PLAN : B

Group Members: Ninad Sinha

Application link: <http://vast-hamlet-42923.herokuapp.com/login>

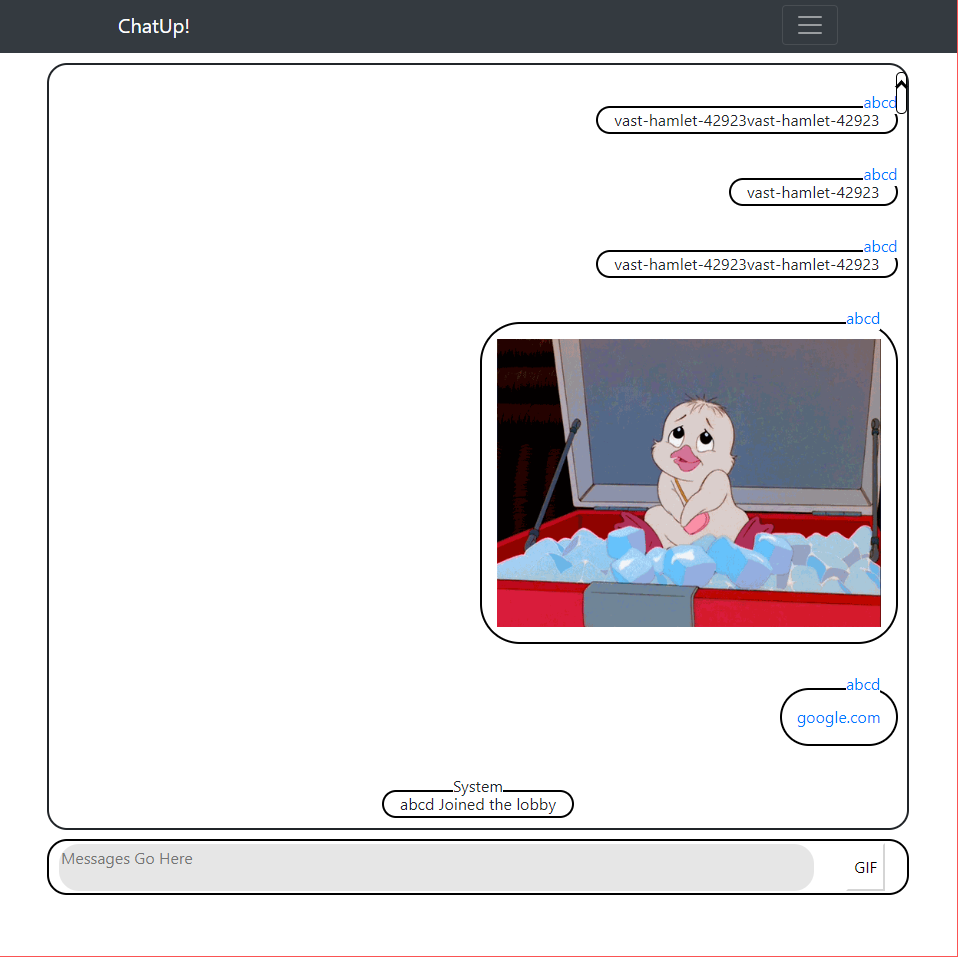
Github link: <https://github.com/AHarmlessPyro/chatApp>

## Technologies used

* Flask – basic server framework
* Flask-Login – provide user management and login
* Flask-SQLAlchemy – provide ORM
* Flask-Migrate – allow migrating easily to Heroku
* Flask-SocketIO (server) / SocketIO (client) – Allow for sending messages over websockets/html polling between sever and client with out needing to re-establish connection
* Flask-WTF – Used to create login and registration pages.
* Bootstrap 4: Used for layout and template used formed on it.
* Giphy API

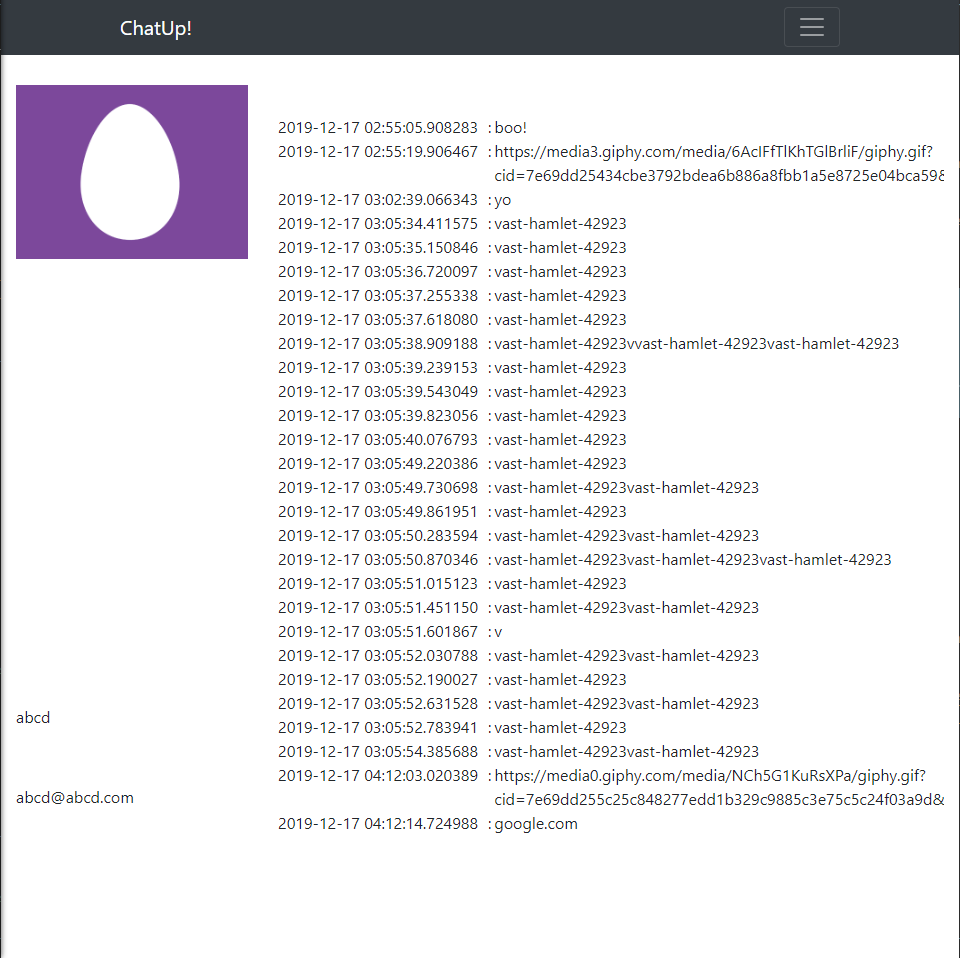
## Description

The application I’ve made is a simple messaging application. In brief, along with normal text chat, it supports passing of html links and images that are visible directly in the chat page itself, similar to a real messaging app and not just showing text links that need to be copied over.

In addition to this, the application also has support for the giphy api. So users can press the GIF button next to the text box and then search for their preference of GIFs so provided.

The app’s primary interface has a group chat like design where however many number of people can join in. By default, the app displays the last 10 messages in the public chat, but on demand, can load more messages. This has the advantage that when the user loads in, there is some minor context to the current chat and doesn’t take as long to load in case there are multiple messages.

Finally, the app also supports viewing history of messages made by a particular user.



On the backend, the storage support for distinguishing URLs, Images and messages is non-existent. That has both a disadvantage, in that each time the message is rendered, the text needs to re-examined to verify nature of the content, and an advantage in that the storage requirements are lessened and data consumed is reduced (as type of message is verified using HEAD requests and not full GET requests); essentially being a processor time vs storage question. In case of the deployment, storage is much smaller as we have 25 MBs of storage vs 550 hours of processor time.

## List of Controllers

There are essentially two kinds of controllers used here, one deriving from the normal routing of flask and the other from messaging for Flask-SocketIO/SocketIO.

Normal Controllers

* Login: Normal method for logging in. Also links to main chat page and user registration
* Logout: Self-Explanatory
* Index: Main Chat Page.
* Register: Page for providing User registration.
* User/<Name> : Provide list of messages for the given user

SocketIO controllers

* Connect: Initial connection event. Used for user registration for server/client messaging.
* getGIF: Used to do get requests on the Giphy API and then pass the URLs back to the the chat.
* sendMessagePublic: Send Message on Public Channels to everyone listening.
* getPrevious: Get the last 10 messages from the current message.

## List of Views

Each of the Normal Controllers (except for the logout) corresponds to a particular view.

## Tables

* User: Basic User Management table. Provides space for a hashed password, username, email, id and list of associated messages
* Messages: Table containing all messages. Contains an id, body, timestamp and the ID of the associated user for backreferencing purposes.

## References

* Flask Mega Tutorial from Miguel Grinberg (<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>)
* Socket IO documentation (<https://socket.io/docs/>) and Flask-Socket IO documentation (<https://flask-socketio.readthedocs.io/en/latest/>)
* SQL alchemy documentation (<https://docs.sqlalchemy.org/en/13/>)
* Innumerable Stack Overflow threads.
* Bare Bootstrap template (<https://startbootstrap.com/templates/bare/>)
* Twitter Egg ( https://static.independent.co.uk/s3fs-public/thumbnails/image/2017/04/01/17/twitter-egg.jpg)