = await User.populate(isChat, {
    path: "latestMessage.sender",
    select: "name pic email",
  });

  if (isChat.length > 0) {
    res.send(isChat[0]);
  } else {
    var chatData = {
      chatName: "sender",
      isGroupChat: false,
      users: [req.user._id, userId],
    };

    try {
      const createdChat = await Chat.create(chatData);
      const FullChat = await Chat.findOne({ _id: createdChat._id }).populate(
        "users",
        "-password"
      );
      res.status(200).json(FullChat);
    } catch (error) {
      res.status(400);
      throw new Error(error.message);
    }
  }
});
```

```
//@description     Fetch all chats for a user
//@route           GET /api/chat/
//@access          Protected
const fetchChats = asyncHandler(async (req, res) => {
  try {
    Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
      .populate("users", "-password")
      .populate("groupAdmin", "-password")
      .populate("latestMessage")
      .sort({ updatedAt: -1 })
      .then(async (results) => {
        results = await User.populate(results, {
          path: "latestMessage.sender",
          select: "name pic email",
        });
        res.status(200).send(results);
      });
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

//@description     Create New Group Chat
//@route           POST /api/chat/group
//@access          Protected
const createGroupChat = asyncHandler(async (req, res) => {
  if (!req.body.users || !req.body.name) {
    return res.status(400).send({ message: "Please Fill all the feilds" });
  }
```

```
  var users = JSON.parse(req.body.users);

  if (users.length < 2) {
    return res
      .status(400)
      .send("More than 2 users are required to form a group chat");
  }

  users.push(req.user);

  try {
    const groupChat = await Chat.create({
      chatName: req.body.name,
      users: users,
      isGroupChat: true,
      groupAdmin: req.user,
    });

    const fullGroupChat = await Chat.findOne({ _id: groupChat._id })
      .populate("users", "-password")
      .populate("groupAdmin", "-password");

    res.status(200).json(fullGroupChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

// @desc    Rename Group
// @route   PUT /api/chat/rename
// @access  Protected
```

```
const renameGroup = asyncHandler(async (req, res) => {
  const { chatId, chatName } = req.body;

  const updatedChat = await Chat.findByIdAndUpdate(
    chatId,
    {
      chatName: chatName,
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!updatedChat) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(updatedChat);
  }
});

// @desc    Remove user from Group
// @route   PUT /api/chat/groupremove
// @access  Protected
const removeFromGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const removed = await Chat.findByIdAndUpdate(
```

```
    chatId,
    {
      $pull: { users: userId },
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!removed) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(removed);
  }
});

// @desc    Add user to Group / Leave
// @route   PUT /api/chat/groupadd
// @access  Protected
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  const added = await Chat.findByIdAndUpdate(
    chatId,
    {
      $push: { users: userId },
    },
```

```
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!added) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(added);
  }
});

module.exports = {
  accessChat,
  fetchChats,
  createGroupChat,
  renameGroup,
  addToGroup,
  removeFromGroup,
};
```

Message Controller

```
const asyncHandler = require("express-async-handler");
const Message = require("../models/messageModel");
const User = require("../models/userModel");
const Chat = require("../models/chatModel");

//@description     Get all Messages
//@route           GET /api/Message/:chatId
```

```
//@access          Protected
const allMessages = asyncHandler(async (req, res) => {
  try {
    const messages = await Message.find({ chat: req.params.chatId })
      .populate("sender", "name pic email")
      .populate("chat");
    res.json(messages);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

//@description     Create New Message
//@route           POST /api/Message/
//@access          Protected
const sendMessage = asyncHandler(async (req, res) => {
  const { content, chatId } = req.body;

  if (!content || !chatId) {
    console.log("Invalid data passed into request");
    return res.sendStatus(400);
  }

  var newMessage = {
    sender: req.user._id,
    content: content,
    chat: chatId,
  };

  try {
    var message = await Message.create(newMessage);
```

```
    message = await message.populate("sender", "name pic").execPopulate();
    message = await message.populate("chat").execPopulate();
    message = await User.populate(message, {
      path: "chat.users",
      select: "name pic email",
    });

    await Chat.findByIdAndUpdate(req.body.chatId, { latestMessage: message });

    res.json(message);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

module.exports = { allMessages, sendMessage };
```

User Controllers

```
const asyncHandler = require("express-async-handler");
const User = require("../models/userModel");
const generateToken = require("../config/generateToken");

//@description     Get or Search all users
//@route           GET /api/user?search=
//@access          Public
const allUsers = asyncHandler(async (req, res) => {
  const keyword = req.query.search
    ? {
        $or: [
```

```
        { name: { $regex: req.query.search, $options: "i" } },
        { email: { $regex: req.query.search, $options: "i" } },
      ],
    }
  : {};

 const users = await User.find(keyword).find({ _id: { $ne: req.user._id } });
 res.send(users);
});

//@description     Register new user
//@route           POST /api/user/
//@access          Public
const registerUser = asyncHandler(async (req, res) => {
 const { name, email, password, pic } = req.body;

 if (!name || !email || !password) {
  res.status(400);
  throw new Error("Please Enter all the Feilds");
 }

 const userExists = await User.findOne({ email });

 if (userExists) {
  res.status(400);
  throw new Error("User already exists");
 }

 const user = await User.create({
  name,
  email,
  password,
```

```
    pic,
  });

  if (user) {
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(400);
    throw new Error("User not found");
  }
});

//@description     Auth the user
//@route           POST /api/users/login
//@access          Public
const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
```

```
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(401);
    throw new Error("Invalid Email or Password");
  }
});

module.exports = { allUsers, registerUser, authUser };
```

Reference

[1]. Mr.Sachin Bansal, Siddarth Dutt Sharma, Shahil Kumar Jha Sakshi Tomar, Roopali Pandey, "Real Time Chat Application", International Journal of Creative Research Thoughts (IJCRT), ISSN: 2320-2882, Issue 3, Volume 11, 3 March 2023.

[2]. Prof.Nargis Shaikh, Prof. Junaid Mandviwala, Harsh Mali, "Chat-With-Me: A MERN-based real-time chat Application", International Journal of Research and Analytical Reviews (IJRAR), ISSN 2349-5138, Issue 2, Volume 10, April 2023.

[3]. Abhishek Bedare, Harsh Jaiswal, Nehalika Kantule, "Lifeline messenger Real-Time Chat Application: Using Mern Stack", International Research Journal of Modernization in Engineering Technology and Science, e-ISSN: 2582-5208, Issue:05,Volume:05, May-2023.

[4]. Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, "An Overview of Real-Time Chat Application", International Journal for Research Trends and Innovation, ISSN: 2456-3315, Issue 6, Volume 7, 2022.