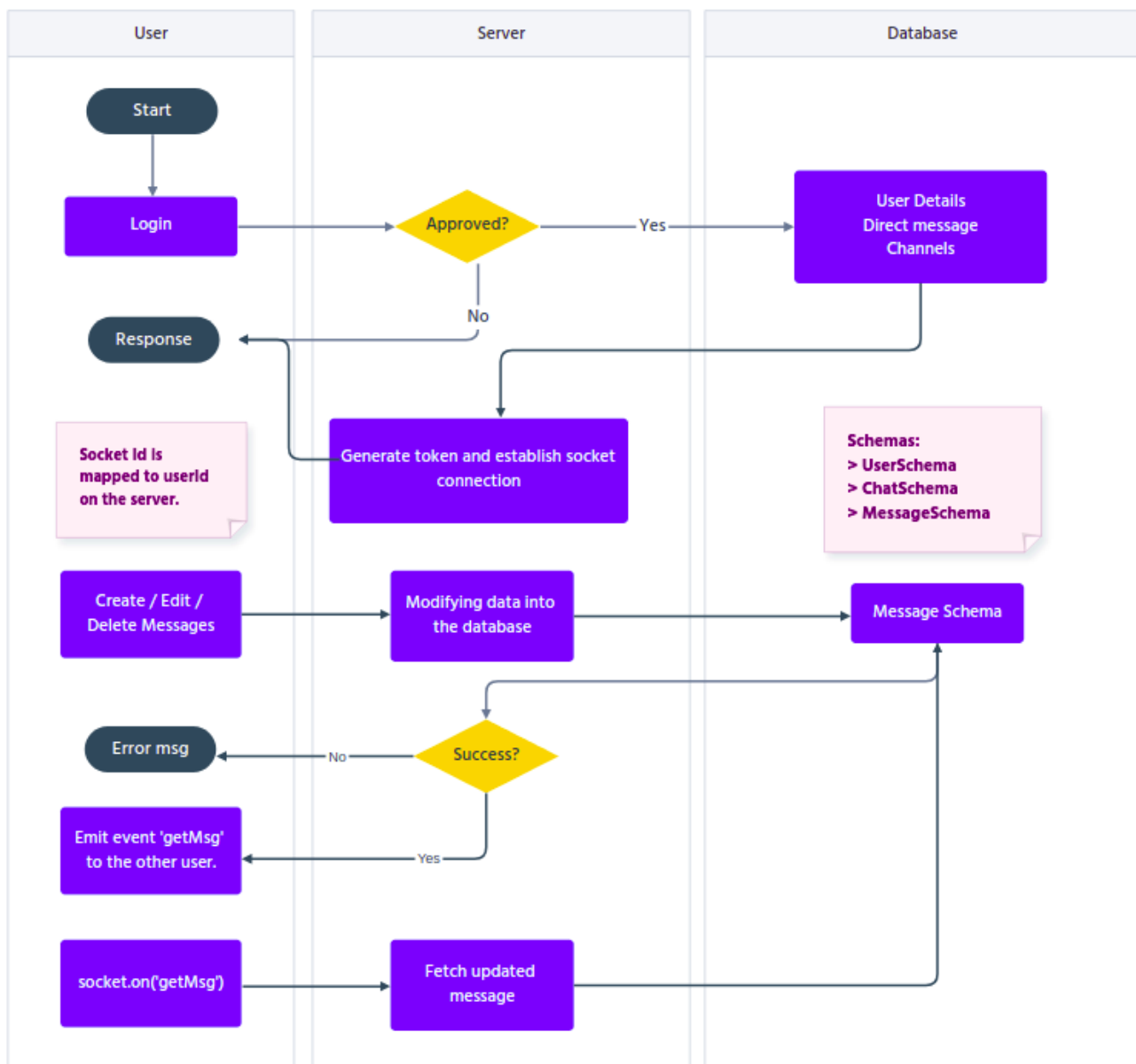


Documentation - Chat Application

Table of Contents

- Introduction
- Overall flow
- System Design
- Technology
- Database and Models
- APIs
- Socketevents
- Edgecases
- WireFrames

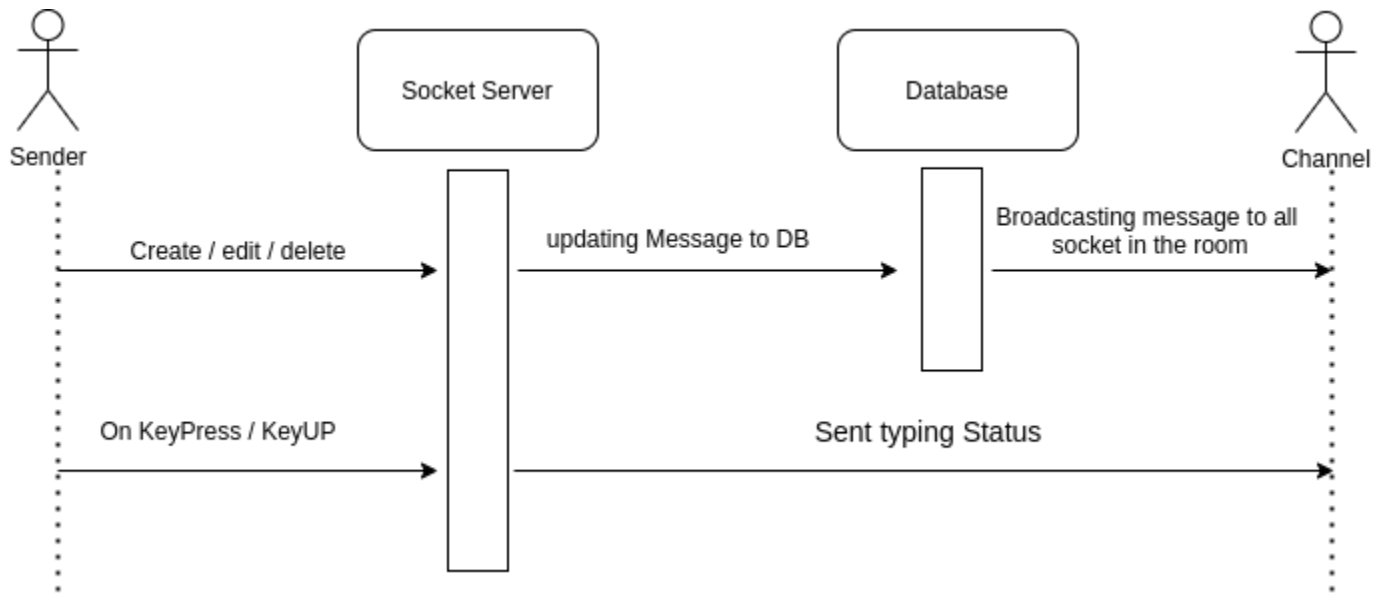
Overall flow:



System design

Basic message flow:

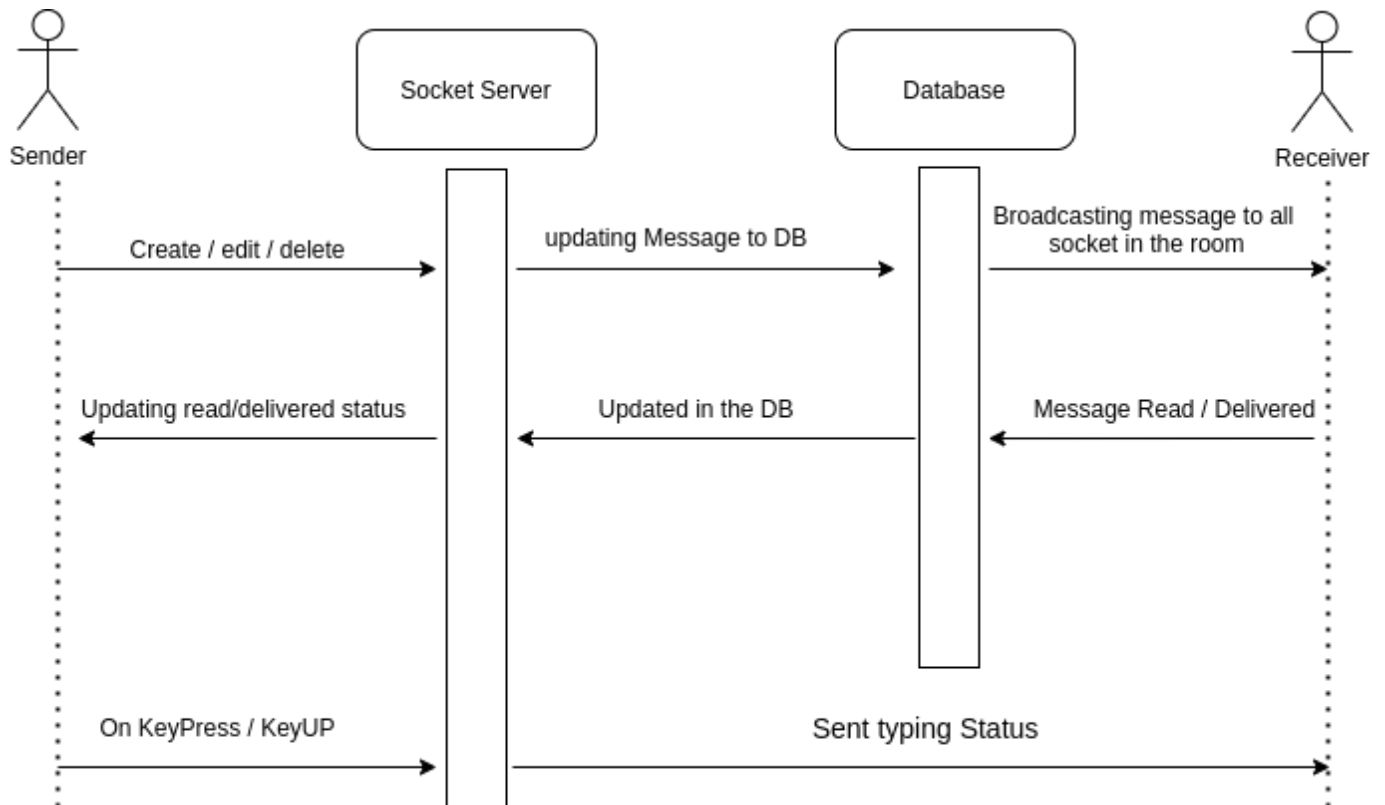
Case 1:



In case of direct messaging, consider two users A and B . When A sends message to B,

- A checks whether B is online or not from the user schema (refer database models).
- If B is online, the message is sent to B and the chat collection is updated (refer chat schema).
- SocketId's are mapped to the username and stored on the server.
- An acknowledgment is sent to A that B has received the message (delivered/seen).

Case 2:



In case of messaging in channels(group messaging), consider a user A. When A sends the message to a particular channel,

- The **channel-message** socket event is triggered (refer socket events) to store the message in the channel collection (refer database models).
 - An acknowledgment is sent to A (sender) that the message has been received to all the users connected to the channel.
-

Technology:

- Nodejs
 - Socketio
 - Reactjs
 - Mongodb (Mongoose)
-

Database and Models :

Collections:

- **User**

```
{
  name:String,
  email:String,
  password: String,
  socketid: String,
  is_online:Boolean,
  lastseen: String,
  directmessage:[
    {
      user_id:String,
      chat_id:String,
    }],
  channels:[{channel_id:Object._id}]
}
```

- **Chat**

```
{
  chatId:String,

  participants:[Object._id],

  type: Enum (Channel/Direct),

  name : String
}
```

- **Messages**

```
{
  _id: Object._id,

  chatId: Object._id,

  text: string,

  seen: [userId],

  delivered: [userId],

  timestamp: Date
}
```

API's :

Fetch User Registration Details API

TYPE: POST

URL : /api/register

HEADERS :

BODY :

```
{
  name: String,

  emailId: String,

  password: String
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "message": "User registered successfully!!"
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "409",
  "error" : "User already exists!!"
}

{
  "status" : "failure",
  "statusCode": "400",
  "error": "Bad payload!!"
}

{
  "status" : "failure",
  "statusCode": "500",
  "error": "Internal Server Error!!"
}
```

Description

- Check whether an email is already present or not.
- If not already exists saving email and password into the database.
- Your password will be salted 10 times and then it will be stored.

Fetch User Login Details API

TYPE : POST

URL : /api/login

HEADERS :

BODY :

```
{
  "emailId": String,
  "password": String
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "auth-token": String,
  "user": {
    "_id": Object._id,
    "userName": String,
    "email": String,
    "lastSeen": String,
    "online": Boolean
  }
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "404",
  "error": "User not Found!!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Incorrect credentials!!"
}
```

Description

- Check whether an email is already present or not. If your email already exists then checks your password with decrypted password.
- Users can access the application only if they provide the correct email and password.

- Users also will get a JWT token and the user details after the correct login.

Fetch User Profile Details API

TYPE : POST

URL : /profile/edit

HEADERS : auth-token

BODY :

```
{
  "profileImage": (File),
  "name": String,
  "phoneNumber": String,
  "status": String;
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "message": "Profile updated sucesfully"
}
```

ERROR RESPONSES :

```
{
  "status":"failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request!"
}
```

Description

- LoggedIn users can edit their profile by changing the name, profile image, mobile number, and status.

Creating new channel API

TYPE : POST

URL : /channel/newchannel

HEADERS : auth-token

BODY :

```
{
  "channelName": String,
  "userId": Object._id
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "channel": {
    "_id": "Object._id",
    "channelName": String,
    "participants": [{ "userId": "Object._id" }]
  }
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request to create new channel!!"
}

{
  "status" : "failure",
  "statusCode": "400",
  "error": "Bad payload"
}
```

```
{
  "status" : "failure",
  "statusCode": "409",
  "error": "Channel name already exists!"
}
```

Description

- Any valid user can create a channel.

Fetching the details of channel members - API

TYPE : POST

URL : /channel/join

HEADERS : auth-token

BODY :

```
{
  "channelId": Object._id,
  "userId": Object._id
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "message": "channel joined sucesfully"
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request"
}
```

```
}

{
  "status" : "failure",
  "statusCode": "400",
  "error": "Bad payload"
}

{
  "status" : "failure",
  "statusCode": "404",
  "error": "Channel not found!"
}
```

Description

- Adds members to channel.

Fetching the message- API

TYPE : POST

URL : /message

HEADERS : auth-token

BODY :

```
{
  "text": String,
  "senderid": "Object._id",
  "type": Enum("create/edit/delete/file)",
  "receiver_id": "Object._id",
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "message": "message posted sucesfully"
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request"
}

{
  "status" : "failure",
  "statusCode": "400",
  "error": "Bad payload"
}
```

Description:

- Store messages to the db.

To view channel details- API

TYPE : GET

URL : /channel/search

HEADERS : auth-token

BODY :

```
{
  searchKeys: String;
}
```

SUCCESS RESPONSE :

```
{
  "status" : "success",
  "statusCode": 200,
  "channels": [{
    "_id": "Object._id",
    "channelName": String,
    "participants": [userId: "Object._id"]
  }]
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request"
}

{
  "status" : "failure",
  "statusCode": "404",
  "error": "Channel not found!!"
}
```

Description:

- To get all the channels that matches the search pattern.

To view User details- API

TYPE : GET

URL : /user/details

HEADERS : auth-token

BODY :

```
{
  userId: 'Object._id';
}
```

SUCCESS RESPONSE :

```
{
  "status": "success",
  "statusCode": 200,
  "user": {
```

```
    "userId": "Object._id",
    "userName": String,
    "Email": String,
    "status": String
  }
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request"
}

{
  "status" : "failure",
  "statusCode": "404",
  "error": "User details not found!!"
}
```

Description:

- Fetching a particular user details using userId.

To view Chat details- API

TYPE : GET

URL : /getchat

HEADERS : auth-token

BODY :

```
{
  "chatId": Object._id,
  "timestamp": Date,
}
```

SUCCESS RESPONSE :

```
{
  "status" : "success",
  "statusCode": 200,
  "message": [
    {
      "_id": Object._id,
      "body": String,
      "delivered": Boolean,
      "seen": Boolean,
      "timestamp": Date,
    }
  ]
}
```

ERROR RESPONSES :

```
{
  "status": "failure",
  "statusCode": "500",
  "error" : "Internal server error!"
}

{
  "status" : "failure",
  "statusCode": "401",
  "error": "Unauthorized request"
}

{
  "status" : "failure",
  "statusCode": "404",
  "error": "chat not Found!!"
}
```

Description:

- Fetching direct chats/channels from database using chatId and userId. Using timestamp we can get all the messages after that timestamp.
- This api is called from the user whenever it receives a trigger for new messages.

Socket Events:

- **Event name : user-status**
 - Payload

- userName
 - timestamp
 - isOnline
- Triggered whenever a user connects or disconnects. Broadcasted to all the sockets including the one that is emitting it.
- **Event name : add-channel-members**
 - Payload
 - channelId
 - userId
- Add a new member to the channel and notify all the connected sockets in the room about the new member.
- **Event name : typing**
 - Payload
 - channelId
 - userName
- Notify the other end about the typing status.
- **Event name : getnewchat**
 - Payload
 - chatId
- **Event name : delivered**
 - Payload
 - chatId
 - userId
 - deliveredIndex
- Updating the delivered status of the messages for direct messages.
- **Event name : read**
 - Payload
 - chatId
 - userId

- readIndex
 - Updating the read status of the users in the DB, and conveying the changes to the other users if connected.
 - **Event name : new-direct message**
 - Payload
 - senderId
 - recieverId
 - Starting a new chat with a user and notify the user.
 - **Event name : acknowledgment**
 - Payload
 - chatId
 - userId
 - messageIndex
 - This event sends an acknowledgment back to the user from the server to notify that it successfully received the message.
 - Useful in handling connection loss or server errors.
-

Edgecases:

Handling traffic

Reconnect attempts at a random interval of time between 0-2 seconds, so that large traffic can be handled in case of server errors.

Storing messages in local storage

Saving all the msg in local storage, after getting the ack from the server, we will be clearing the local storage. If there is a network issue, the message will be stored in the local storage and will be sent once the connection re-establishes.

Multiple login

Mapping userId to a list of socketId's on server/DB for handling multiple logins.

Front end:

Reactjs is used to create the front-end application. Redux is used for state management.

- **Store :**

User:

- userId
- socketId
- listOfChannels
- directMessages
- email
- lastSeen

DirectMessage:

- messages
- messageId
- timestamp

Channel:

- channelId
- channelName
- membersId

CurrentChat:

- messages
- messageId
- timestamp
- senderReadIndex
- senderDeliveredIndex
- receiverReadIndex
- receiverDeliveredIndex


WireFrames :

Signin

Email

Password

Login


 **Hyperverge**


Channels


General

Product


Direct Messages

 John


 Peter Smith

 Richard


Product 22 +

 **Prem** 08.30 am


Guys are you done with for design document ?

 **Ashish** 08.32 am


We are making key points on Docs

 **Roshini** 08.33 am


Let me start with typing MD file

 **Prem** 08.35 am

Yeah sure, go ahead



 **Prem** 08.36 am


We have show our design document on monday to sai

 **Roshini** 08.37 am

Sounds great !

Message


 **Hyperverge**


Channels


General


Product


Direct Messages

 John


 Peter Smith

 Richard


 **John**
typing..

 **Kevin** 10.30 am

Hey John, how are you ?



 **John** 10.32 am

I'm good, what about you ?

 **Kevin** 10.33 am

I'm fine, whether your team completed your problem statement 3 ?

Message

20 / 20