MANUAL DE PROGRAMADOR CHATBOT

1. Introducción:

El presente manual está diseñado para guiar a los programadores en el proceso de creación y manejo de flujos de conversación utilizando la librería @bot-whatsapp. Se abordarán funciones esenciales como addKeyword, addAnswer, y addAction, así como la manera de implementar eventos predefinidos.

2. addKeyword: La función addKeyword es importada desde @bot-whatsapp y se utiliza para iniciar un flujo de conversación basado en una configuración predefinida. Para crear un flujo, es imperativo declarar addKeyword así:

```
const { createBot, createProvider, createFlow, addKeyword, EVENTS } = require('@bot-whatsapp/bot')
```

Al momento de crear un flujo se debe declarar addKeyword de la siguiente manera

const flujoEjemplo = addKeyword('palabra clave de flujo')

La palabra clave activará el bot, redirigiéndolo a dicho flujo. addKeyword se convierte en el punto de inicio para cualquier diálogo, detectando palabras o frases clave y desencadenando un flujo predefinido de conversación.

3. EVENTS: addKeyword puede trabajar en conjunto con eventos predefinidos disponibles en @bot-whatsapp. Los eventos son acciones específicas o interacciones del usuario que desencadenan respuestas o flujos específicos.

- **EVENTS.WELCOME:** Se usa para que el Flow inicial empiece con cualquier palabra o carácter que se digite por parte del usuario.

- **EVENTS.DOCUMENT:** Se usa para detectar cuando el usuario envíe documentos en su mensaje. Esta función está disponible para usarla según el requerimiento del bot.

- **EVENTS.LOCATION:** Se usa para detectar cuando el usuario envíe localización en su mensaje. Esta función está disponible para usarla según el requerimiento del bot.

- **EVENTS.MEDIA:** Se usa para detectar cuando el usuario envíe imágenes o videos en su mensaje. Esta función está disponible para usarla según el requerimiento del bot.

- **EVENTS.VOICE_NOTE:** Se usa para detectar cuando el usuario envíe notas de voz en su mensaje. Esta función está disponible para usarla según el requerimiento del bot.

- **EVENTS.ACTION:** Este evento es diferente a los demás ya que se utiliza para moverse entre flujos sin palabras clave. Se puede mover mediante gotoFlow, esta función se explica mas adelante en el documento

4. AddAnswer: Esta función es vital para responder al usuario según el flujo conversacional en curso. Cada flujo debe tener al menos un addAnswer o un addAction. Se configura de la siguiente manera:

const flujoEjemplo = addKeyword('palabra clave de flujo')

.addAnswer('mensaje que se mostrara al usuario')

Opciones Adicionales:

- **Array:** Dentro del mensaje se pueden poner varias líneas de texto que se quieren mostrar el usuario, tal como se muestra en las siguientes imágenes.

```
flowPrincipal = addKeyword('hola')
    .addAnswer(['mensaje1', 'mensaje2'])
```

- **Delay:** Esta opción retrasa el mensaje la cantidad de tiempo que se configure. Este tiempo deberá darse en milisegundos.

```
flowPrincipal = addKeyword('hola')
    .addAnswer('mensaje retrasado por tiempo', { delay: 2000 })
```

- **Capture:** Esta función viene predefinida en false porque lo que es necesario declararla en true para su uso. Se encarga de capturar el mensaje que quiere digitar el usuario, es decir si queremos preguntarle algo al usuario va a esperar hasta recibir una respuesta a la pregunta.

```
flowPrincipal = addKeyword('hola')
    .addAnswer('Cual es su nombre', { capture: true })
    // No continua hasta recibir la respuesta del usuario

    //Pasa a este addAnswer despues de responder la primera pregunta
    .addAnswer('cual es su edad',{capture:true})
```

- **Idle:** Debe ir acompañado obligatoriamente de un capture, ya que este idle se encarga de finalizar el tiempo de espera de respuesta del usuario. Se debe definir en milisegundos. También se usará gotoFlow el cual se lo explica mas adelante en este documento. Para el correcto funcionamiento se debe hacer una desestructuración de la función para poner el condicional.

```
flowPrincipal = addKeyword('hola')
    .addAnswer('Tiempo de capture', { idle:15000 , capture:true},async({gotoFlow})=>{
        if (ctx?.idleFallBack) {
            return gotoFlow(flujoFinal)
        }
        //Si se recibe respuesta antes del tiempo estipulado se continua con la logica del flujo
    })
```

- **Media:** Permite enviar PDF, imágenes y videos

```
flowPrincipal = addKeyword('hola')
    .addAnswer('Envio multimedia', { media: /ruta/imagen/video/pdf })
    //puede ser ruta local o url
```

Todas las funciones mencionadas en el addAnswer pueden ir juntas dependiendo la necesidad de cada flujo.

5. addAction: Implementa exactamente las mismas funciones con la diferencia que no es necesario para un mensaje para mostrar al usuario, ya que se puede usar únicamente para realizar validaciones.

```
flowPrincipal = addKeyword('hola')
    .addAction({ media:ruta/media }
    //Envia imagen sin texto.
    )
```

Se pueden usar todas las funciones antes mencionadas en addAnswer

6. Desestructuración: Tanto addAnswer como addAction pueden incorporarse en una función asíncrona, permitiendo una interacción más dinámica y adaptable. Las opciones disponibles son ctx, endFlow, gotoFlow, flowDynamic, fallback y provider.

```
flowPrincipal = addKeyword('hola')
    .addAnswer('Mensaje',async(ctx,{gotoFlow,endFlow,flowDynamic,fallBack,provider })=>{
        /**
        * Logica del flujo
        */
    })
```

En la imagen anterior se puede observar la manera de codificar las diferentes funciones mencionadas y se aclara que no es necesario tener todas las funciones dentro de la desestructuración, solo se debe declarar las que sean necesarias para el manejo del flujo que se está trabajando.

A continuación, se describirá la utilidad de cada una:

Ctx: Es la abreviación de contexto y tiene varios usos entre los que encontramos ctx.body, ctx.from, ctx.pushName.

```
flowPrincipal = addKeyword('hola')
    .addAnswer('Mensaje',{capture:true},async(ctx,{gotoFlow,endFlow,flowDynamic,fallBack,provider })=>{
        mensajeRecibido = ctx.body
        numeroDeUsuario = ctx.from
        nombreDeWhatsapp = ctx.pushName
    })
```

gotoFlow: Se encarga de llevar de un Flow a otro, esta función se usa para hacer validaciones o ejecutar procesos dentro del flujo y que una vez terminado o cumplida la validación vaya de flow1 a flow2. Puede ir dentro de condicionales y se puede usar un gotoFlow dentro de cada condicional.

```
flowPrincipal = addKeyword('hola')
    .addAnswer(['1. Ir a flujo1','2. Ir a flujo2'],{capture:true},async(ctx,{gotoFlow})=>{
        if(ctx.body === '1'){
            gotoFlow(flow1)
        }
        if(ctx.body === '2'){
            gotoFlow(flow2)
        }
    })
```

En el ejemplo anterior se muestra la manera como se debe aplicar gotoFlow y su uso va a depender de las validaciones que se requiera hacer, como consultar API, bases de datos entre otros.

fallBack: Se encarga de enviar el último mensaje del addAnswer, siempre y cuando no se cumpla ninguna condición dentro del Flow, tal y como se muestra en el siguiente ejemplo.

```
flowPrincipal = addKeyword('hola')
    .addAnswer(['1. Ir a flujo1','2. Ir a flujo2'],{capture:true},async(ctx,{gotoFlow,fallBack})=>{
        if(ctx.body === '1'){
            gotoFlow(flow1)
        }
        if(ctx.body === '2'){
            gotoFlow(flow2)
        }
        else fallBack()
    })
```

También se puede enviar un mensaje personalizado para hacerle entender al usuario que esta sucediendo y así garantizar usabilidad en el bot.

```
flowPrincipal = addKeyword('hola')
    .addAnswer(['1. Ir a flujo1','2. Ir a flujo2'],{capture:true},async(ctx,{gotoFlow,fallBack})=>{
        if(ctx.body === '1'){
            gotoFlow(flow1)
        }
        if(ctx.body === '2'){
            gotoFlow(flow2)
        }
        else {fallBack('Mensaje personalizado de fallBack')}
    })
```

endFlow: Este método finaliza los flujos del bot, se lo puede hacer dentro de un condicional después de hacer validaciones o también por fallBack tal y como se muestra en el siguiente ejemplo.

```
contadorFallBack
flowPrincipal = addKeyword('hola')
    .addAnswer(['1. Ir a flujo1', '2. Ir a flujo2'], { capture: true }, async (ctx, { gotoFlow, fallBack }) => {
        if (ctx.body === '1') {
            gotoFlow(flow1)
        }
        if (ctx.body === '2') {
            gotoFlow(flow2)
        }
        else {
            contadorFallBack++;
            if (contadorFallBack === 3) {
                contadorFallBack = 0
                return endFlow({ body: '❌ *Opcion no valida* ❌ \n\n*ChatBot* finalizado, nos vemos luego. 👋😊' });
            } else {
                fallBack(` ⚠️ *Selecciona una opcion valida* ⚠️ `);
            }
        }
    }
    )
```

flowDynamic: Esta función se utiliza para devolver mensajes dinámicos que pueden venir de una API o Base de datos, también sirve para enviar mensajes dentro de los addAnswer para un claro entendimiento de la función, se implementaran flowDynamic en el ejemplo que se viene trabajando.

```
contadorFallBack
flowPrincipal = addKeyword('hola')
    .addAnswer(['1. Ir a flujo1', '2. Ir a flujo2'], { capture: true }, async (ctx, { gotoFlow, fallBack, flowDynamic }) => {
        if (ctx.body === '1') {
            await flowDynamic('Te vamos a llevar a flujo1')
            gotoFlow(flow1)
        }
        if (ctx.body === '2') {
            await flowDynamic('Te vamos a llevar a flujo2')
            gotoFlow(flow2)
        }
        else {
            contadorFallBack++;
            if (contadorFallBack === 3) {
                contadorFallBack = 0
                return endFlow({ body: '❌ *Opcion no valida* ❌ \n\n*ChatBot* finalizado, nos vemos luego. 👋😇' });
            } else {
                fallBack(` ⚠️ *Selecciona una opcion valida* ⚠️ `);
            }
        }
    }
    )
```

8. Conclusión:

El dominio de estas funciones y técnicas es esencial para desarrollar chatbots interactivos y eficientes en @bot-whatsapp. Con una comprensión completa de las capacidades y limitaciones, los programadores estarán equipados para diseñar flujos de conversación personalizados que satisfagan las necesidades específicas de los usuarios finales y los objetivos empresariales.

Para una guía más detallada y visual, se incluyen imágenes y diagramas de flujo en las siguientes secciones del manual para ilustrar la implementación efectiva de estas funciones en un entorno real.