Creación de Bots de Discord con Discord.js

Manuel de Castro Caballero

GUI
Grupo Universitario de Informática
Escuela de Ingeniería Informática, Universidad de Valladolid

Introducción

- **Discord:** chat de voz y texto enfocado a videojugadores.
- **node.js:** *runtime* de javascript con base similar a Chrome.
  - **JavaScript:** lenguaje interpretado enfocado al desarrollo de páginas web.

Figura: Logo de discord

Figura: Logo de node.js

JavaScript

- Sintaxis relativamente similar a Java...
- ... no tiene nada que ver con Java.
- Scripting $\rightarrow$ interpretado.
- (Se ejecuta en el cliente.)
- ...
- No hace falta saber de antemano JavaScript para hacer un bot de Discord.

Figura: Logo de JavaScript

Node.js

- Se utiliza para ejecutar JavaScript sin un navegador.
- Instalación:
  `https://nodejs.org/`

Discord.js

- Módulo de **node.js** para interactuar con la API de Discord.
- Básicamente la opción más popular para hacer bots de Discord.
- Ventajas sobre otras librerías/modulos. [...]
- Documentación: `https://discord.js.org/#/docs/main/stable/general/welcome`

Figura: Logo de Discord.js

```
const Discord = require('discord.js');
const client = new Discord.Client();

client.on('ready', () => {
  console.log(`Logged in as ${client.user.tag}!`);
});

client.on('message', msg => {
  if (msg.content === 'ping') {
    msg.reply('Pong!');
  }
});

client.login('token');
```

Figura: Ejemplo de uso de Discord.js

Registrar un bot en discord

- (Iniciar sesión en Discord desde el navegador.)
- Crear una aplicación de Discord:
  `https://discordapp.com/developers/applications/`
- Bot → Añadir Bot
- Profit.
- **Token**: Identificador del bot **en todo Discord**.
  IMPORTANTE mantenerlo secreto.

- Añadir bot a un servidor (2 opciones):
  - OAuth2 → Scopes (seleccionar "Bot") → Acceder a la URL generada.
  - `https://discordapi.com/permissions.html`
    Pegar la CLIENT ID de nuestra aplicación → Acceder a la URL generada.

Creando el entorno javascript

- (Nueva carpeta para nuestro bot.)
- Iniciar terminal en el directorio del bot.
- Entorno general:
  `npm init`
  - Rellenar, o dejar los valores por defecto: nombre, versión, descripción, script de inicio, comando de prueba, repositorio git, palabras clave, autor, licencia.
  - Confirmar (yes).
- Dependencias de discord.js:
  `npm install discord.js`
- (Opcional) Cargador de variables de entorno:
  `npm install dotenv`
  (o cualquier otro módulo equivalente)

.env

Fichero **privado** con las *variables de entorno*. No debería compartirse. (Añadir en .gitignore.)

```
TOKEN=[token del bot de Discord; el que hemos copiado antes.]
PRE=[prefijo para los comandos. Suele ser "!".]
ROL=[rol mínimo para interactuar con el bot.]
[...]
```

index.js

- Importar discord.js en una constante:

```
const Discord = require("discord.js");
```

- Inicializar las variables de entorno:

```
require("dotenv").config();
```

- Creación y registro del bot (como objeto):

```
const bot = new Discord.Client();
bot.login(process.env.TOKEN);
```

Orientación a eventos: Mensaje de bienvenida

Acción al iniciarse el bot, utilizando una función lambda:

```
bot.on("ready", () => {
    console.log("Hola Discord!");
});
```

Validación de comandos

Detectamos un mensaje; comprobamos que cumple las características de "comando":

```
bot.on("message", msg => {
    // IMPORTANTE: no interactuar con bots.
    if (msg.author.bot)
        return;

    // Comprobar prefijo:
    if (!msg.content.startsWith(process.env.PREFIX))
        return;

    // Comprobar rol del autor:
    if (!msg.member.roles.find(r =>
        r.name === process.env.ROL))
        return;

    [...]
});
```

Analizador de comandos

Separamos el mensaje en comando y argumentos:

```
bot.on("message", msg => {
    [...]

    // Argumentos:
    let args = msg.content  // Contenido (texto) del mensaje.
        .substring(process.env.PREFIX.length) // Sin prefijo.
        .split(" ");        // Separado por espacios.

    // Comando:
    let cmd = args.shift()  // Primera palabra del mensaje.
                            // (Se elimina de args.)
        .toLowerCase();     // Comando en minúsculas.

    [...]
});
```

Procesador de comandos

Ejecutamos el script (fichero) correspondiente al comando:

```
bot.on("message", msg => {
    [...]

    try {

        // Actualizar el script del comando (por si acaso):
        delete require.cache[require
            .resolve(`./commands/${cmd}.js`)];

        // Cargar el script correspondiente:
        let cmdScript = require(`./commands/${cmd}.js`);
        cmdScript.run(bot, msg, args);

    } catch (e) { console.log(e.stack); }
});
```

Argumentos a comandos

- **Objeto bot**, ya que es quien tiene que enviar mensajes, responder, etc...
- **Mensaje como objeto msg**, para acceder a sus campos como *autor*, *canal*, *texto*...
- **Argumentos** del usuario para el comando, que funcionan como opciones.

Encuestas con reacciones

- ¿Cómo tomáis decisiones grupales en Discord?
- Las encuestas con reacciones tienen numerosos inconvenientes:
  - Las opciones se tienen que describir en el mensaje.
  - Se puede "votar" a más de una opción a la vez.
  - Todas las opciones empiezan con 1 voto.
  - Se puede seguir reaccionando al mensaje con otras reacciones.
  - ...

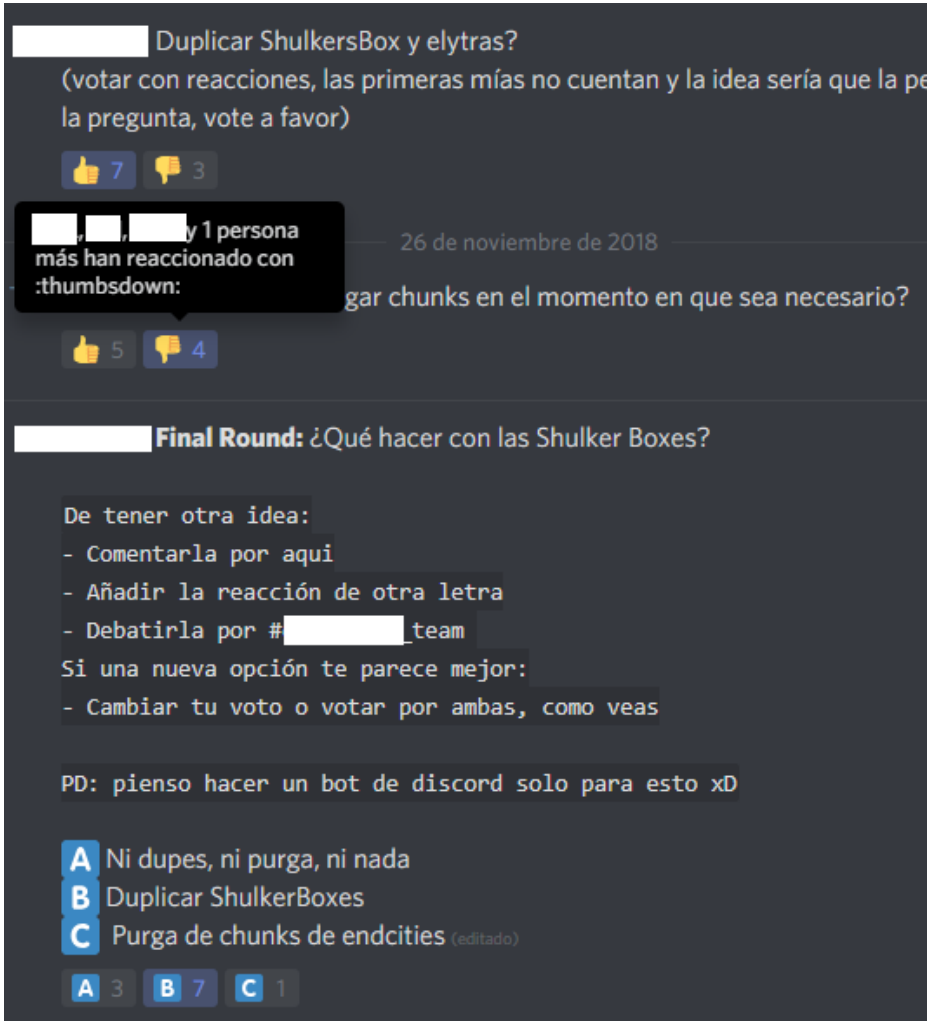

Figura: Encuestas utilizando reacciones

Encuestas por bots

- Crear encuestas mediante comandos:
  `!poll [Nombre] [Opción1] [Opción2] [...]`
- Comandos para otras opciones: votar (`!poll vote`), cerrar (`!poll close`), ver anteriores (`!poll view`), añadir un temporizador (`!poll timer`), borrar encuestas del sistema (`!poll purge`).
  - Para simplificar, solo permitiremos una encuesta a la vez en el sistema (la última creada). Implementaremos `!poll`, `!poll vote` y `!poll view`.
- Se utilizará "_" para indicar separación de palabras en los argumentos:
  `!poll Encuesta_de_prueba Opción_1 Opción_2 Opción_14`
  (El bot las reemplazará por espacios.)
- El resultado será un mensaje RichEmbed.

Módulo encuesta

- Representamos las encuestas como objetos (JSON).
  - Las guardaremos como ficheros para poder recuperarlas.
- Los votos serán listas de usuarios para cada opción (número de votos = longitud de la lista).
  - Para cada voto, primero se borra el usuario de todas las listas en las que esté y luego se añade a la solicitada.
  - Se mostrará el porcentaje de votos para cada opción.
- Métodos:
  - `constructor(nombre, opciones, servidor, canal)`
  - `vote(opcion, usuario)`
  - `getTotalVotes()`
  - `save()`
  - `static fromJSON(json)`
- El módulo se exporta con:

```
[Requires...]
class Encuesta { [Cuerpo del módulo...] }
module.exports = Encuesta;
```

`constructor()`

Crea el objeto encuesta. Simplemente inicializa los atributos.

- `nombre`: nombre de la encuesta (se muestra en la cabecera).
- `opciones`: lista con todas las opciones de la encuesta. Debe tener **2 o más elementos**.
- `servidor`: identificador del servidor (guild) en el que se creó la encuesta, utilizado para guardar las encuestas en ficheros. Necesario si se ejecuta el mismo bot en varios servidores distintos.
- `canal`: identificador del canal en el que se creó la encuesta, utilizado para guardar las encuestas en ficheros. Necesario si se quiere tener encuestas distintas para cada canal (y mantener privadas las encuestas de canales privados).

`vote()`

Añade un voto a una opción. Debe eliminar el anterior voto del usuario, de haberlo (eliminar al usuario de las listas en las que aparezca).

- `opcion`: índice de la opción a la que añadir el voto. Debe estar **en el rango de opciones**.
- `usuario`: string que represente al usuario en el servidor. Debe ser **invariante y única** para cada usuario (ejemplo: nombre de usuario con discriminador: Pepito#0420).

getTotalVotes()

Devuelve el número total de votos en la encuesta (contando todas las opciones). Utilizado para calcular el porcentaje de votos en una opción.

save()

Guarda la encuesta en un fichero JSON para poder recuperarla luego.

```
let fs = require("fs");

[...]

save() {
    // Guardamos la encuesta en una carpeta
    // con el servidor por nombre:
    let dir = `./encuestas/${this.servidor}`;
    // Creamos la carpeta si no exite:
    if (!fs.existsSync(dir)) fs.mkdirSync(dir);

    // La encuesta se guarda con el nombre del canal:
    fs.writeFile(`${dir}/${this.canal}.json`,
        JSON.stringify(this), // String JSON del objeto.
        err => if (err) throw err);
}
```

static fromJSON()

Carga una encuesta a partir de un fichero JSON.
Un fichero JSON no guarda los métodos de un objeto, solo sus atributos. Hay que crear una nueva encuesta (lo que crea sus métodos), y asignarle los atributos del JSON.

- `json`: objeto JSON creado a partir del fichero de la encuesta.

```
static fromJSON(json) {
    // Creamos un objeto encuesta "por defecto" temporal:
    let enc = new Encuesta("", [0, 0], ".temp", ".temp");

    // Le asignamos el objeto JSON:
    // (Es decir, le pasamos los atributos del objeto JSON.)
    Object.assign(enc, json);
    return enc;
}
```

poll.js

Script que se ejecuta cuando el bot recibe el comando `!poll`.

- Debe importar el módulo Encuesta y `"fs"` (para abrir la encuesta del canal).
- Se utilizará un switch para ejecutar la opción correcta del comando.
- Implementará un método asíncrono `async enviar(encuesta, msg)` que enviará el RichEmbed de la encuesta.

RichEmbed

Mensajes especiales, con un borde coloreado y cosas fancy dentro.

- `setColor(color)`: cambia el color del borde.
- `setTitle(titulo)`: cambia el título del embed.
- `addField(nombre, descripción)`: añade un campo (y su descripción) al embed; hasta un máximo de 25 campos.
- ... (ver documentación).

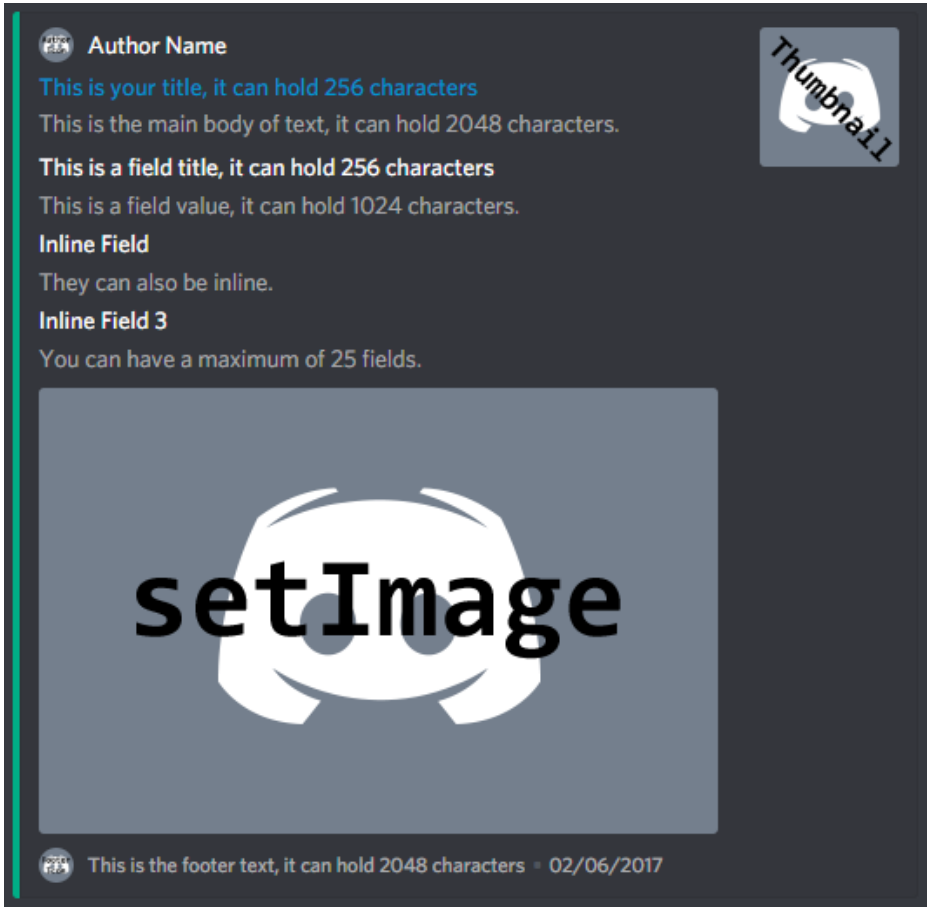

Figura: Mensaje embed de Discord

Guías para programar el bot

- Tener siempre la documentación cerca. Es muy útil.
- Con un `mensaje` puedes acceder a su `usuario`, su `canal` y su `guild`.
- Con un `canal` puedes acceder a su `guild` y a todos sus `mensajes` (como lista).
- Con un `guild` puedes acceder a todos sus `canales` (como lista) y a todos sus `miembros` (como lista).
- Con un `usuario` puedes acceder a su `canal de mensajes directos`.
- Con un objeto puedes acceder a todas sus propiedades básicas y de esperar (`nombre` e `id` para `usuarios`, `canales` y `guilds`, ...).
- ...

Agradecimientos

- Al **Grupo Universitario de Informática**, especialmente a **@HylianPablo**, por su inestimable ayuda y por proporcionar el material necesario para realizar el taller.
  - Seguidnos en Redes Sociales:

    Twitter: @GUI_UVa
    Instagram: @gui_uva
- A **RetroDevelopment**, porque ~~le he copiado mucho codigo~~ su código me sirvió como guía a la hora de aprender a hacer bots de Discord y plantear este taller.
- A **todos los asistentes**, por su entusiasmo por aprender más.

Contacto y dudas

- **GitHub**: 0xb01u
- **Telegram**: @bomilk
- **Presencial**: sede del GUI. Si no estoy, preguntad por Bolu.
- Intentaré resolver cualquier duda que tengáis o ayudaros a seguir desarrollando vuestro bot.

¿Qué nos depara el futuro?

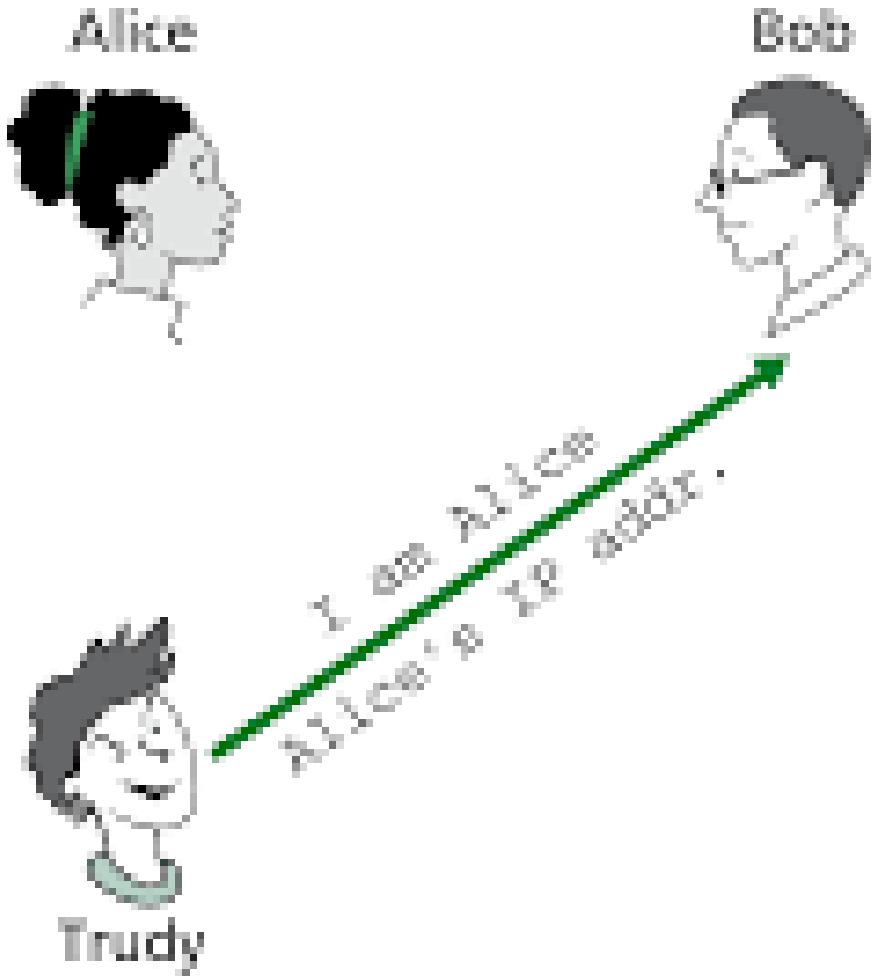

Figura: ¿Cómo podemos hacer que sólo Alice pueda enviar mensajes en su nombre?

Figura: ¿Cuál es este juego?