

Proyecto chat con flask y websockets

Natalia Rios Agudelo
Alejandro Obando Gil

Profesor Cesar Jaramillo
Sistemas Distribuidos

Ingeniería de sistemas y computación
Universidad Tecnológica de Pereira
2020

1. PLANTEAMIENTO DEL PROBLEMA

El proyecto consiste en crear un servidor y un cliente (debe ser instanciado varias veces en máquinas diferentes, mínimo 3) de un programa de mensajería instantánea (CHAT).

Una persona se conecta al servidor utilizando el programa cliente. Cuando se conecta por primera vez al servidor debe crear una cuenta. El servidor almacenará los datos de dicha cuenta localmente y persistente (archivo o base de datos). La información requerida será: Nombres, apellidos, login, password, edad y género. Una vez registrada, una persona debe poderse conectar utilizando su usuario y contraseña. El servidor cuenta con diferentes salas de conversaciones. Estas salas son creadas por los clientes. Cuando se inicia el servidor, este cuenta con una sola sala que es la sala por defecto (las salas no serán persistentes, es decir, si un usuario crea una sala y luego se desconecta la sala será borrada). Cuando un cliente se conecta queda ubicado en dicha sala. Todos los mensajes que envíe un cliente se les reenvían a todos los clientes que se encuentren en dicha sala, al mensaje se le antepone el nombre del usuario que origina el mensaje.

Los clientes pueden enviar mensajes en el servidor. La forma de diferenciarlos es que los comandos iniciaran con el carácter '#’.

Los comandos disponibles son:

#cR <nombreSala>. Crear sala con el nombreSala. El servidor de forma automática ingresa a este cliente a la sala que creo.

#gR <nombreSala> Entrar a la sala nombreSala.

#eR. Salir de la sala en que se encuentra. El servidor enviara al cliente a la sala por defecto. Si el cliente ingresa este comando estando en la sala por defecto, no tendrá ningún efecto.

#exit. Desconectará al cliente del servidor

#lR Lista los nombres de todas las salas disponibles y el número de participantes de cada una.

#dR <nombreSala>. Elimina la sala nombreSala. Un cliente solo puede eliminar las salas que creo.

#show users: Muestra el listado de todos los usuarios en todo el sistema

#\private<nombreusuario>: Envía un mensaje privado a un usuario determinado sin importar en qué sala se encuentre

Requerimientos Básicos:

El servidor y los clientes deben estar en máquinas diferentes (mínimo3).

El proyecto debe ser desarrollado con hilos y sockets

Cuando un cliente se conecta con el servidor éste debe ser sincronizado con la hora del servidor.

El servidor tiene que ser implementado en lenguaje Python, los clientes pueden ser implementados en el lenguaje que desee (igual, sugiero Python).

Se requiere un cliente versión Móvil (puede ser nativo)

El sistema operativo es de su elección.

2. CONFIGURACIÓN DE LA APP

Herramientas de desarrollo

-) Lenguajes: Python 3(servidor), javascript(cliente)
-) Frameworks o librerías principales: Flask de Python, socketIO de Python, SQLAlchemy
-) Gestor de Base de datos: SQLPostgres
-) Entorno virtual: virtualEnvironment de python
-) Control de versiones: Git y GitHub
-) Entorno de desarrollo: Visual Estudio
-) Proveedor de servicio en la nube(Paas): Heroku
-) Otras tecnologías: Uso de websockets, hilos y sincronización de servidor y cliente

INSTALACION

Instalar pip de python:

```
python -m pip install -U pip
```

Crear entorno con virtual environment:

Instalar:

```
python -m install virtualenv
```

Ó en su defecto `py -m pip install --user virtualenv`

Crear carpeta de entorno virtual en el proyecto:

```
py -m venv env
```

Activar entorno virtual:

```
source env/Scripts/activate
```

Asignar python a entorno virtual:

```
Which python
```

```
Which python3
```

```
Deactivate
```

```
Which python
```

```
source env/Scripts/activate
```

```
Which python
```

Crear app en heroku:

Crear cuenta en heroku

Crear nueva aplicación en heroku

En heroku adicionamos postgresSQL en /proyecto/resource/Add-ons/Heroku postgres

Click en Provision

Instalar homebrew(solo para Linux):

```
Brew install postgres
```

Instalar postgres:

Descargar postgres desde su página oficial.

```
psql -v  
postgres -version
```

Crear variable de entorno de postgres

GIT:

Para trabajar con git en nuestro proyecto:

Primero inicializamos git dentro de la carpeta del proyecto

```
Git init
```

Archivo (gitignore.py):

En este archivo guardaremos todas las carpetas y archivos que queremos que git nos ignore en nuestro proyecto

Agregamos todos los archivos incluyendo el gitignore.py

```
Git add .
```

```
Git commit -m
```

Instalar flask:

```
Pip3 install flask
```

Crear la BD con PSQL y SQLAlchemy:

Instalar sqlalchemy:

```
pip3 install flask-sqlalchemy  
pip3 install psycopg2
```

Ruta de conexion con heroku postgres:

```
Psq1  
postgres://odzwjrzlprudi1:8317735ed8c2403e044449e353a227dbc9ca3a0afc17  
ba794f37cfc40420558d@ec2-54-161-150-170.compute-  
1.amazonaws.com:5432/d57q5bus761s43
```

Para ver las tablas que tenemos creadas:

```
\dt
```

```
\d (nombre de la tabla)
```

Para evitar errores con postgres, se debe trabajar con la misma versión de postgres tanto en el servidor(heroku) como instalado en el cliente(PC)

Instalar para manejo de sqlalchemy(error):

```
pip install pylint-flask
```

Agregar estas líneas de código al archivo settings.json de vscode:

```
{
  "python.linting.pylintArgs": [
    "--load-plugins",
    "pylint-flask"
  ]
}
```

3. DISEÑO DE LA APP

Tabla modelo entidad-relación:



4. REFERENCIAS

Pip: <https://pip.pypa.io/en/stable/installing/#installing-with-get-pip-py>

virtualenv: <https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/?highlight=virtualenv#installing-virtualenv>

Heroku: <https://www.heroku.com>

PostgreSQL: <https://www.postgresql.org/>

Git: <https://git-scm.com/downloads>

Flask: <http://flask.pocoo.org/>

WTForms: <https://wtforms.readthedocs.io/en/stable/>

SQLAlchemy: <https://docs.sqlalchemy.org/en/latest/>

Flask-SQLAlchemy: <http://flask-sqlalchemy.pocoo.org/2.3/>

Libreria hash:

https://passlib.readthedocs.io/en/stable/lib/passlib.hash.pbkdf2_digest.html

Flask login: <https://flask-login.readthedocs.io/en/latest/>

Flask message: <https://flask.palletsprojects.com/en/1.0.x/patterns/flashing/>

Flask socketio: <https://flask-socketio.readthedocs.io/en/latest/>

WebSocket compatibility: <https://caniuse.com/#search=Websocket>

Python time: <https://docs.python.org/3/library/time.html>

Generar clave: <https://flask.palletsprojects.com/en/1.0.x/quickstart/>

Install Heroku CLI: <https://devcenter.heroku.com/articles/heroku-cli>

GLOSARIO

Comandos de git

- * git init Oye Git, voy a usar estos documentos contigo ¿vale?
- * git add <file> Pasa los docs a staging area
- * git add . Pasa todos los archivos
- * git commit Pasa los docs de staging area a repository (Después de esto se te va a abrir el editor de código VIN en donde tendrás que escribir un comentario, si te quieres evitar abrir VIN entonces utiliza los siguientes comandos:)
- * git commit -m "comentario" Lo mismo que el commit regular, pero ahora no necesitas entrar a VIN
- * git status Ver en que status (wd, sa, r) están los docs
- * git push Subir los docs a un server (Github)
- * git pull Traer los docs de un server, traer los cambios de tus compañeros
- * git clone Hacerte una copia de lo que está en el server a tu PC
- * git checkout -- <file> Para revertir los cambios de los archivos
- * git diff <file> Para ver las diferencias hechas en los archivos
- * git branch Ver las ramas que hay ("master" es la rama default)
- * git branch "nombre" Crear una nueva rama
- * git checkout "nombre" Ir a una rama en específico

- * git config -- global user.email "email" Para configurar email del usuario
- * git config -- global user.name "nombre" Para configurar nombre del usuario

vin Es el editor de código de git desde la consola, ahí escribes un comentario para la nueva versión que estés versionando (si no te deja escribir presiona a letra i). Cuando termines presiona esc y luego :wq (write & quit) pasa salir.

.gitignore Es un archivo reservado de git que tenemos que crear si queremos decirle a git los archivos que no vamos a utilizar y así decida ignorarlos.

Escribe dentro de el archivo .gitignore los nombres de los archivos que desees ignorar.

Nota: Las carpetas se escriben solas y los archivos con su terminación.