Proceso de desarrollo de API Twitch Analytics

Entrega 1:

Primeros pasos: Configuración de entorno y obtención de credenciales

El primer paso que se dio a la hora de comenzar con el proyecto era, evidentemente, encontrar la forma de obtener los datos de Twitch. Para ello nos registramos en Twitch Developers y generamos un Token inicial que guardamos en un documento de texto para hacer las pruebas locales.

Programación en local:

Creamos una aplicación de escritorio que funcionaba mediante un menú que permitía ejecutar los tres casos de uso, simplemente para verificar que conseguíamos conectarnos a la API de Twitch y obtener los datos que nos interesaban. Cuando este punto estaba completado decidimos pasar a la parte del servicio web.

Obtención y configuración del servidor web:

Obtuvimos una VPS “gratuita” a través de OracleCloud y guardamos las claves públicas y privadas que nos dieron. Después hicimos una conexión ssh con la clave privada para crear un usuario.

Para levantar el servidor accedimos a la web de Oracle y permitimos el tráfico entrante desde la 0.0.0.0 (cualquier dirección ip) por el puerto 80 y autorizamos el tráfico saliente por el 443. Desactivamos los firewalls para permitir conexiones.

Continuamos la configuración instalándole php, httpd (Apache 2), dependencias de json, curl...

Método de trabajo:

Inicialmente trabajamos de manera remota sobre la máquina del servidor a través de una conexión ssh, gestionando los archivos por terminal y editando los ficheros con nano. Después variamos entre utilizar FileZilla y conexiones ssh en phpStorm para la edición de archivos mediante el IDE.

Grupo Tucán: Joritz Etxenike, Ander Etxabe, Aritz Huarte y Patxi Bueno

Tras separar los distintos casos de uso en archivos diferentes, dimos los últimos retoques al funcionamiento de la web, como analizar las diferentes respuestas de los códigos de estado http y el quitar las extensiones .php de la URL que se pide utilizando un .htaccess.

Una vez todo esto estaba terminado hicimos comprobaciones (tanto manuales como usando PostMan) de los distintos endpoints. Al ver que el funcionamiento era correcto cerramos el proyecto y subimos todo a GitHub.

Problemas y deuda técnica:

- Control de versiones/GitHub: Gran parte del desarrollo se dió en el servidor de producción mediante conexiones ssh en vez de trabajar en los repositorios locales, y la subida de archivos al servidor se hacía mediante FileZilla o funcionalidades del IDE. Esta forma de trabajar no permite sacar provecho del mismo git y dificulta el reparto de tareas y el trabajo en paralelo entre otras cosas.
- Pruebas y desarrollo en producción: Consecuencia de lo anterior, una vez pudimos comprobar que conectábamos con la API de Twitch el resto del desarrollo y pruebas fue puramente en producción, por desconocimiento de métodos para trabajar en local en ese punto.
- Credenciales “hard-codeadadas”: Al principio incluso el token estaba escrito directamente en los programas, eso lo arreglamos con la función desarrollada en token.php. Las credenciales de Twitch siguen estando hard-codeadas en ese código.
- Base de Datos: dado que solo había que implementar GETs y pudimos resolver el filtrado en el Caso de uso 3, nuestra aplicación hace una comunicación y traspaso de información directo con la API de Twitch, pero como idea a futuro será casi obligatorio introducir una BBDD que nos ayude tanto a filtrar como almacenar información.

Entrega 2:

Primeros pasos:

Lo primero que hicimos fue arregla los problemas que teníamos en la entrega uno, el principal era que las respuestas que nuestra api devolvía no eran de tipo json, sino html. Para solucionar esto, tuvimos que añadir la siguiente cabecera: *header("Content-Type: application/json"); .* Una vez hecho esto, dividimos la tarea en diferentes issue.

Configuración del servidor:

Para añadir las funcionalidades pedidas, era necesario tener una base de datos que gestionase los usuarios y la cache del nuevo endpoint *top of the tops.* Decidimos descargar mariabd como base de datos para guardados.

Método de trabajo:

En un inicio estuvimos trabajando en local, pero dado que la bbdd estaba en el servidor, las comprobaciones sobre la misma no eran cómodas hacerlas en local. Decidimos codificar todo lo que pudiésemos en local, sin comprobar las consultas, estas las comprobamos al desplegar el código. Para pasar el código a producción volvimos a usar filezilla y phpstrom. Además, decidimos cambiar la estructura dentro de la api para la facilitar el trabajo en paralelo.

Grupo Tucán: Joritz Etxenike, Ander Etxabe, Aritz Huarte y Patxi Bueno

Problemas y deuda técnica:

- Control de versiones: Hemos mejorado el control de versiones desde la práctica uno, pero, aun así, se nos hace difícil usar git. Además, es necesario que trabajemos con diferentes branches para evitar problemas y luego hacer merges mejor.
- Problemas con sql en php: Hemos tenido varios problemas relacionados a como generar consultas desde el php, y como estas al no funcionar nos ha costado encontrar los errores, ya que no aparecen directamente, son errores entre la sintaxis de php y sql.
- Problemas en la reorganización de los ficheros: Al decidir cambiar toda la estructura de la api, para que index.php sea el endpoint principal y que este redirija a los demás, no hemos encontrado varios problemas, el más extraño por decirlo así era un problema al utilizar rutas relativas, el php no era capaz de encontrar las dependencias, y decidimos utilizar rutas absolutas para los require_once.
- Puesta en producción: Necesitamos mejorar el despliegue, hemos acabado copiando desde filezilla, cosa que no es comoda, además al trabajar algo en producción se ha hecho incomodo subirlo desde producción a github.