Indian Institute of Technology, Kanpur
CS425: Computer Networks

ASSIGNMENT
2

Team:

Devashish Mehta	150226
Gurpreet Singh	150259
Prabhsimran Deep Singh	150493
Jaismin Kaur	150298

CatTalks: A centralized chat application

Todo list

Gurpreet: I think we should merge working and features

Mention the way we are encoding videos and sending it over the network (as broken down into simple frames)

1. Introduction

We have developed a text and video chat application named as "CatTalks". CatTalks is meant for person to person live video chat. It can also be used as an instant messaging service same as the messenger (by Facebook).

2. Features

In Catwalks, we have provided users with features as follows:

1. **Sign Up** Each User has to create an account before using the application. Currently, the user is supposed to provide his name, Username and a password for his account. This Username acts as its identity and everyone in the Network has a unique Username.
2. **Connecting with others** For chatting with a person, there has to a formal connection b/w the two. To connect with one user, other has to add his Username into his friends list. This will send a friend request to the other user who can choose to either accept or cancel the request.
3. **Backup for text messages** We have provided Backup For text messages. If the other User is not Online, text messages can still be sent to the User. These messages are kept stored until

the user comes online. We have also ensured that the user can differentiate b/w the read and Unread messages.

4. **Live Video Chat** Live Video Chat Option has also been provided for the users. If the Other User is Not Online, the request for the chat is just denied. The Other User can also chose to accept or deny the call. We haven't implemented voice service for the video chats.

3. Libraries and Languages Used

We have used different Languages for the frontend and Backend. We have also used Some libraries for the backend. The Libraries Involved in the Backend are as follows:

1. **Flask** Flask is a web application framework written in Python. A Web Application Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine.

   Flask is quite flexible. It doesn't require you to use any particular project or code layout. Flask-SocketIO gives Flask applications access to low latency bi-directional communications between the clients and the server. The client-side application can use any of the SocketIO official clients libraries in Javascript, C++, Java and Swift, or any compatible client to establish a permanent connection to the server. It enables us to use sockets easily for inter-user communication.

   Flask has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down. We have used flask SocketIO to send text messages as well as live video from one client to other. We have used rooms in SocketIO to manage interparty communication.

2. **MongoDb** MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemata.

   In CatTalks, mongoDb is used for storing user data, which includes text messages, message requests and login details of the user. We have chosen mongoDb for the following reasons:

   - The very basic feature of MongoDB is that it is a schema-less database, and therefore no schema migrations. Since MongoDB is schema-free, we can dynamically add components to the database without worrying about the effects.
   - It stores the data in the form of BSON (Binary JSON), ruby hashes etc, which helps to store the data in a very rich way while being capable of holding arrays and other documents.
   - Since, it is a NOSQL database, it is secure because no sql injection can be made.
   - The support for Sharding is one of its key feature. Sharding is the process of storing the data in different machines and MongoDB's ability to process the data, as and when the size of the data grows. This results in the horizontal scaling. With sharding, more amount of data can be written and read back as and when there is an increase in the data growth.

It provides us with low latency and higher security. In case the application grows, a SQL database would be faster, however all the features suffice a smooth run for an application of a small scale, such as ours.

For front end, we have used Javascript (with socket-io, as mentioned earlier). In order to capture video (using the user's webcam), we have used HTML5 framework to manipulate media devices.

4. Working

We use *emit* method to send data from client to server and vice-versa. Each client is assigned a different *room*.

1. **Register** If the user is not already registered, the *username* (used as key) is added to the database of users and session variables , *name* and *username*, are created for the user. The user is then redirected to the homepage of the application.
2. **Login** If the user has registered before, upon login, the user joins the room created for him and is sent to the homepage of the application where he can view his friends along with the messages shared with them.
3. **Send message request** In order to send message to another user the user has to add the other user to his friends. To send a request an emit is sent to the other user's room which is captured at the front-end of the other user. The requests received by user are maintained in a list. The user to whom the request is sent may choose to accept or reject it. If rejected, the *username* of the other user is popped out from the user's list of requests. Otherwise, the friend list of both the users are updated and they are prompted with a connect to other user message.
4. **Messages** A user may send a message to any of his friends. When a message is sent, an emit is sent to the rooms of both the users from the server. The emit is caught at the front-end and the message is displayed to both the users. If the other user is not online, the unread message is stored in the other user's $feed$ (in mongodb database). Every new message is highlighted, by showing user's block in bold and adding a split saying unread messages from below.
5. **Video chat** A user may engage in one video chat at a time. A user may video chat with another user only if the other user is online and not engaged in a video chat already. Upon making a request to video chat, the other user receives a message (an emit sent to the other user's room) if he want's to video chat or not. If the other user rejects the call, the user receives a notification of the same. Otherwise the feed from the user is sent via emit to the other user's room and vice-versa. The feed is sent by processing the camera input and sending screen-shots every 50ms (which creates the illusion of a continuous video).

5. Future Work

As mentioned earlier, we haven't implemented voice service along with live video chats. We hope to bring this feature into our application in future.

Currently, all our messages have to go through the server which can act as the bottleneck in case of an attack/breakdown. We hope to reduce this dependence on the server in future.