PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanza (I/2017)
Tarea 7

1. Objetivos

- Interacción con APIs para la creación de un programa.
- Sintetizar conocimientos del curso.
- Uso de `flask` para la implementación de un servidor HTTP.

2. Introducción

Han pasado días desde tu rotundo éxito con **PrograPop** y te llega la espantosa noticia: el Malvado Dr. Mavrakis ha sufrido un grave accidente, se tropezó en el baño con un jabón y la caída provocó su muerte ~~por fin~~. Luego de su muerte, te das cuenta de que tal despiadado personaje se convirtió en una parte importante de tu vida, nadie trata de frustrar tus planes, ya no hay nadie urdiendo un plan malévolo en contra de la humanidad. En medio de este sentimiento de vacío tomas una decisión: usando tus vastos conocimientos sobre APIs decides crear un bot de **Telegram** en donde inmortalizarás para siempre la conciencia del malvado doctor (al más puro estilo Trascendence). Este bot estará conectado también con **GitHub**, de manera que puedas monitorear ciertos eventos de los repositorios a través del bot y replicar el supuesto trabajo del malvado doctor.

3. Problema

En esta tarea debes crear una aplicación que consuma las APIs de **GitHub** y **Telegram** para lograr notificar y comentar issues de **GitHub** al chat de **Telegram**. Además, se desea tener la capacidad de realizar una serie de acciones desde el chat de **Telegram** sin necesidad de acceder a la pagina de **GitHub**. Las notificaciones y acciones que debe soportar el bot son las siguientes:

1. Enviar un mensaje a través de Telegram apenas se publique una issue en algún repositorio previamente creado.
2. Publicar un comentario en alguna issue correspondiente en Github cuándo algún usuario de Telegram envíe el comando correspondiente.
3. Etiquetar una issue cuándo algún usuario de Telegram envíe el comando correspondiente.
4. Cerrar una issue cuándo algún usuario de Telegram envíe el comando correspondiente.

Cómo ya puedes imaginar, debes programar un servidor en `flask` que sirva de *pegamento* entre las dos APIs. En este servidor correrá el bot.

4. Especificaciones

4.1. API

Para esta tarea tendrán que usar la API de **GitHub**, **Telegram** y algún motor de búsqueda. Una API es una *Application Programming Interface* y consiste en un conjunto de recursos que permiten consumir funcionalidades de alguna plataforma en particular, de forma amigable para desarrolladores. Existen APIs de distintos servicios: **Facebook**, **Amazon Web Services (AWS)**, **Youtube**, etc. Como se dijo al principio, en esta tarea deberán usar la API de **GitHub**, **Telegram** y algún motor de búsqueda. Como se pueden imaginar, es de vital importancia que los clientes que consumen alguna API (es decir, ustedes) sepan exactamente qué parámetros deben entregar al realizar una solicitud, así como también qué se espera que retornen los recursos de la API. Por ello es que deben estudiar y comprender la documentación de la API que se va a consumir.

4.2. Github

Para esta tarea, deberán crear un repositorio **público** y registrarlo en un formulario que se habilitará en SIDING. Para que Github notifique la creación de issues en el repositorio, se debe hacer uso de los *webhooks* que ofrece. Un *webhook* es una herramienta que ofrece Github para enviar información a un *endpoint* de algún servidor web cuándo ocurre algún evento.

Para más información sobre el uso de *webhooks* se recomienda leer el siguiente enlace: `https://developer.github.com/webhooks/`. Para una explicación interactiva de cómo consumir con la API de Github se recomienda revisar este notebook de un semestre anterior.

Parte de las funcionalidades del bot a crear es poder responder issues, para esto el bot tiene que conectarse a **Telegram** y tener implementado una serie de comandos para que los usuarios de **Telegram** puedan mandar la respuesta desde el chat y el bot replique diche respuesta en forma de comentario en la issue de **Github**.

4.3. Telegram

Desde un chat de **Telegram**, se debe poder buscar, responder, etiquetar y cerrar una issue de **GitHub**. A continuación se detalla los comandos que el bot debe ser capaz de interpretar.

4.3.1. Comandos

Desde **Telegram** a **GitHub**:

- `/get #num_issue`: el bot debe enviar un mensaje con la issue correspondiente.
- `/post #num_issue *respuesta`: el bot debe publicar la respuesta que se indica en la issue correspondiente.
- `/label #num_issue label`: el bot debe asignar la etiqueta que se indica en la issue correspondiente.
- `/close #num_issue`: el bot debe cerrar la issue indicada.

Para todos los comandos, luego de efectuarse, el bot deberá enviar por **Telegram** un mensaje confirmando que la acción se ha realizado.

4.3.2. Formato de las notificaciones

En el chat de **Telegram** las notificaciones deben seguir el siguiente formato:

```
[<autor_issue>]

[#<num_issue> - <título_issue>]
```

`<Texto>`

`[Link: <link_issue>]`

4.3.3. Cómo interactuar con la API de Telegram

Es necesario estar registrado en el servicio de mensajería Telegram para el desarrollo de esta tarea.

Paso 1: BotFather En Telegram, deben buscar a BotFather e iniciar una conversación. Luego, deben escribir y enviar el comando `/newbot`. Una vez que BotFather les conteste, deben seguir las instrucciones:

- Elegir un nombre para el bot.
- Elegir un usuario para el bot (ojo que el nombre debe terminar en `bot`).

Apenas tengan todo listo, BotFather les responderá que su bot ya ha sido creado[1] y les entregará el `token`[2] de accesso.

Es importante configurar el about del bot para poder comprobar la autoría de ellos. Poniendo el nombre de usuario de Github basta y sobra. Para lograr lo anterior debes usar el comando `/setabouttext`.

Paso 2: API endpoint Este será el medio de comunicación entre el bot y el código que escribirán. Entonces, los mensajes enviados desde el servidor en `flask` a este *endpoint* serán acciones que el bot realizará. Finalmente, su servidor obtendrá este mensaje y enviará devuelta al link la respuesta que será visible en un chat de Telegram. El link tiene la siguiente estructura: `https://api.telegram.org/bot<token>/METHOD_NAME`.

Paso 3: Crear un webhook El uso de un *webhook* evita tener al bot *preguntando* continuamente al servidor de Telegram por nuevos updates. Mediante esta configuración, Telegram hará una *request* a su servidor cuándo ocurra algún evento con el bot. La información del evento está serializada en JSON.

Se debe hacer una *request* a la API de Telegram para indicar el *endpoint* con el cuál trabajará el *webhook*. Se debe usar el método `setWebhook` de la API de Telegram.

Los parámetros que recibe son:

- URL del servidor a la que se quiere enviar los eventos. Importante es que la URL empiece con `https://`.
- Otros parámetros opcionales que no son necesarios para el alcance de esta tarea.

Es posible deshacer los cambios con el método `deleteWebhook`, que no requiere parámetros.

Paso 4: Requests Todas las *requests* a la API de Telegram tienen que ser realizadas siguiendo el mismo formato señalado anteriormente: `https://api.telegram.org/bot<token>/METHOD_NAME`.
Por ejemplo: `https://api.telegram.org/bot123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11/getMe`

Los métodos de la API soportan los verbos `GET` y `POST` de HTTP. Las respuestas desde la API de Telegram vienen serializadas en JSON.

Se sugiere revisar el siguiente enlace para estudiar la API de Telegram: `https://core.telegram.org/bots/api`.

4.4. Heroku

Heroku es una plataforma que permite almacenar, compilar y correr aplicaciones en un servidor externo. Soporta varios lenguajes y es una herramienta muy útil cuando no se cuenta con un servidor propio para almacenar dichas aplicaciones.

[1] Pueden acceder a su bot con el siguiente link: telegram.me/inserte_bot_username
[2] Si se les pierde pueden generar otro con el comando /token

4.4.1. Cómo Hacer deploy

Paso 1: Instalar heroku Deben descargar desde `aquí` la **Heroku CLI** esto les permitirá acceder a comandos de heroku a traves de la consola.

Paso 2: Login Una vez instalada la aplicación deben crear una cuenta de usuario en la página de heroku y luego hacer login con el comando `heroku login`

Paso 3: Crear la app Una vez instalado, dentro de su repositorio del curso deben crear la aplicación en heroku para que esta pueda ser accedida luego a través de internet. Para esto deben usar el comando: `heroku create`

Paso 4: Cambiarle el nombre a su aplicación (opcional) Si lo desean, pueden cambiar el nombre de su aplicación con el siguiente comando: `heroku apps:rename nombre_nuevo --app nombre_antiguo`

Paso 5: Crear el código en su carpeta T07 En su carpeta T07 deben crear el código principal de su aplicación junto con un archvio `main.py` que importa su aplicación de flask y la corre.

Paso 6: Archivos de configuración Para que heroku sepa qué lenguaje, qué dependencias tiene la aplicación y cómo correrla deben crear 3 archivos. `requirements.txt`, `runtime.txt` y `Porcfile`.

- **runtime.txt** En este archivo debes escribir qué lenguaje utilizarás. Por lo general es sólo una línea que contiene `python-3.6.1`.
- **requirements.txt** En este archivo debes escribir qué librerias utilizaras. Para generar este archivo puedes instalar la libreria `pipreqs` con el comando `pip3 install pipreqs` y luego generar el archivo con el comando `pipreqs /path/a/T07`.
- **Procfile** En este archivo debes escribir básicamente qué comando deberá correr heroku para iniciar tu aplicación. En este archivo debes escribir lo siguiente: `web: flask run --host=0.0.0.0 --port=$PORT`. **Importante:** En esta ocasión estamos corriendo la aplicación directamente con el comando flask. Esto **no** es recomendable en un entorno de producción debido a que el servidor web de flask presenta un mal desempeño, como esta es una aplicación simple no exigiremos que sea ejecutado de otra forma pero recuerden que esta **no es la forma** en que se debe correr una aplicación flask en producción.
- **Variables de entorno** Para correr tu aplicación debes setear una sola variable de entorno llamada `FLASK_APP` la cual debe contener el nombre del archivo principal en que se encuentra tu aplicación. La forma de setear variables de entorno en heroku es mediante el siguiente comando: `heroku config:set --app <Nombre de su app>FLASK_APP=main.py`.

Paso 7: Pushea tu aplicación a heroku Ahora tu aplicación está lista para ver el mundo real, haz commit de tus cambios y pushealos a heroku. Como la aplicación se encuentra en una subcarpeta y no en la carpeta raíz del repositorio es necesario escribir el siguiente comando: `git subtree push --prefix Tareas/T07 heroku master`.[3] En caso de que sea necesario sobreescribir forzosamente el repositorio remoto de heroku se puede ejecutar el siguiente comando: `git push heroku 'git subtree split --prefix Tareas/T07 master':master --force`.

Paso 8: Acceder a la página web ¡Listo! Tu página se encuentra ahora abierta al mundo. Solo queda ingresar al link entregado por heroku y ver que todo esté funcionando correctamente. Si quieres ver los mensajes de error de tu aplicación puedes usar el siguiente comando: `heroku logs`.

[3] Este comando solo subirá los cambios al repositorio de heroku, es decir, no se verán reflejados en el github del curso

4.5. BONUS: Buscador (10 % de la nota final)

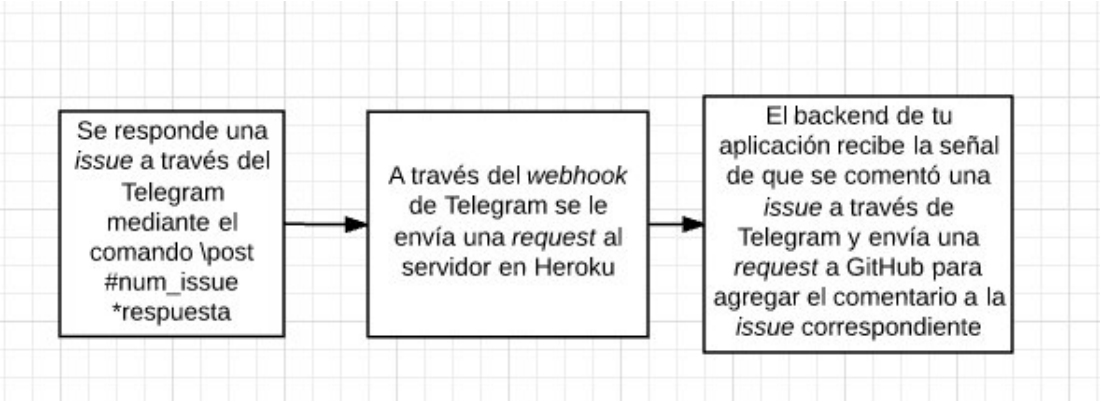
Cada vez que se abra una issue que contenga en su contenido algún bloque de código de *Markdown* que en su última línea contenga alguna excepción de Python, el bot debe comentar automáticamente un enlace que pueda ser de ayuda para solucionar el problema que se plantea. Para encontrar el enlace, deberá usar la API de algún motor de búsqueda a su elección[4]

En caso que la issue contenga mas de una excepción de Python, se debe escoger solo un error y proponer el primer link que se encuentre con dicho error. La forma de seleccionar el error queda su criterio.

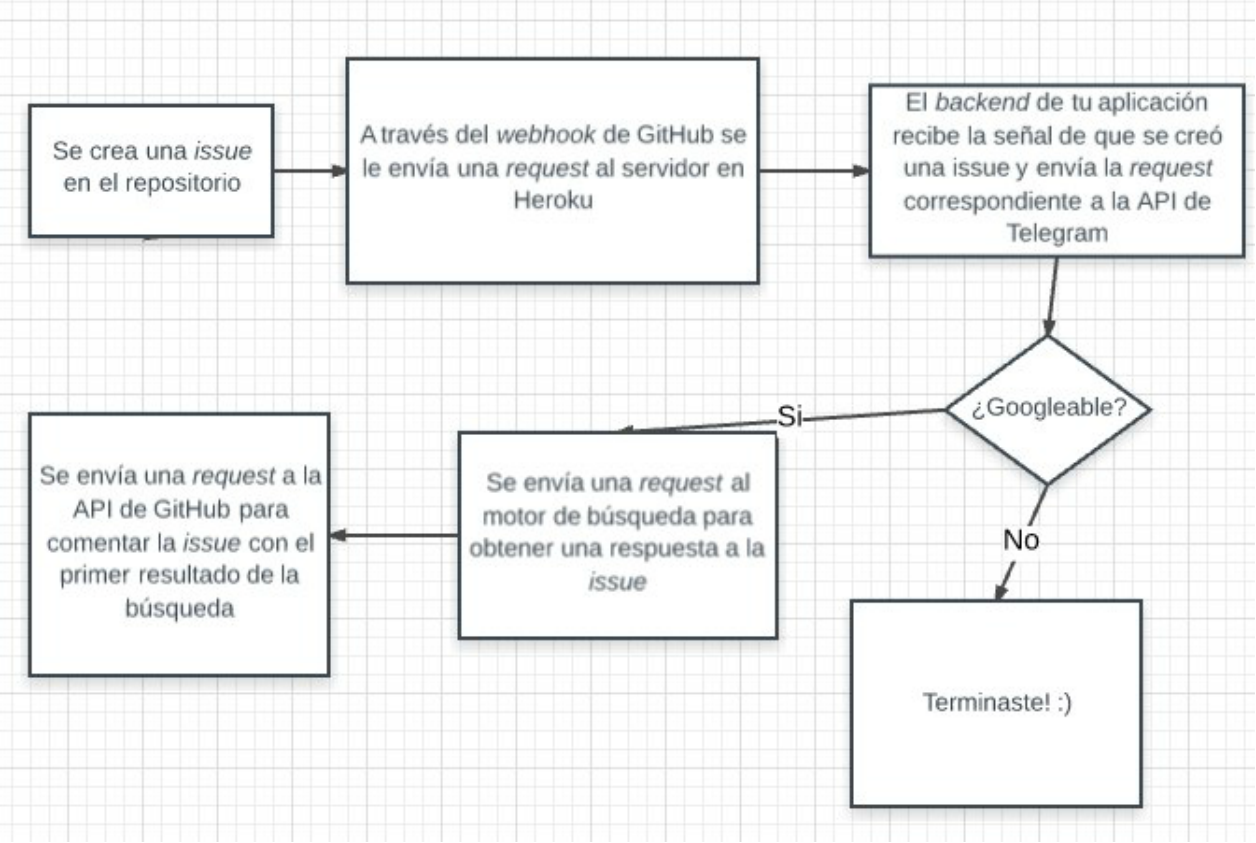
Luego del comentario automático, si la issue se cierra sin ningún otro comentario aparte del realizado por el bot o alguno del autor de la issue (puede dar las gracias), ponerle a ésta el label *Googleable*

5. Workflow

A modo de guía, a continuación se muestran dos diagramas de eventos que el bot debe de seguir cuando se reponde una issue desde **Telegram** o cuando se abre una issue en el repositorio.



[4] Bing, DuckDuckGo, Google, etc. Se recomiendan los dos primeros dado que tienen APIs simples y bien documentadas para buscar. Google tiene mejores resultados pero no es tan directo acceder a su motor de búsqueda programáticamente sin el uso de librerías especializadas.



6. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.5
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos `PEP8`.
- Si no se encuentra especificado en el enunciado, asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje[5] de su tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debe adjuntar un archivo `README.md` donde comente sus alcances y el funcionamiento de su sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*.
- Cree un módulo para cada conjunto de clases. Divídalas por las relaciones y los tipos que poseen en común. **Tareas que implementen el servidor o cliente en un sólo archivo serán castigadas fuertemente**.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por encima de ningún otro.

[5] Hasta −5 décimas.

7. Entrega

- **Fecha/hora:** 26 de Junio - 23:59 hrs
- **Lugar:** GIT - Carpeta: Tareas/T07

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).