Manual de programador

29-marzo-2023

Información del Documento:

Proyecto:	Manual de programador YALO
Elaborado por:	Daniel Holguín Domínguez / Backend Intern Developer Santiago Marín Ocampo / Backend Intern Developer

Contenido

Tabla de figuras

Flows

Un flujo contiene los componentes principales (Flow Builder, Engagement, Commerce) para la creación de soluciones conversacionales.

Como crear un flujo

Para la creación de un flujo se debe acceder al apartado que se encuentra señalado a continuación:

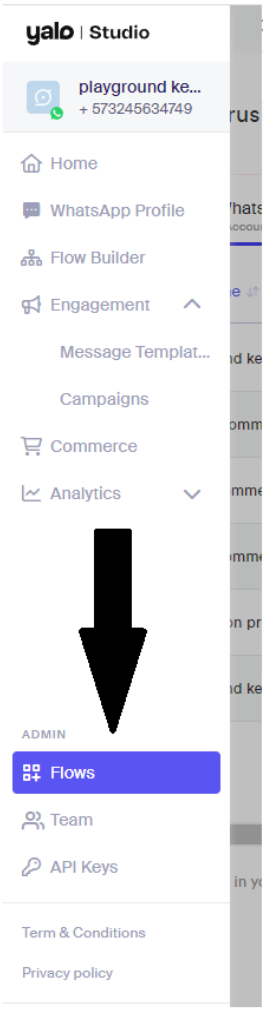


Figura 1: Como crear un flujo

Una vez seleccionado el apartado, encontrará una interfaz en donde se listan los flujos ya creados y la posibilidad de crear uno nuevo. Para crear un nuevo flujo debe seleccionar la opción «*NEW FLOW*» mostrada en las figuras a continuación:

Figura 2: Opción para crear un nuevo flujo

Esta opción desplegará un menú modal para donde se configurará el nombre del flujo y el idioma de este.

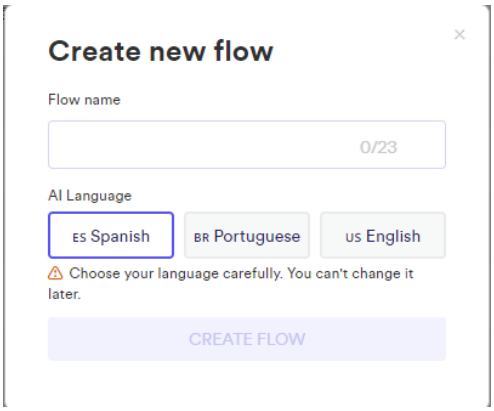


Figura 3: Menú modal para crear un nuevo flujo

Adicionalmente, como se ve en la *figura 2* para los flujos ya creados existen las funciones de configurar, duplicar, descargar y eliminar; con la opción descargar se refiere a exportar un flujo para subirlo por ejemplo a otra cuenta mediante la opción de importación que ofrece Yalo, en la fecha hacia abajo al lado del botón de «*NEW FLOW*»

Como asignar número al flujo

Para asignar un numero al flujo se debe activar mediante Facebook Bussiness Manager o directamente usando la plataforma de Yalo Studio, ambas opciones tienen un tiempo de activación necesario, para Facebook Bussiness Manager el tiempo es de 30 minutos, mientras que mediante Yalo Studio tarda alrededor de 2 días. Cabe destacar que para ambas opciones es necesario una cuenta en Facebook Bussiness Manager.

Perfil de WhatsApp (WhatsApp Profile)

Este apartado se habilita luego de asignar correctamente el número al flujo, permite modificar datos principales del perfil WhatsApp como la foto de perfil, la dirección, la descripción del negocio, categoría del negocio, sitio web y email de contacto, la interfaz se ve de la siguiente manera:

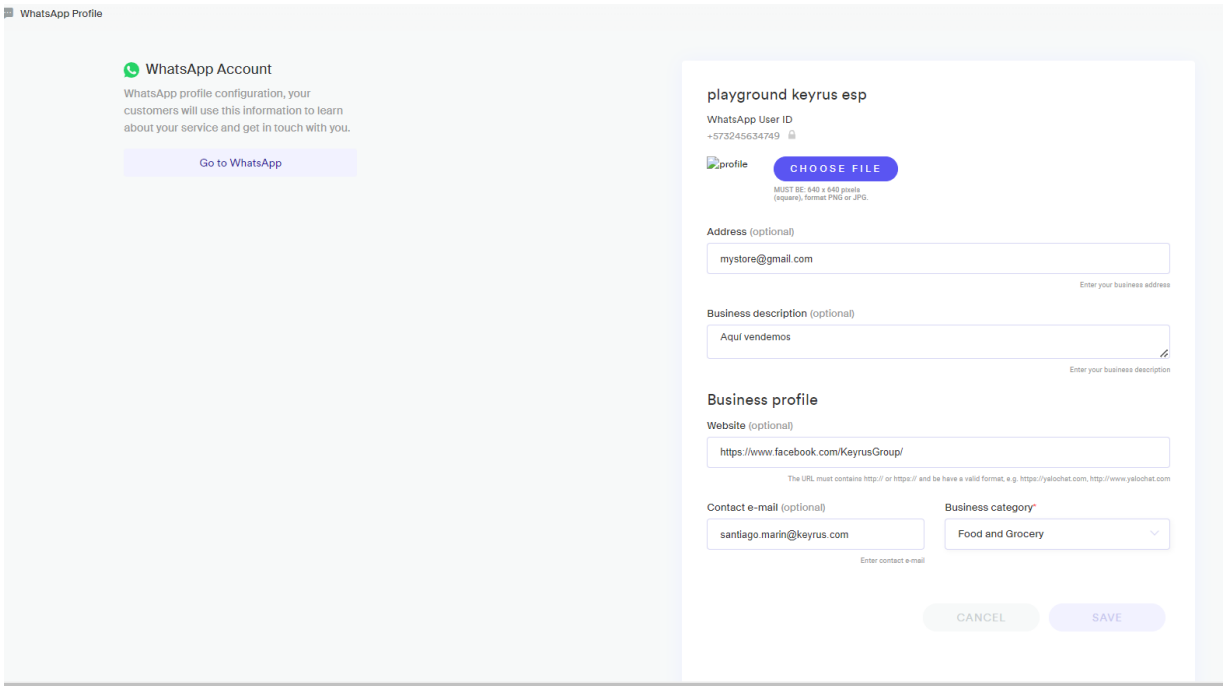


Figura 4: Interfaz personalización perfil WhatsApp

Como conectarse a un flujo

Para conectarse a un flujo y probarlo por medio de WhatsApp se hace de la siguiente manera:
Si el flujo no tiene número asignado podrá usar el de Yalo Studio https://wa.me/+5215548374302, a ese chat se escribe el siguiente comando *!test workflow-name*, la palabra !test seguido del nombre del flujo. Si por el contrario el flujo tiene número asignado tendrá que escribir ese comando en el número que fue asignado.

Flow Builder

Flow Builder es la herramienta principal de Yalo Studio para crear flujos conversacionales y se divide en los siguientes apartados:

Canvas

Existen 4 tipos de elementos elegibles en el canvas:

Main starter: Es el inicio del flujo iniciado por el usuario. Este elemento es creado automáticamente cuando se crea un flujo. Solo existen un main starter por flujo.

Webhook: También es el inicio de un flujo, pero a diferencia del Main starter, es el inicio de una conversación proactiva, es decir es iniciado por Yalo y no por un usuario; mediante una petición REST tipo POST. Pueden existir más de un webhook por flujo.

Campaign start: Se comporta igual que un webhook, pero la diferencia con este es la forma de activarse; se activa mediante una campaña. Cuando se lanza una campaña, luego de ser recibida por el usuario, empieza en el flujo donde se encuentra el

campaign start. Pueden existir más de un campaign start por flujo.

Step: Son lo que le dan la estructura y el comportamiento al flujo; mensajes, preguntas, condicionales, etc.…

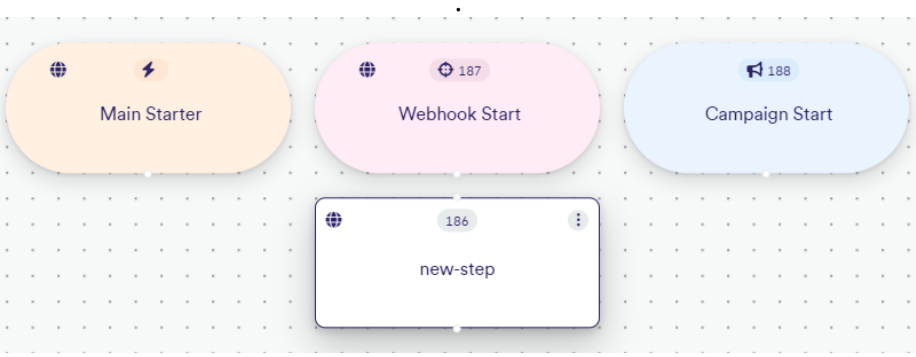


Figura 5: Tipos de steps (pasos)

Variables

Context: El contexto es donde Yalo Studio almacena las variables en formato JSON. Estas variables almacenadas aquí, son temporales, es decir, se borran después de determinado tiempo según la cantidad de tiempo asignada en la sesión del flujo. Por defecto viene asignada a 24 horas.

Profile: El perfil también sirve para almacenar variables, la diferencia con el contexto es que estas variables son persistentes, es decir, no se borran después de expirar la sesión

Yalo por defecto trae las variables vacías: *userMessage*, *userMessageType*, *currentStep*, *userId*, *userName*, *webhookPayload*, *campaignPayload*. Sin embargo, con la librería Context y Profile que se explica más adelante (sección librerías yalo) se puede crear, modificar o eliminar variables. O también crear y modificar variables mediante las actions (sección acciones/variables).

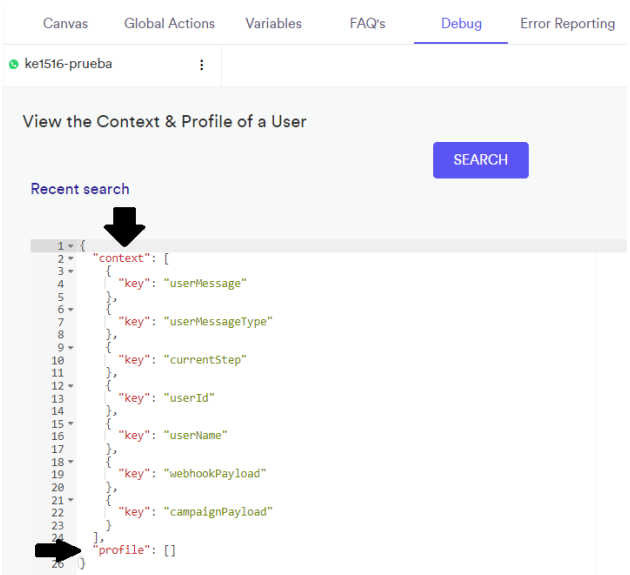


Figura 6: Context & profile variables

Un ejemplo de uso de profile, es para guardar la respuesta (positiva o negativa) de los términos y condiciones y habeas data

en el primer uso del *chatbot*. En la siguiente figura se ve esta variable creada con un valor true (aceptó los términos y condiciones, por lo tanto, no se le vuelve a preguntar).

Figura 7: Ejemplo de uso variable de Profile

Ejemplos y configuración elementos de canvas.

1. Main starter

Cuando un usuario escribe al chat de Yalo, empieza el flujo por el *main starter* y este a su vez llama al step *greeting*, para responder con un saludo preconfigurado.

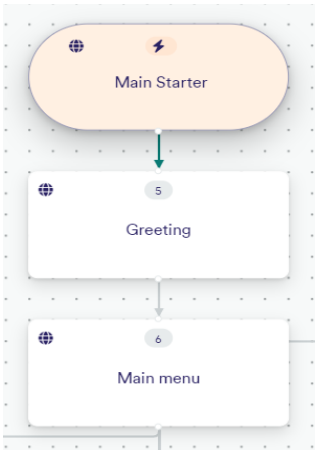


Figura 8: Ejemplo main starter

Paso 1: Seleccionar el main starter, a lado izquierdo de la pantalla aparece las configuraciones del elemento, elegir la actions de *go to*, se desplegará una lista para elegir el paso el cual se desea que continue el flujo; para este caso elegimos el paso llamado *greeting*.

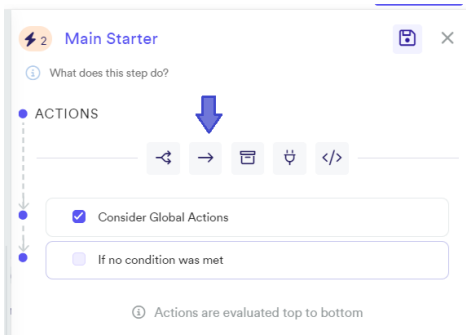


Figura 9: Configuración main starter

2. Webhook

Cuando se desea iniciar una conversación proactiva utilizamos el webhook; por ejemplo, si se necesita enviarle un mensaje con un recordatorio a un usuario:

Paso 1: Conectar el webhook start al paso *Reminder* con la action *go to*.

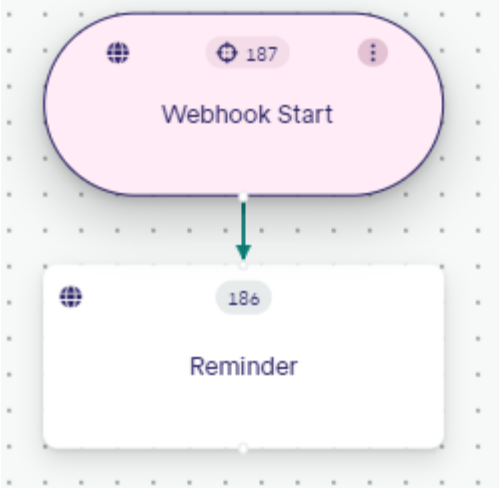


Figura 10: Ejemplo de webhook

Paso 2: Buscar y guardar la URL del webhook; con este URL mediante una petición REST tipo POST, activaremos el flujo.

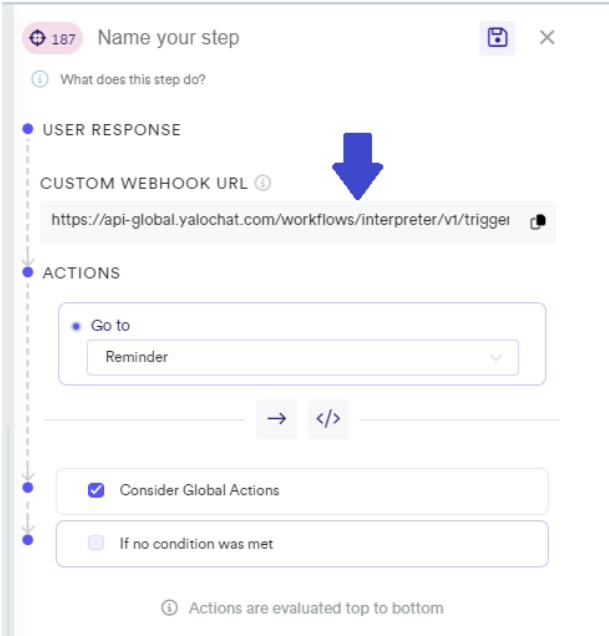


Figura 11: Configuración webhook step

Paso 3: Preparar la petición POST; la petición requiere un formato de body especifico:

```
{
    "userId": "Código de área + número de teléfono del usuario"
}
```

También se pueden enviar datos mediante la petición, para ser utilizados por el flujo:

```
{
    "userId": "Código de área + número de teléfono del usuario",
```

```
    "key": "valor",
    "key": "valor",
    "key": "valor"
    .
    .
    .
}
```

Ejemplo utilizando postman:

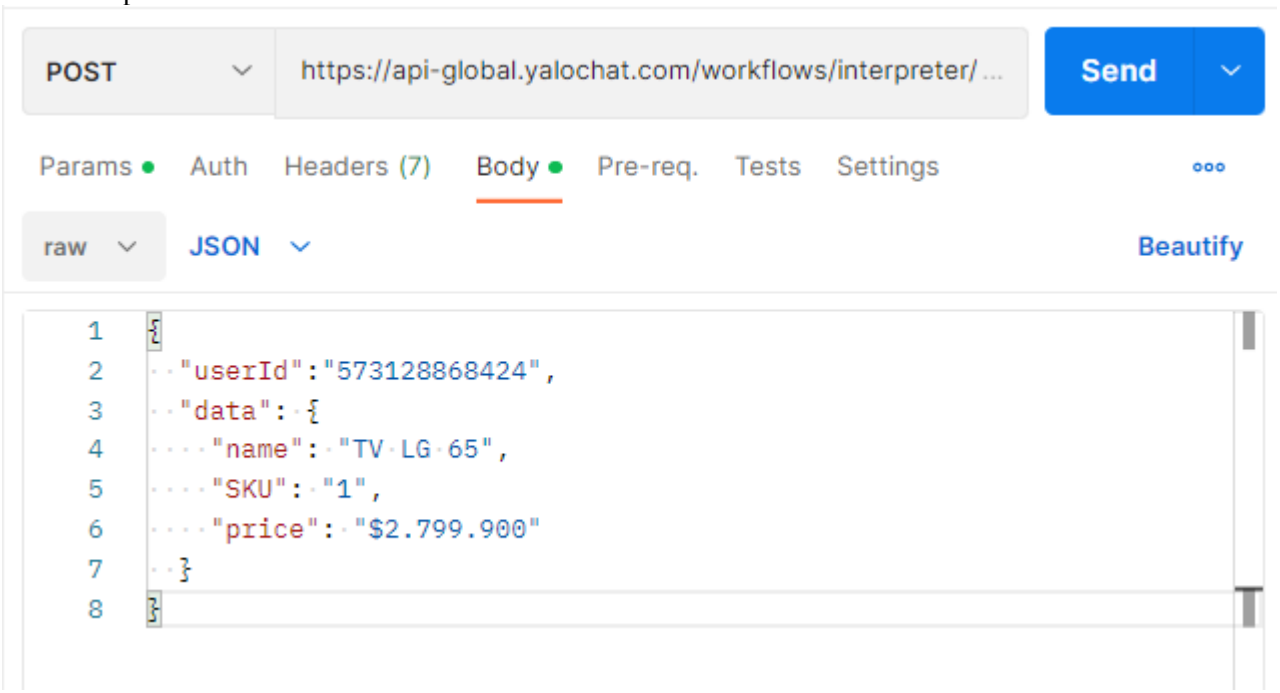


Figura 12: Ejemplo de POST en postman

Los datos enviados por la petición se guardar en una variable de contexto llamada *webhookPayload*.

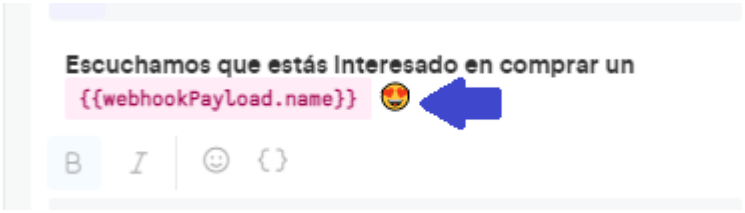


Figura 13: Ejemplo de llamado variable del webhook

3. Campaign start

Su configuración es igual al de un Main starter, con la diferencia de su activación, que será explicada en la sección de engagement.

4. Step

Un step tiene las siguientes funciones:

1. Mensajes

Los mensajes se dividen en 4 tipos:

1. Texto

Mensajes de texto al usuario, Si se tiene variables en el contexto, se puede insertar usando el siguiente formato {{nombre de la variable}}

Ejemplo:

Figura 14: Ejemplo de uso de la variable

Si la variable es un JSON se puede acceder a sus valores de la siguiente forma {{[nombre del JSON].[nombre de la propiedad]}}

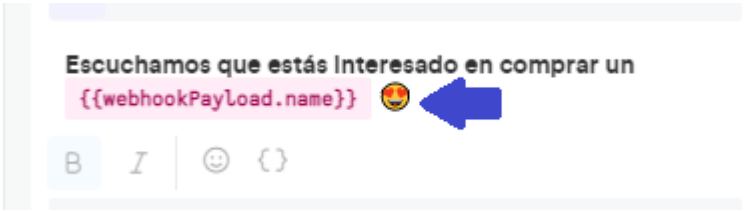


Figura 15: Ejemplo de acceso a valor de variable

HandleBars

Adicionalmente, en la sección de texto se pueden usar HandleBars built-in helpers para mostrar información al usuario de manera interactiva, actualmente están implementados los siguientes HandleBars:

#ifEquals: es un condicional que evalúa si el valor de la variable X es igual al de la variable Y.

#ifNotEquals: es un condicional que evalúa si el valor de la variable X NO es igual al de la variable Y.

#each: sirve para iterar en una lista de elementos.

Para cerrar los bloques se usa la barra invertida: /ifEquals, /ifNotEquals, /each

Ejemplo de aplicación de HandleBars:

Nota: CartItems es un arreglo guardado en el contexto del chat.

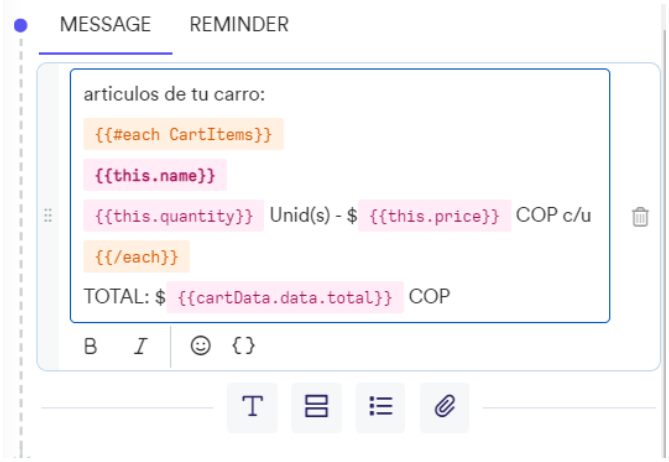


Figura 16: Ejemplo de uso de HandleBars

En el apartado de código de Lua de *storefront* podrán encontrar ejemplos más complejos usando HandleBars

2. Botones

   Se puede configurar un máximo de 3 botones, cuando el usuario le da clic al botón, el mensaje del botón se escribe como respuesta de parte del usuario.

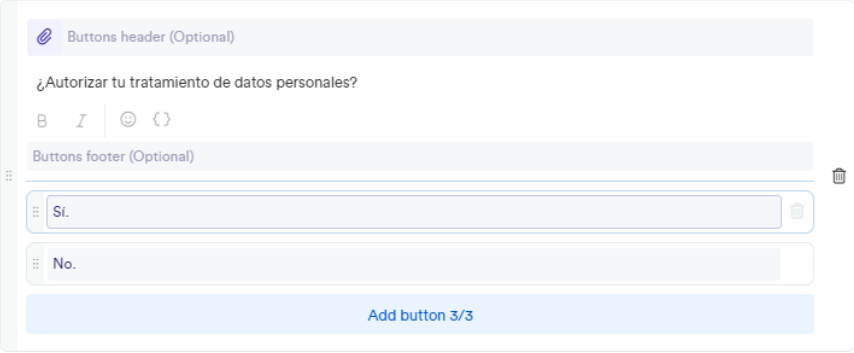


Figura 17: Ejemplo de uso de botones

3. Listas

Su funcionamiento es igual a de los botones, con la diferencia de que las listas se abren en una ventana modal donde se muestras todos los ítems de la lista. Se pueden configurar un máximo de 10 ítems.

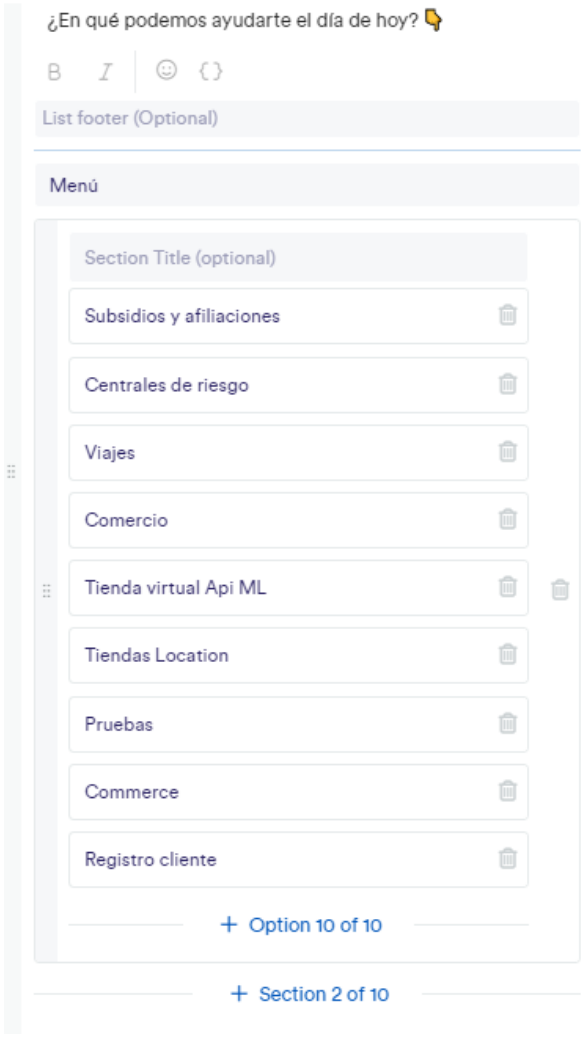


Figura 18: Ejemplo de uso de las listas

4. Multimedia

Se pueden enviar mensajes con imagen, video, audio o documentos mediante URL.

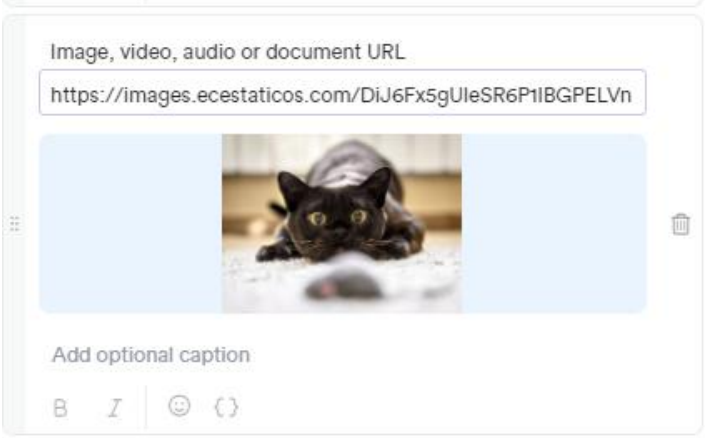


Figura 19: Ejemplo de uso de multimedia

Tipos de media soportado:

Supported Content-Types

Media	Supported Content-Types
audio	audio/aac, audio/mp4, audio/amr, audio/mpeg, audio/ogg; codecs=opus **Note:** The base audio/ogg type is not supported.
document	Any valid MIME-type
image	image/jpeg, image/png
sticker	image/webp
video	video/mp4, video/3gpp **Note:** • Only H.264 video codec and AAC audio codec is supported. • Only videos with a single audio stream are supported.

Figura 20: formatos soportados en cada tipo de mensaje

Nota: Cuando se envía un documento mediante esta opción, se envía sin nombre (aunque el documento tenga nombre), para corregir esto, se hace una implementación mediante código Lua.

Template.media(type: string, url: string, message: string, filename: string)

type: el tipo de recurso; image, video, document, audio.
url: url del recurso.
message: mensaje que será enviado junto al recurso.
filename: nombre con el cual se mandará el recurso.

Ejemplo:

```
filename = "Política de tratamiento de datos personales Nestlé.pdf"
Template.media("document",      "https://www.nestle.com.co/sites/g/files/pydnoa531/files/2019-10/Politica-Tratamiento-de-Datos-Personales-Nestle.pdf", "Léela aquí ☝", filename)
```

1. User response

Una vez que todos los mensajes sean enviados del step, continuara hacia la parte de *user response*, en esta parte se puede elegir si el flujo continúa automáticamente, después de un *delay* programado o si se espera a que el usuario responda, por ejemplo, en el caso de que el mensaje sea una pregunta o una elección.

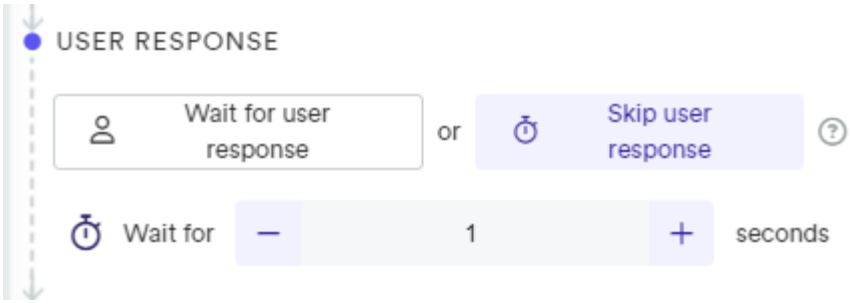


Figura 21: Interfaz de user response

2. *Actions*

Se ejecutan después del *User response* y es donde va la lógica y estructura del paso, en este caso existen cinco posibilidades que son conditions, go to, variables, web API y code los cuales se describen a continuación:

1. Condition: Los condicionales se utiliza para dividir el flujo, según la respuesta del usuario. También se pueden guardar variables o ejecutar peticiones REST (Estas serán explicadas más adelante) según el condicional.

Tipos de condiciones:

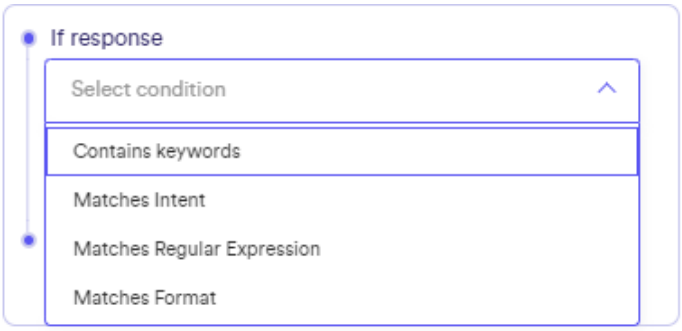


Figura 22: Tipos de condicionales

Contains keywords: Si la respuesta del usuario contiene las palabras claves escritas, entrara al condicional.

Figura 23: Condicional keywords

Nota: las palabras claves no deben llevar tildes; aunque la respuesta del usuario lleve tilde, hará match con la palabra clave sin tilde.

Matches Regular Expression: Si la respuesta del usuario cumple con la expresión regular, entrara al condicional.

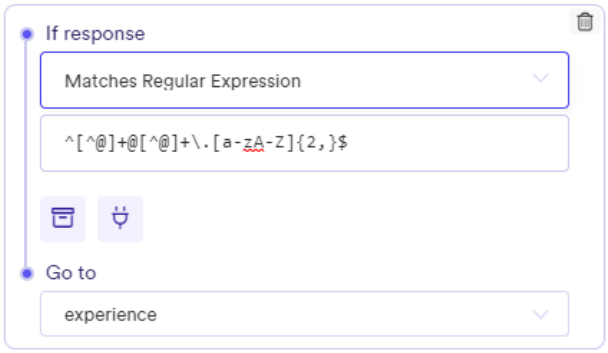


Figura 24: Condicional expresión regular

Matches Format: Si el mensaje del usuario tiene un formato que corresponde con los seleccionados, entrara al condicional.

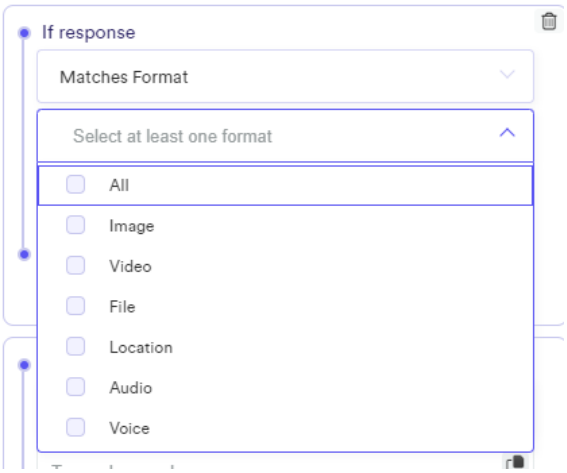


Figura 25: Condicional de formato

Go to: Se utiliza para ir directamente a un siguiente step sin evaluar antes un condicional.

Figura 26: Acción go to

Nota: Los condicionales se evalúa de arriba hacia abajo, el flujo entrara por el primer condicional que cumpla la condición, si ningún condicional cumple la condicional se ira por el próximo *go to* que encuentre.

2. Variable:

Esta acción se utiliza para guardar variables en el contexto.

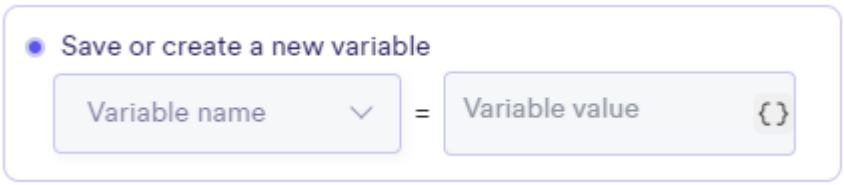


Figura 27: Interfaz creación de variable

Al lado derecho se pone el nombre de la variable donde se quiere guardar el valor, dentro de {{}} y a la izquierda se pone el valor a guardar; el valor a guardar puede ser escrito directamente en este campo o provenir de otra variable del contexto, como por ejemplo el {{userMessage}}.

Ejemplo: Si al usuario se le solicita su nombre se puede guardar de la siguiente forma

Figura 28: Ejemplo de creación de variable

3. Web API:

Se utiliza para realizar peticiones REST.

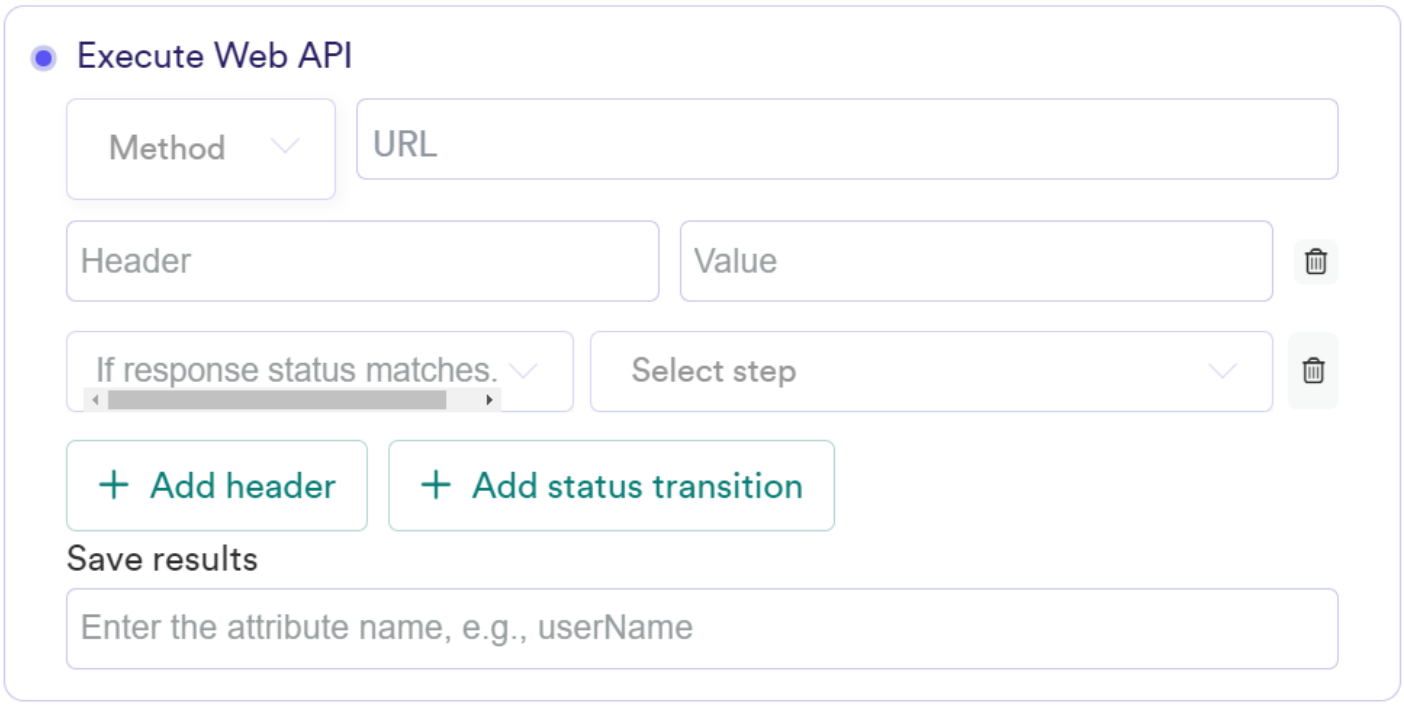


Figura 29: Interfaz Web API

Se puede añadir un *status transition*, los cuales se utilizan para guiar el flujo a diferentes *steps* según el estatus de la petición, por ejemplo, si el status de la petición es un 200 se guía hacia determinado step, pero si el *status* es 404 se guía hacia otro.

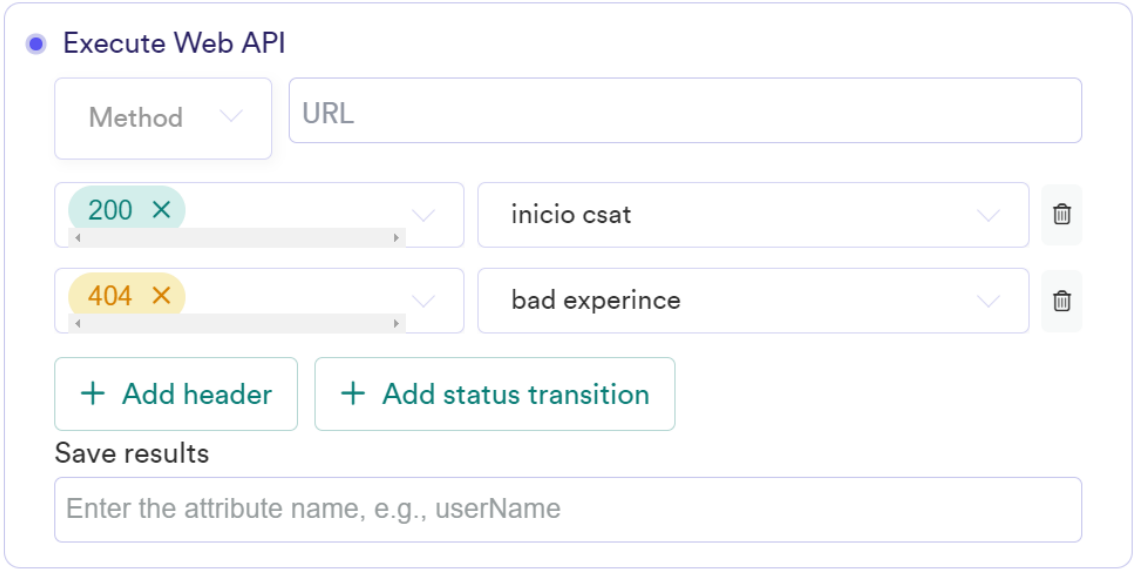


Figura 30: Ejemplo de uso Web API

En el campo de *save results*, se pone el nombre de la variable de contexto donde se guardará el *payload* de la petición.

4. Code:

Yalo trabaja de la mano con el lenguaje de programación Lua, así que para flujos más complejos donde se necesita lógica, que no se puede lograr con las configuraciones básicas, existe la opción de implementar código Lua en los *steps*, que se ejecutan en el orden de las *actions*.

Yalo tienes algunas librerías implementadas:

Librería Context

Se utiliza para acceder y escribir las variables de contexto.

```
Context.get(key: string) -> string
```

key: nombre de la variable en el contexto.
Retorna el valor de la variable.

```
Context.set(key: string, value: string|table) -> nil
```

key: nombre de la variable con el cual será guardada en el contexto.
value: valor de la variable.
No retorna.

```
Context.delete(key: string) -> nil
```

key: nombre de la variable que será eliminada del contexto.
No retorna.

```
Context.check(key: string) -> Boolean
```

Key: nombre de la variable.
Retorna true si la variable existe en el contexto y false en caso contrario.

Librería Profile

La librería Profile tiene los mismos métodos que Context con la diferencia de que es para las variables guardadas en el profile.

```
Profile.set(key: string, value: string) -> nil
Profile.get(key: string) -> string
Profile.delete(key: string) -> nil
Profile.check(key: string) -> boolean
```

Librería Global

Las variables globales solo se pueden acceder o validar.

```
Global.get(key: string) -> string
Global.check(key: string) -> boolean
```

Librería input

Se utiliza para evaluar si cadenas coinciden con otra, si cumples una expresión regular o si se encuentra en una lista de keywords. También proporciona métodos para recuperar el perfil y los detalles del último mensaje del usuario.

```
Input.match(regex: string, expression: string, value?: string, options?: table) -> boolean
```

regex: tipo de coincidencia.

Tipos de coincidencia:
contains: mediante una lista de palabras claves.
regex: mediante una expresión regular.
isEqualTo: mediante comparación directa.

Expression: cadena o expresión regular con la cual se evalúa.
value: cadena a evaluar.
options: opciones adicionales de procesamiento a la cadena a evaluar.

Formato de options;

```
Options ={
    toLower = true,
    removeDiacritics = true,
    reduceRepeatedAlphanumerics = true
}
```

toLower: convierte la cadena a minúsculas
removeDiacritics: Quita las diacríticas de la cadena, como por ejemplos las tildes.
reduceRepeatedAlphanumerics: Si existe una letra o número que se repite consecutivamente 3 veces o más se reduce a solo dos, por ejemplo: AAAA

Retorna true si la cadena cumple con las condiciones y false en caso contrario.
Ejemplos:

```
match = Input.match('contains', 'blue|red', 'blue')
match = Input.match('regex', '^[0-9()-]+$', '(656) 123-1234')
match = Input.match('isEqualTo', 'Studio NG', 'Yalo')
```

```
Input.getMessage() -> ContextMessage
```

Retorna algunos detalles del último mensaje del usuario.

Formato contextMessage:

```
ContextMessage =
{
  message: string,
  type: string,
```

```
  step: string
}
```

message: último mensaje.
type: tipo de formato del mensaje.
Opciones de type:
text, interactive, image, video, file, location, audio
step: nombre del paso donde se escribió el mensaje.

```
Input.getProfile() -> ContextProfile
```

Retorna algunos detalles del profile del usuario

Formato de contextProfile:

```
ContextProfile =
{
  userId: string,
  name: string
}
```

userId: código de área + número de teléfono
name: nombre que el usuario tiene asignado en su perfil.

Librería JSON

Esta librería se utiliza pasar del formato Lua a formato JSON y viceversa.

Ejemplo de formato:

Variable tipo table en Lua

```
{
    message: "hola",
    type: "text",
    step: "Home"
}
```

Formato JSON:

```
{
    "message": "hola",
```

```
    "type": "text",
    "step": "Home"
}
```

```
JSON.encode(object: table) -> string
```

Pasa de formato Lua a formato JSON.

object: Variable de Lua tipo table
Retorna: un string con el formato JSON.

```
JSON.decode(jsonString: string) -> table
```

Pasa de formato Json a formato Lua.

jsonString: string con el formato Json.

Retorna: una variable tipo table.

Nota: Es importante tener en cuenta que, cuando se trae a una variable tipo table del contexto, esta viene en formato Json y se comporta como un string en Lua. Para tratarla como un objeto Lua, se debe hacer el JSON.decode a la variable.

A su vez cuando se quiera guardar una variable tipo objecto desde Lua al contexto, se debe convertir a Json antes, con JSON.encode.

Librería HTTP:

Se utiliza para realizar peticiones Rest, desde el código Lua.

```
HTTP.get(url: string, headers?: table) -> HttpResponse
HTTP.post(url: string, body?: table, headers?: table) -> HttpResponse
HTTP.put(url: string, body?: table, headers?: table -> HttpResponse)
HTTP.patch(url: string, body?: table, headers?: table) -> HttpResponse
HTTP.delete(url: string, headers?: table) -> HttpResponse
```

Formato de respuesta:

```
HttpResponse =
{
  data: table,
  status: number
  statusText: string
  config: table
}
```

data: payload de la petición en formato table.

status: código de estado de respuesta HTTP.
statusText: estado de la petición.
config: parámetros con los cuales se hizo la petición.

Librería Debug.

Se utiliza para enviar mensajes a WhatsApp desde el código Lua. Son muy útiles para depurar el código y buscar solución a los errores en este.

```
Debug.print(text: string) -> nil
```

text: cadena a imprimir en el chat de Whatsapp.

Nota: Yalo no recomienda utilizar esta librería para imprimir mensajes al usuario; es posible que esta librería sea modificada en futuras versiones de Yalo.

Librería Workflow.

Se utiliza para hacer saltos hacia otro step desde el código Lua.

```
Workflow.branch(stepId: number) -> nil
```

stepId: número del paso al cual se quiere saltar.
No retorna.

Nota: Después de saltar a otro step utilizando esta librería, todo el código Lua debajo de esta instrucción no será ejecutado.

Librería Invoke

Se utiliza para llamar funciones AWS lambdas.

```
Invoke.cloudFunction(lambda: string, body: string) -> ServerlessResponse
```

lambda: nombre de la function lambda.
body: payload de la lambda en formato json.

Formato de ServerlessResponse

```
ServerlessResponse =
{
  success: boolean -- True if the lambda was executed successfully
  payload: table -- Response from the lambda
}
```

success: true si la función lambda fue ejecutada exitosamente.

payload: respuesta de la lambda en formato table Lua.

Nota: Solo se pueden AWS lambdas de Yalo actualmente.

Librería FAQs

Se utiliza para el proceso de analizar preguntas del usuario y compararlas con un banco de preguntas previamente guardas en el apartado de FAQs, mediante la IA de reconocimiento de lenguaje natural de Yalo.

```
FAQ.searchQuestion(question: string, sheetId: string, config?: table) -> FAQResponse
```

question: pregunta hecha por el usuario.
sheetId: id de la hoja donde se encuentras las preguntas a comparar.
config: configuraciones adicionales.

Formato config:

```
Config =
{
  outputAttr: string
  numAnswers: number
  minSingleSimilarity: number
  minMultipleSimilarity: number
  lambdaScript: string
}
```

outputAttr: Nombre de la llave en la que se almacena la respuesta.
numAnswers: Número máximo de respuestas devueltas después de la comparación.
minSingleSimilarity: El umbral para la coincidencia única.
minMultipleSimilarity: El umbral para la coincidencia múltiple.
lambdaScript: AWS lambda para procesar las coincidencias de preguntas frecuentes.

Retorna una tabla Lua con el siguiente formato:

```
FAQResponse =
{
  success: boolean
  singleMatch: boolean
  multipleMatch: boolean
  noMatch: boolean
  searchId: string
  documentId: string
  faqs: FAQsQuestions
  matchQuestion: string
  matchAnswer: string
}
```

Success: True si la ejecución fue exitosa.
singleMatch: True si hubo una coincidencia exacta para la pregunta.
multipleMatch: True si coincidieron varias preguntas.
noMatch: Verdadero si no hubo ninguna coincidencia para la pregunta.
searchId: La búsqueda es devuelta por la búsqueda semántica sin servidor.
documentId: La identificación del documento de los resultados de búsqueda.
Faqs: La lista de todas las preguntas coincidentes.
matchQuestion: Si la respuesta es singleMatch, esta contiene la pregunta coincidente. O N/A si no se encontraron respuestas
matchAnswer: Si la respuesta es singleMatch, contiene la respuesta coincidente. O N/A si no se encontraron respuestas.

Formato FAQs

```
FAQsQuestions =
{
  index: number
  indexEmoji: string
  title: string
}
```

Librería Location

Se utiliza para buscar los negocios físicos más cercanos al usuario y brindarle información sobre estos.

```
Location.stores(keyword: string, radius: number, placeType: string, address: string) -> LocationResult
```

keyword: nombre del negocio.
number: es el radio en metros, para buscar alrededor de la ubicación del usuario.
placeType: el tipo de negocio, tal como está asignado en API de Google Places.
Address: dirección de usuario (ubicación).

Formato LocationResult

```
LocationResult =
{
  success: boolean
  places: LocationDetail
}
```

success: true si se encontró al menos una tienda en el rango.
places: lugares encontrados y sus detalles.

Formato LocationDetail.

```
LocationDetail =
{
  address_components: AddressComponent[]
```

```
  adr_address: string
  business_status: string
  formatted_address: string
  formatted_phone_number: string
  geometry: Geometry
  icon: string
  international_phone_number: string
  name: string
  opening_hours: ExtendedBusinesHours[]
  photos: Photo[]
  place_id: string
  plus_code: Plus
  price_level: number
  rating: number
  reference: string
  reviews: Review[]
  types: PlaceType[]
  url: string
  user_ratings_total: number
  utc_offset: number
  vicinity: string
  website: string
}
```

Formato de AddressComponet

```
AddressComponent =
{
  long_name: string
  short_name: string
  types: string
}
```

Formato de ExtendedBusinessHours

```
ExtendedBusinessHours
{
    weekday_text: string[]
    periods {
        close {
            day: number
            time: string
        }
        open {
            day: number
```

```
            time: string
        }
    }
}
```

Librería String

Se utiliza para limpiar un string o aplicarle algún formato.

`String.cleanText(text: string) -> string`

Quita las diacríticas de la cadena, como por ejemplos las tildes.

`String.onlyNumbers(text: string) -> number`

Selecciona solo los números de la cadena.

`String.split(text: string) -> table`

Transforma la cadena en un array de cadenas, donde cada palabra es un elemento del array.

`String.encodeURI(text: string) -> string`

Transforma la cadena a un URI, remplazando los espacios con %.

`String.formatCurrency(value: string, language: string, currency: string ) -> string`

Aplica un formato de divisa a la cadena.

Nota: para volver toda la cadena a minúsculas utilice string.lower(value: string), con la letra s minúscula en el nombre de la librería.

Librería Schedule

Se utiliza para enviar un mensaje o hacer una transición después de un retraso dado sin actividad del usuario.

```
Schedule.message(stepId: number, delayInMinutes: number)
Schedule.branch(stepId: number, delayInMinutes: number)
```

stepId: número del paso al saltar.
delayInMinutes: tiempo de inactividad programa para activar la acción en minutos

La diferencia entre Schedule.message y Schedule.branch es que el primero solo mostrara el mensaje del paso del stepId

pasado como parámetro, pero el flujo no saltara a este paso, en cambio el Schedule.branch una vez pasado el tiempo de inactividad saltara el flujo a este paso.

Nota: Si el usuario tiene eventos programados, pero envía un mensaje antes de que caduquen esos retrasos, se cancelarán. Puede tener tantos eventos programados como necesite; se ponen en cola correctamente según el retraso proporcionado en minutos.

Librería BusinessHours

Se utiliza para saber si una tienda está cerrada o abierta en el momento que el usuario la consulta.

```
BusinessHours.config(options: BusinessHoursOptions) -> StoreStatus
```

options: zona horaria y horarios de la tienda.

Formato de options

```
BusinessHoursOptions = {
  timezone = "America/Denver"
  monday = "09:00-17:00"
  tuesday = "09:00-17:00"
  wednesday = "09:00-17:00"
  thursday = "09:00-17:00"
  friday = "09:00-17:00"
  saturday = "closed"
  sunday = "closed"
}
```

Timezone: zona horaria donde se encuentra la tienda; https://timezonedb.com/time-zones.

Los demás campos son los días de la semana como clave y como valor se pone el rango de horas en que esa abierta esa tienda; el formato es horario militar y solo dos dígitos para la hora y los minutos ejemplo: 09:00-17:00. También se puede poner la palabra closed como valor para indicar que la tienda no está abierta ese día.

Formato del storeStatus

```
StoreStatus = {
  error = nil,
  open = true,
}
```

error: si ocurre algún error con la ejecución, devolverá la información del error en este campo.
open: devolverá un true si la tienda si se encuentra abierta en el horario y día en el que el cliente está consultado y false en caso contrario.

Librería Commerce

Esta librería cuenta con métodos que se conectan al api del commerce de Yalo.

Nota: Estos métodos retornan una variable tipo JSON.

```
Commerce.init(name: string, url?: string) -> headlessResponse
```

Inicializa el commerce

name: nombre de la tienda creada en el apartado commerce.
url: url del headless. Se puede omitir este parámetro.

Formato de respuesta

```
{
  "status": "ok",
  "data": {
    "storefrontName": "my-storefront"
 }
}
```

status: retorna ok, si la inicialización del commerce fue correcta.

```
Commerce.sessionCreate(type: 'code' | 'phoneNumber', value: string) -> headlessResponse
```

Crea la sesión para que el usuario pueda acceder a la tienda correspondiente.
Nota: solo usuarios registrados en la tienda pueden acceder a esta; dado que para crear la sesión se necesita el code o el phoneNumber registrados previamente.
type: Tipo de código con el cual se va a crear la sesión, se puede iniciar mediante code; que es código asignado al usuario cuando se registra en la tienda o el phoneNumber que es el número de teléfono registrado.
value: valor del código

Formato de headlessResponse
Devuelve los detalles del usuario, de la tienda y el estatus de la acción.

```
{
  "status": "ok",
  "data": {
    "id": "62d096e623e6fd4021966afe",
    "customFields": null,
    "workflow": {
        ...
    },
    "configuration": {
      "checkoutRules": {
        ...
      }
    },
    "customer": {
```

```
        ...
    }
  }
}
```

`Commerce.cartGet() -> headlessResponse`

Retorna el carrito de la sesión que este creada en el momento.
Formato de headlessResponse

```
{
  "status": "ok",
  "data": {
    "id": "62d096e623e6fd4021966afb",
    "items": [{
        ...
    }],
    "total": 98.5600004196167,
    "status": "IN_PROGRESS",
    "warnings": null,
    "customFields": null
  }
}
```

`Commerce.orderCreate() -> headlessResponse`

Crea la orden, teniendo en cuenta el carrito y el usuario de la sesión creada.

Retorna los detalles de la orden y el id de esta.
Formato de headlessResponse

```
{
  "status": "ok",
  "data": {
    "id": "62d096ff23e6fd4021966b69",
    "customerCode": "38313220000148",
    "storeCode": null,
    "items": [
      {
        ...
      }
    ],
    "status": "CREATED",
```

```
    "processedAt": null,
    "externalRef": null,
    "parentOrderUid": null,
    "notes": null,
    "externalErrorMessage": null,
    "externalMessage": null,
    "source": null,
    "customFields": null,
    "total": 98.5600004196167,
    "sequence": 0,
    "checkoutRules": [],
    "postOrderCheck": null,
    "preOrderCheck": null,
    "activePromotions": [],
    "createdAt": {
      ...
    },
    "updatedAt": {
      ...
    }
  }
}
```


`Commerce.orderConfirm(orderId: string) -> headlessResponse`

Cambia el estatus de la orden a confirmada.

orderId: id de la orden a confirmar.

Retorna lo mismo que el orderCreate.

Commerce.sessionExpire() -> headlessResponse

Cierra la sesión; es importante cerrar la sesión después de que el cliente cancela la orden o la confirma.

Formato de headlessResponse


```
{
  "status": "ok",
  "data": {
    "id": "62d096e623e6fd4021966afe",
    "customFields": null,
    "status": "EXPIRED",
    "cartUid": "62d096e623e6fd4021966afb",
    "workflow": {
```

```
        ...
    }
  }
}
```

Global Actions

Las acciones globales son acciones condicionales que se pueden ejecutar en cualquier punto de la conversación, siempre y cuando, en el paso (step) en que el usuario esté tenga habilitado la opción «consider global actions». Un ejemplo de aplicación puede ser que cuando el usuario escriba la palabra “Hola” regrese al inicio del flujo. Esto es con el fin de que no se acabe el flujo de interacción en caso de que el usuario requiera usar el bot nuevamente en menos de 24 horas.

Para crear una acción global se debe acceder al siguiente apartado y seleccionar el botón «*Add Action*»:

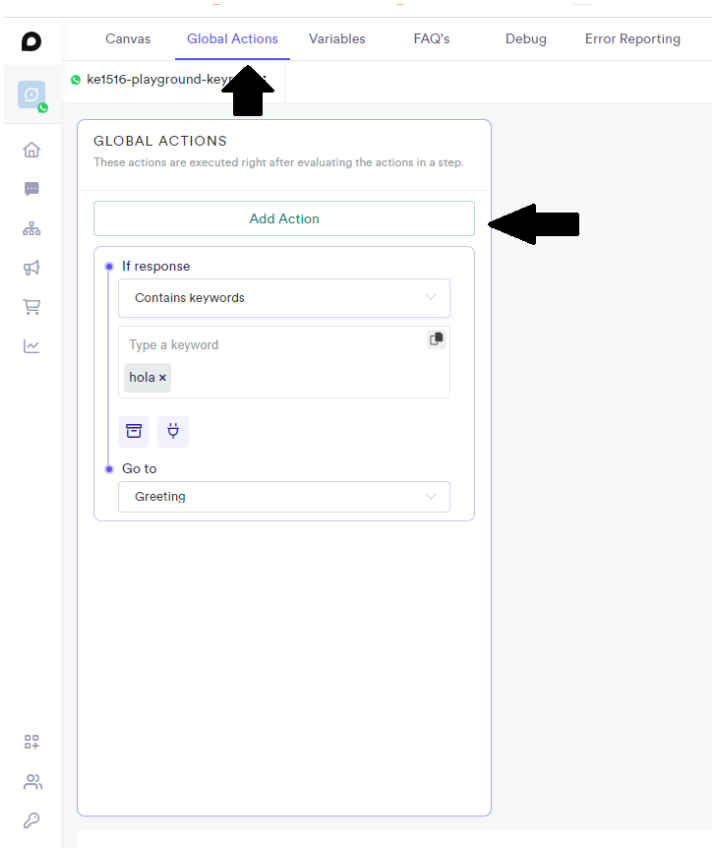


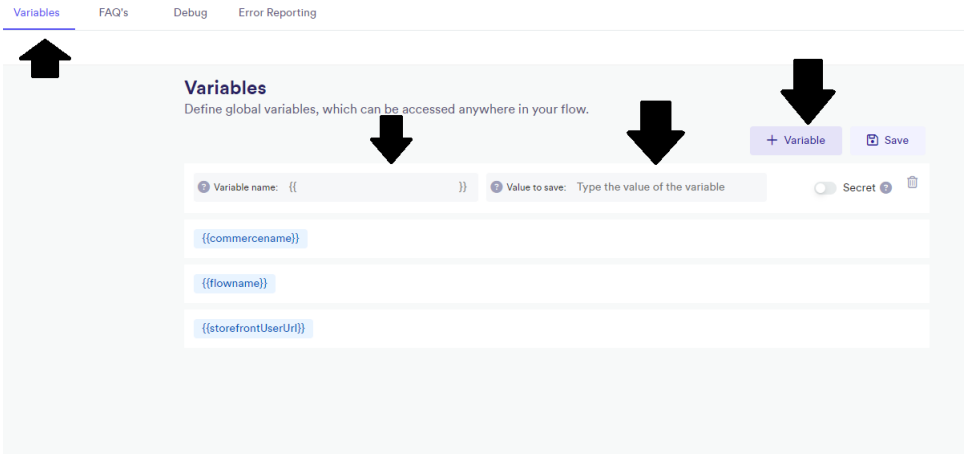
Figura 31: Interfaz de global actions

Ahí se elige la condición deseada para ejecutar la acción y el paso a donde se va a dirigir.

Variables

Al igual que las acciones globales, también existe un apartado para las variables globales, estas variables son de tipo estáticas y pueden ser accedidas desde cualquier parte del flujo.
Para crear una variable se ingresa al apartado Variables y seleccionamos la opción «*+ Variable*» este botón habilita los

campos para poner el nombre de la variable y el valor a guardar en ella.



Finalmente, podemos dejar este valor de manera censurada con la opción «*Secret*» y guardarla con el botón «*Save*»

FAQs

Para cargar preguntas frecuentes, existe un apartado dedicado donde se pueden almacenar en forma de tabla, dichas preguntas Yalo Studio permite cargarlas mediante un archivo separado por comas (.CSV) o una a una en la interfaz interna de Yalo. A continuación, se muestra la interfaz para cargar dichas preguntas señalando el botón (+) «*add row*» y a algunos ítems de interés:

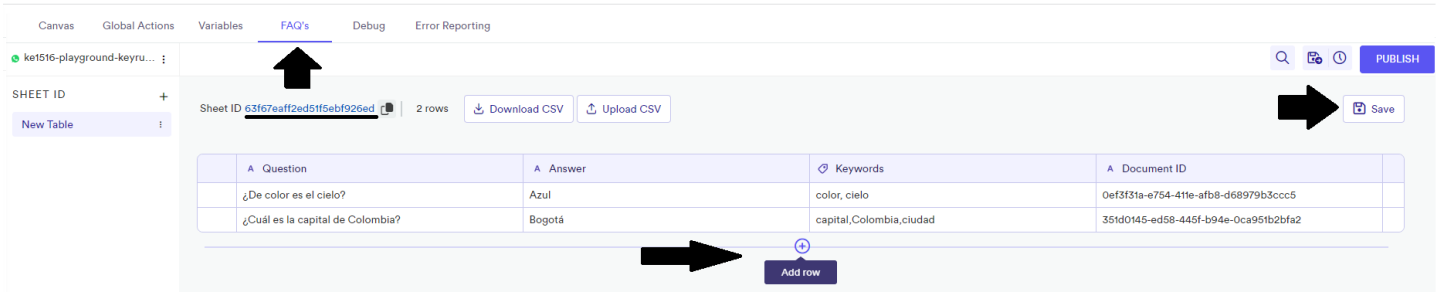


Figura 32: Interfaz configuración de FAQs

Para conectar la tabla con el flujo se recomienda usar la plantilla FAQs de Yalo y en el apartado de código de Lua cambiar el *Sheet ID* como se ve en la figura:

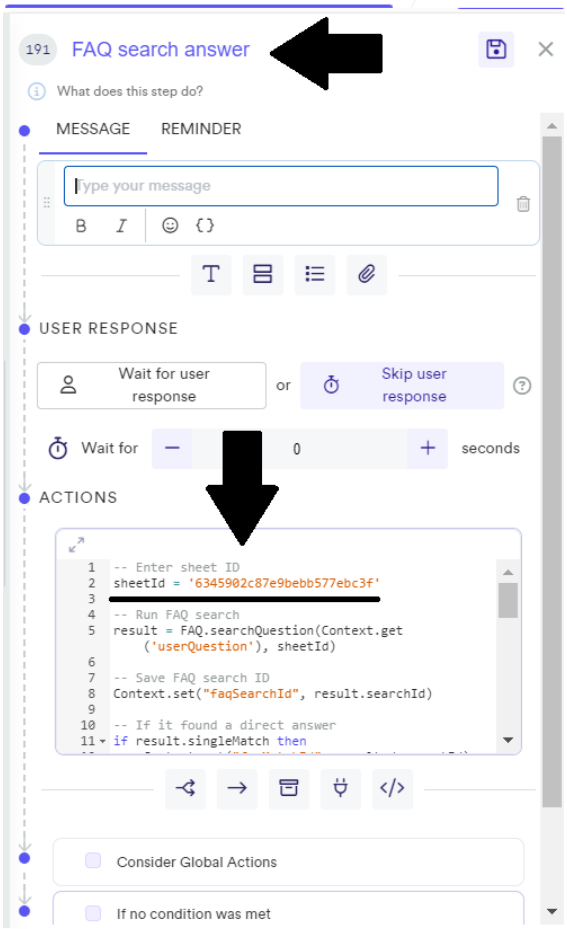


Figura 33: Step plantilla YALO sheetID

Debug

El apartado de *debug* sirve para revisar el valor de las variables de contexto y perfil de un usuario/número en particular. Se utiliza agregando el número del usuario que desea ver y presionando el botón «*Search*».

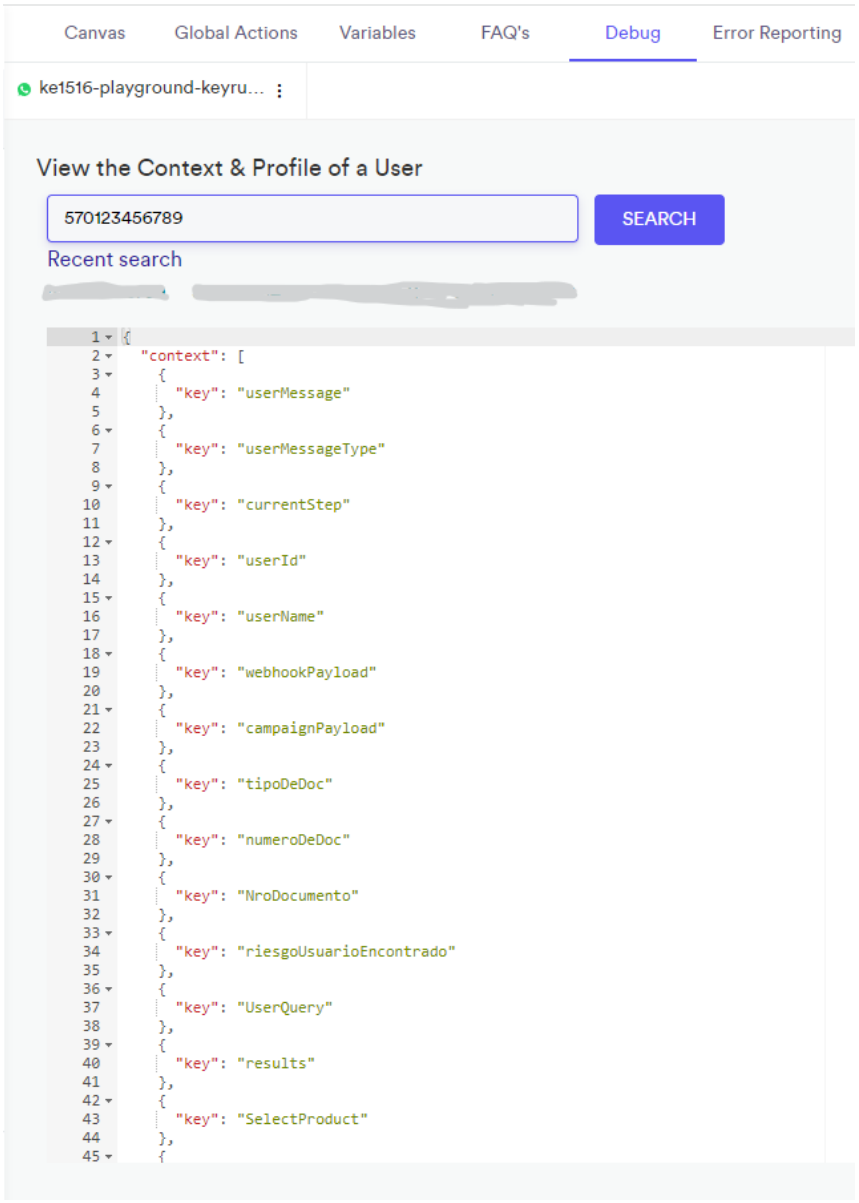


Figura 34: Interfaz de Debug

Error Reporting

Por último, la sección de *error reporting* sirve para identificar en que «step» ocurrió el error y de que tipo fue, generalmente los errores mostrados aquí son por procesos con código Lua o *bad request* en llamados API.

Engagement

Plantillas de mensaje

El requisito para activar esta interfaz es tener un número asignado en el flujo como se menciona al principio, sin el numero asignado no se podrá ingresar.

Las plantillas de mensajes se utilizan para enviar las campañas, estas deben ser aprobadas por WhatsApp y tarda alrededor de un día en aprobarse o rechazarse.
Para crear una nueva plantilla de mensaje se ingresa a la interfaz y se oprime el botón «CREATE NEW». Como se ve en la figura:

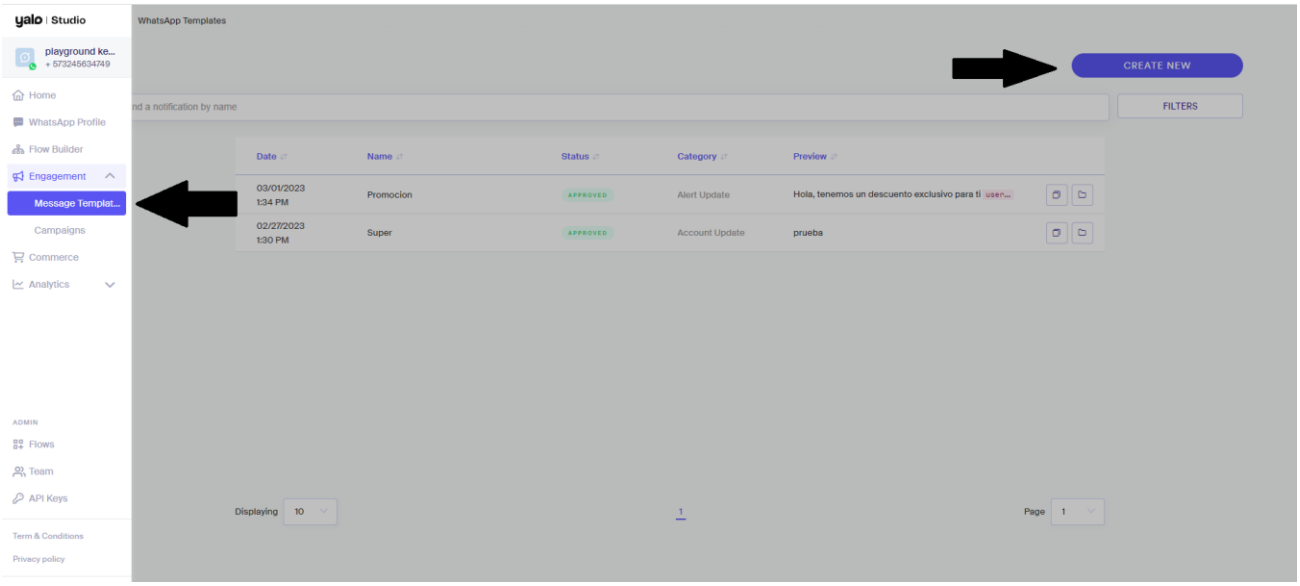


Figura 35: Interfaz message templates

Después, podemos crear la plantilla asignando un nombre, categoría, lenguaje, tipo de mensaje, y el mensaje como tal, al lado derecho se puede observar una *preview* y resumen de la configuración del mensaje y un botón para hacer el envío a WhatsApp para aprobación, el tiempo de aprobación es de aproximadamente un día.

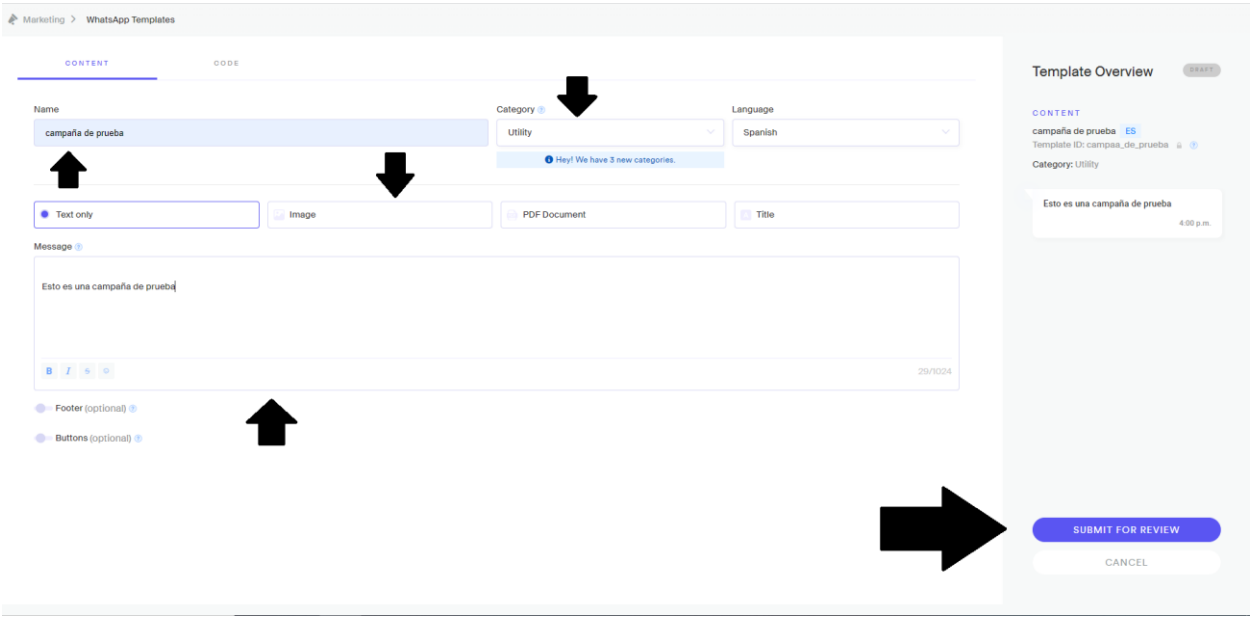


Figura 36: Interfaz configuración de Whatsapp templates

Una vez aprobada la plantilla aparecerá de la siguiente manera:

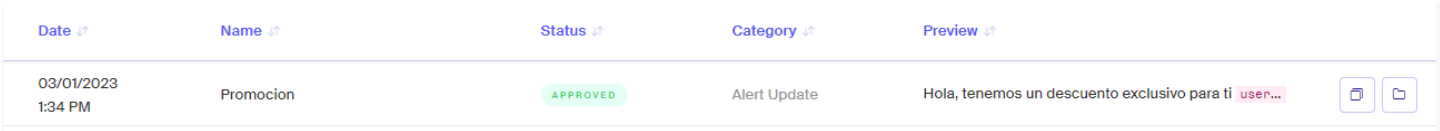


Figura 37: Tabla de plantillas

Al ingresar aparece la siguiente interfaz donde podemos contectar la plantilla a un flujo de tipo campaign starter (explicado en la sección de canvas).

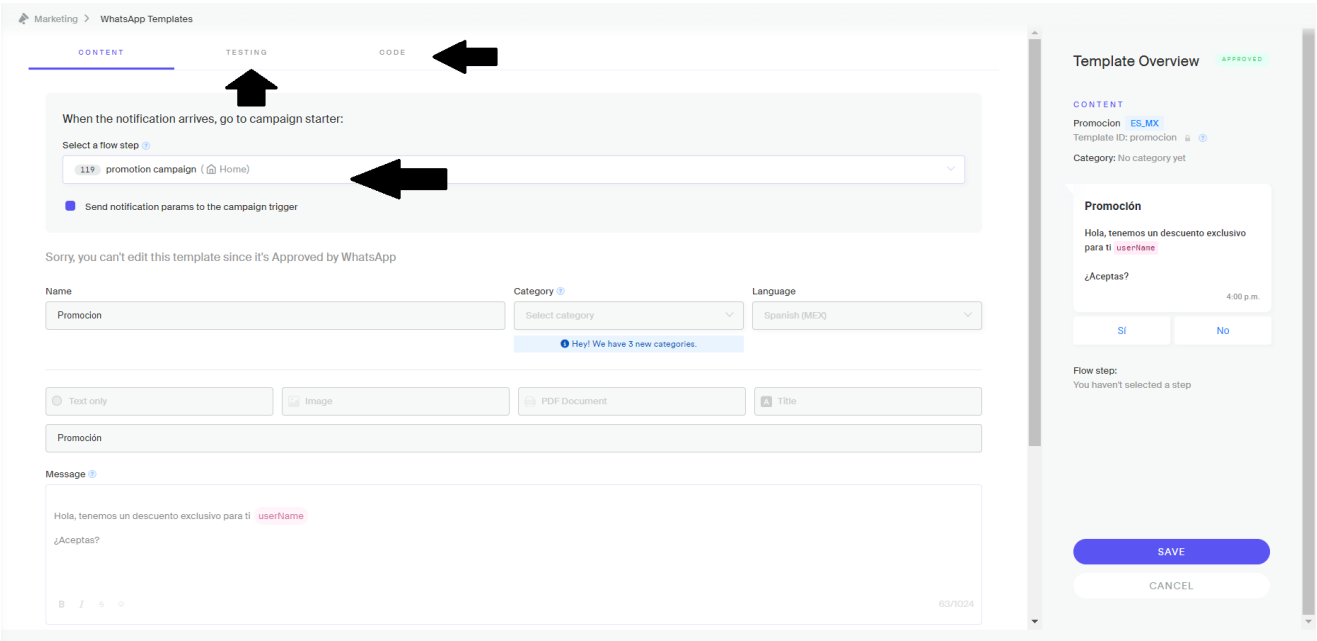


Figura 38: Interfaz content

Adicionalmente, existen dos apartados más, uno para hacer el testeo de la plantilla y otro para disparar la plantilla desde un servicio externo mediante un POST.

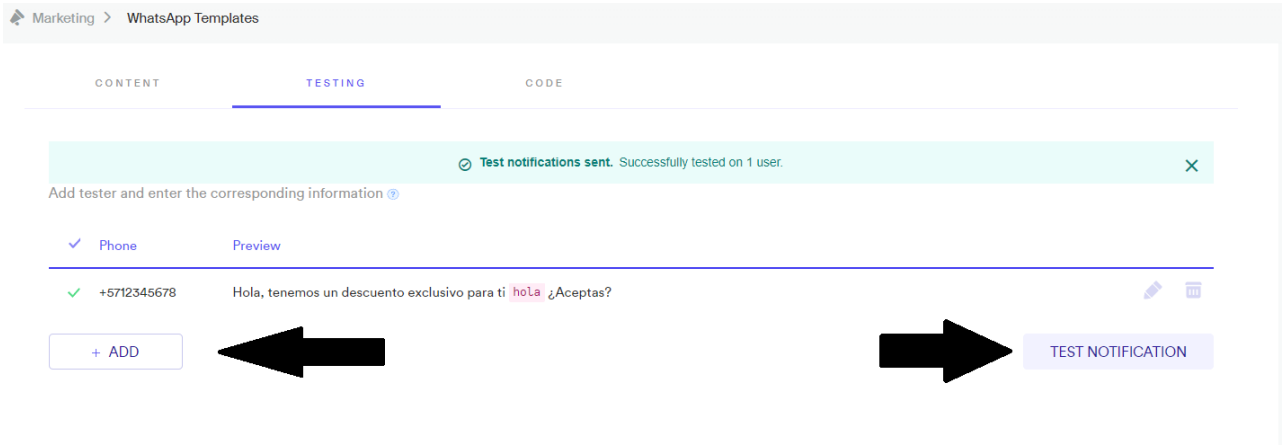


Figura 39: Interfaz testing

Figura 40: Interfaz de code

Para usar el código para hacer el post debe activar una API Key en la siguiente interfaz:

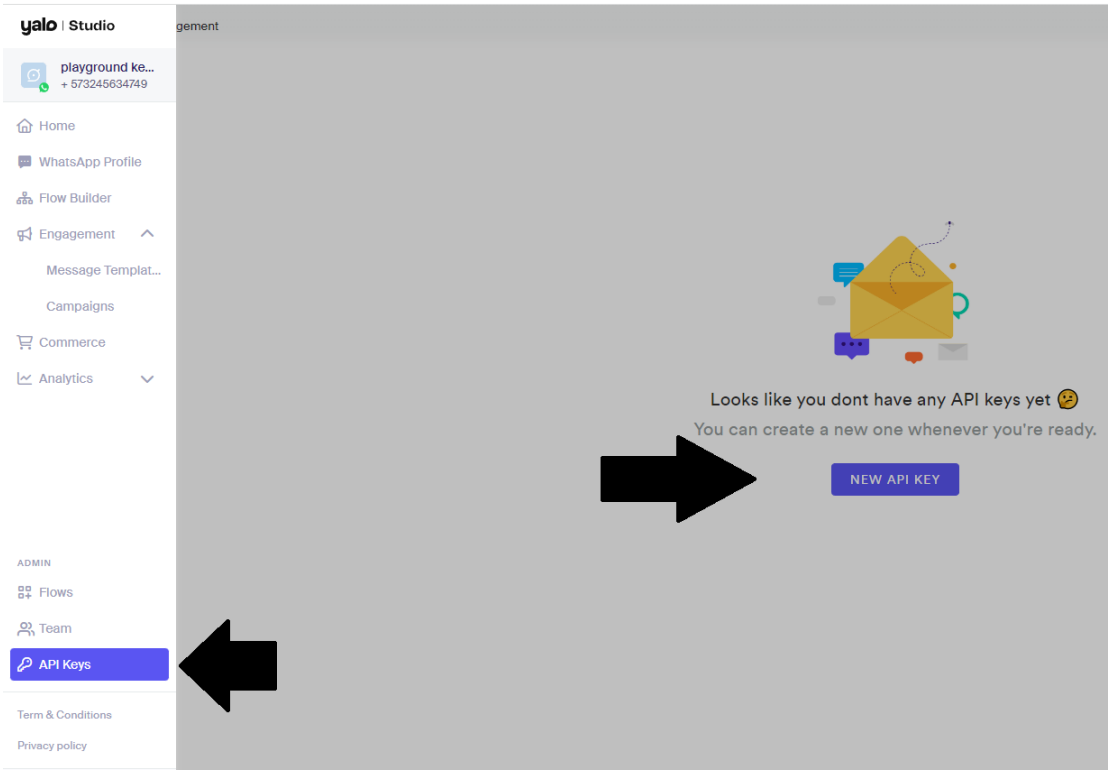


Figura 41: Ejemplo como crear API key

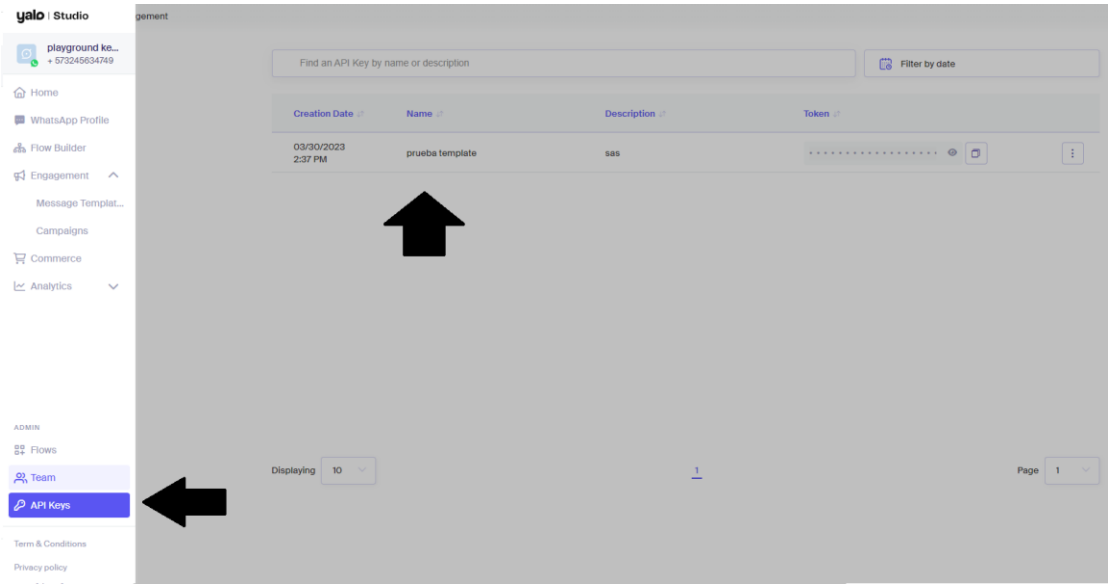


Figura 42: Interfaz de API keys

De esta manera aparecerá en el selector de la figura anterior llamado «api key»

Campañas

El apartado de campañas sirve para programar el envío de una campaña a partir de una plantilla de mensaje, su interfaz se ve de la siguiente manera y también es requerido que el flujo tenga asignado un número de teléfono.

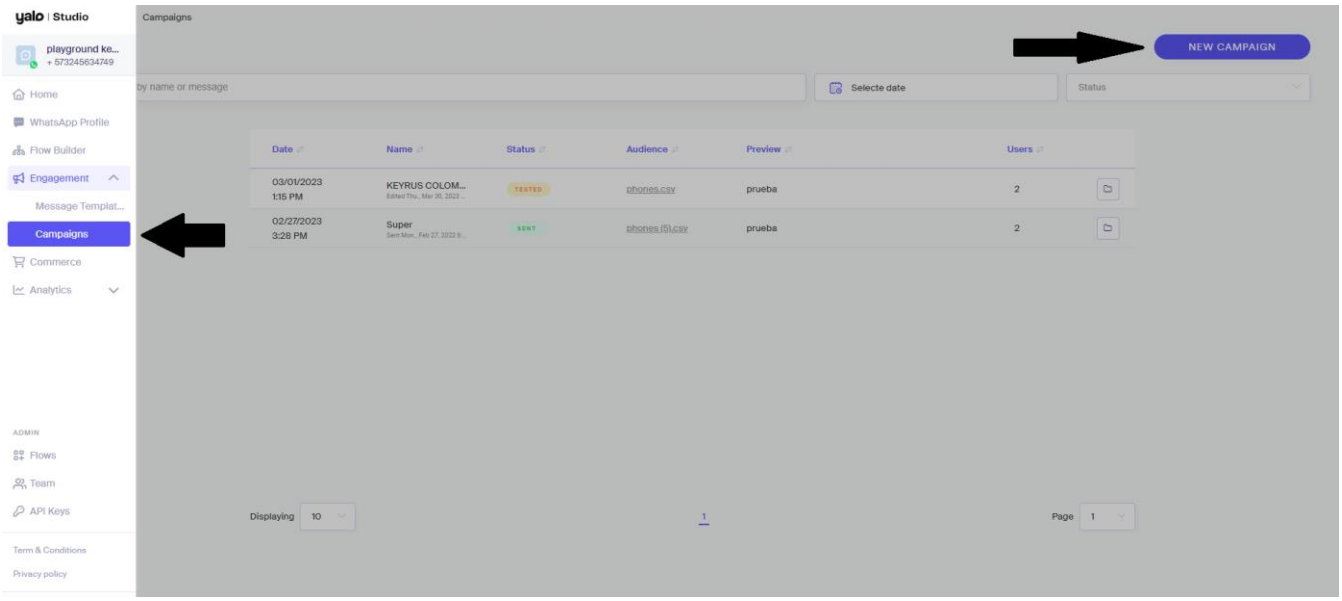


Figura 43: Interfaz de campaigns

Al crear una nueva campaña la interfaz de configuración es la siguiente:

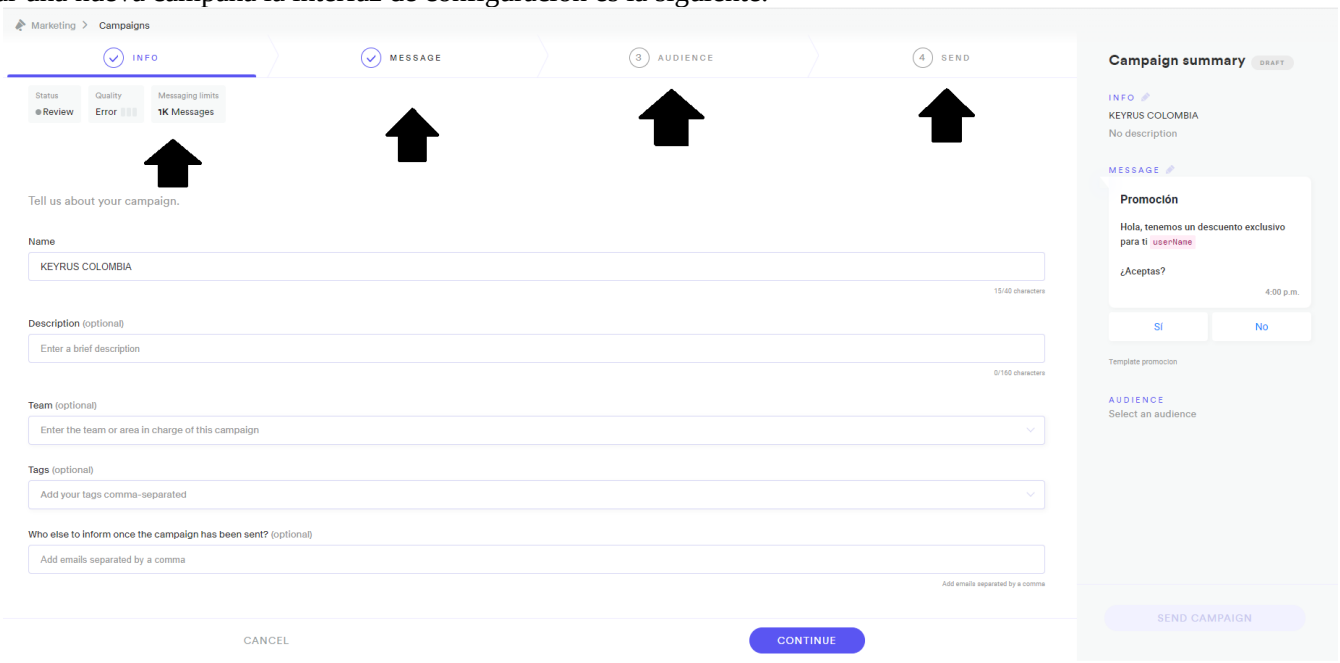


Figura 44: Interfaz para crear campañas

En el apartado de siguiente elige la plantilla de mensaje (*message*) que desea usar, en audiencia (*audience*) carga los números de los usuarios en formato csv a los que le va a enviar la campaña y finalmente en enviar (*send*) programa la fecha y hora en que se enviará la campaña.

Commerce

Como crear una storefront

Para crear una tienda accedemos al apartado Commerce como se ve en la siguiente figura:

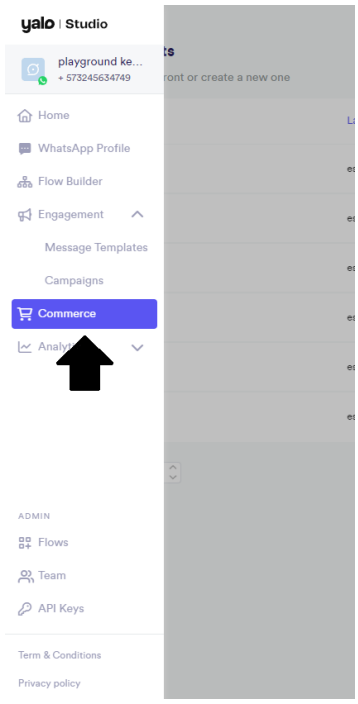


Figura 45: Menú Yalo

Ahí se encontrará las tiendas creadas y la opción crear una nueva tienda, para crearla debe oprimir el botón llamado «*NEW STOREFRONT*» ubicado en la esquina superior derecha de la interfaz. Al oprimir el botón se abre un menú modal solicitando el nombre de la tienda y el correo del dueño
(IMPORTANTE: elegir bien el nombre de la tienda porque dicho nombre no se puede modificar)

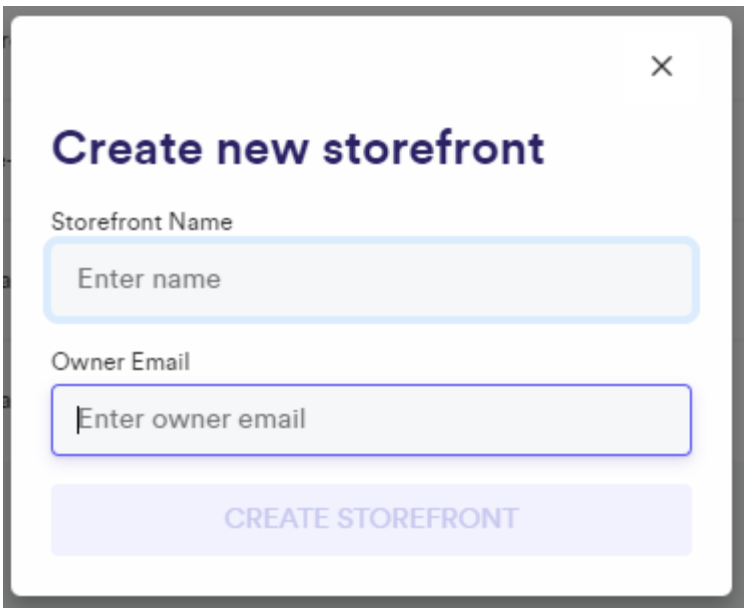


Figura 46: Formulario creación de storefront

Configuración de la tienda

En la interfaz de la tienda se tiene acceso a las configuraciones básicas y a las reglas de negocio como se ve en la siguiente figura:

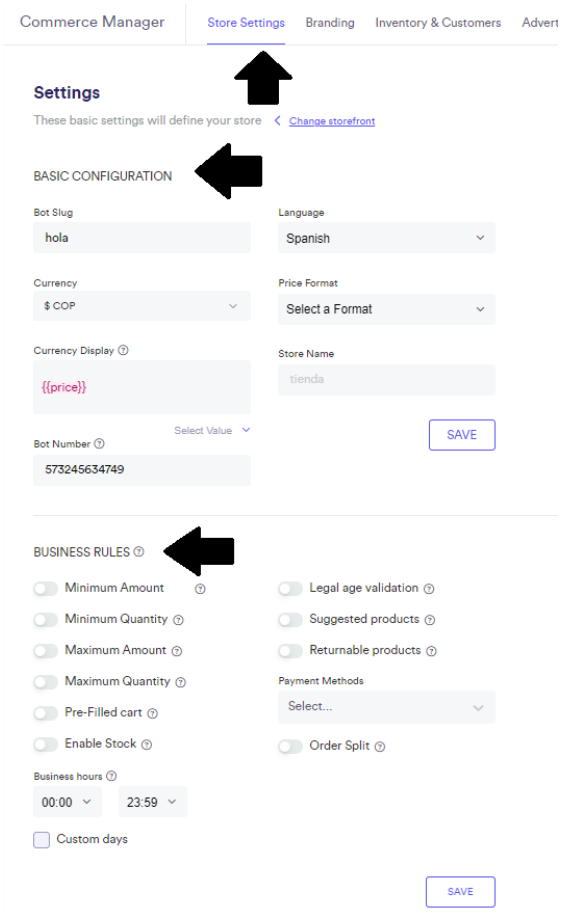


Figura 47: Interfaz de store settings

En las configuraciones básicas se permite modificar cosas como el lenguaje de la tienda, la divisa, la forma en que se muestra la divisa y el número del bot a donde se redirecciona después de hacer la compra.
Para las reglas de negocio hay una serie de opciones seleccionables sobre la cantidad máxima y mínima de los productos y el dinero, así como las horas de negocio, entre otros.

Modificación de la marca

La sección de marca (*branding*) permite agregar logos, modificar la forma en que se muestran y ver de forma preliminar como se vería en el contexto real.

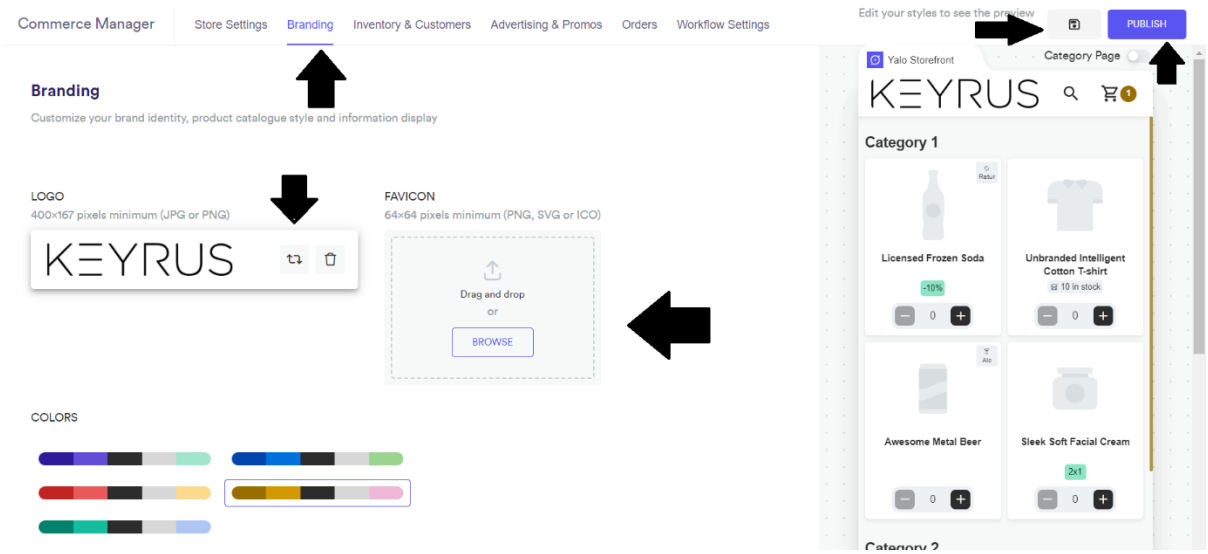


Figura 48: Interfaz de branding

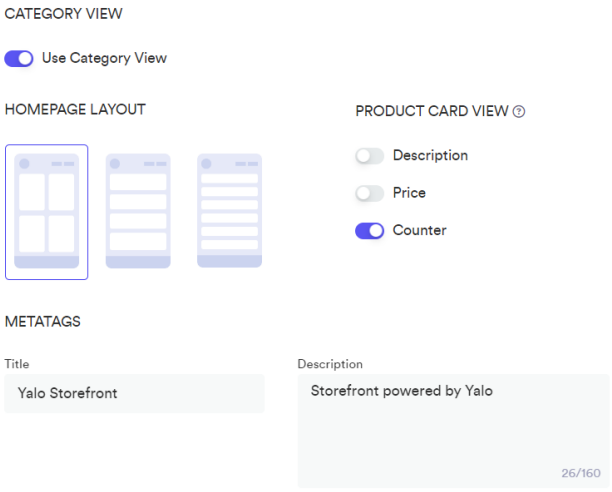


Figura 49: Interfaz de category view

Inventario y clientes

En este apartado se cargan los productos, precios, stock y clientes de la tienda en cuestión, para cargarlos, Yalo Studio proporciona unos formatos CSV para llenarlos y subirlos. Para subirlos se debe presionar el botón + ADD PRODUCTS, + ADD PRICES, + ADD STOCKS, + ADD CUSTOMERS según en el apartado en que se encuentre.

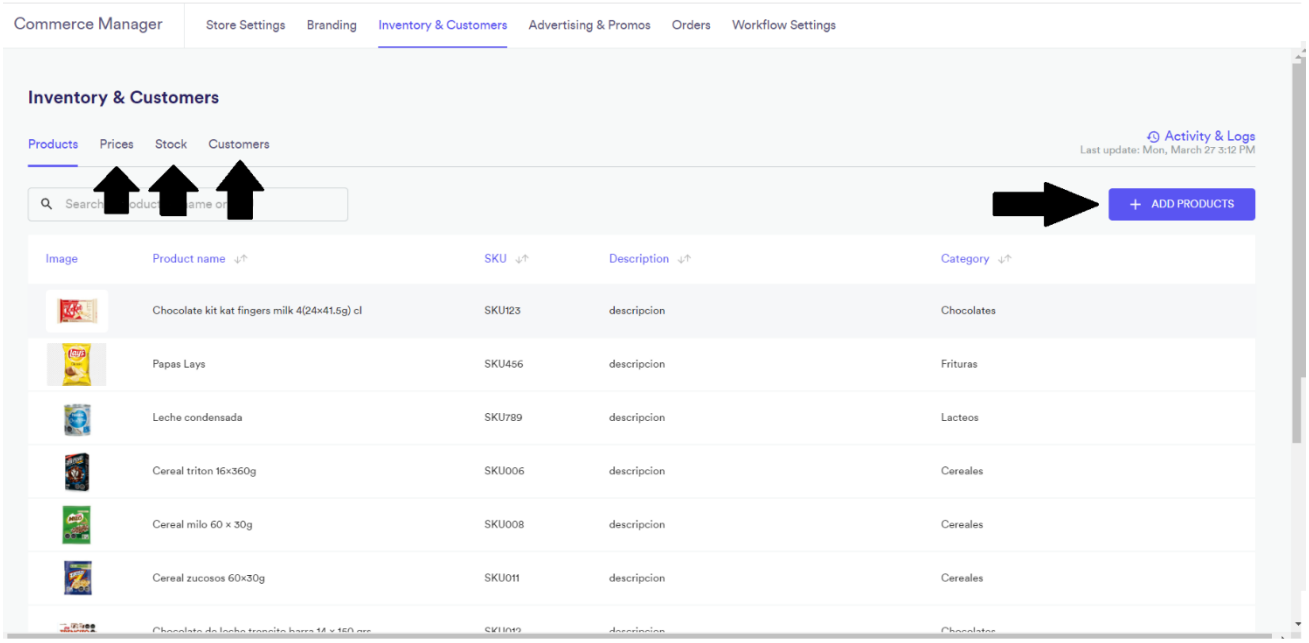


Figura 50:Tabla de products

IMPORTANTE: la forma en que Yalo Studio relaciona las tablas PRODUCTS – PRICES – CUSTOMERS es mediante dos llaves foráneas entre PRODUCTS Y PRICES la llave foránea es la columna SKU y entre PRICES Y CUSTOMERS la llave foránea es la columna TYPE, de esta manera a cada tipo de usuario se le permite asignar un precio diferente por producto como se ve en las siguientes figuras.

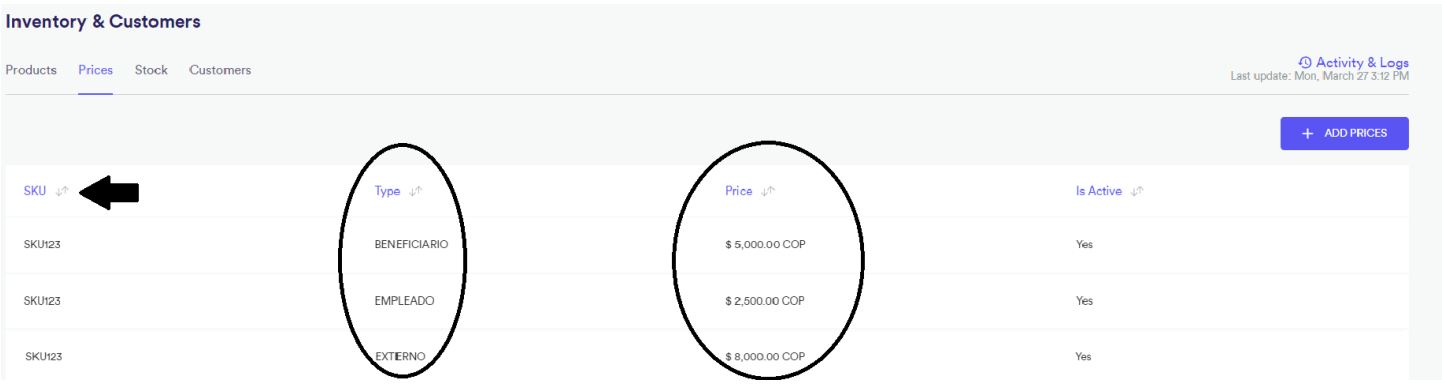


Figura 51:Tabla de prices

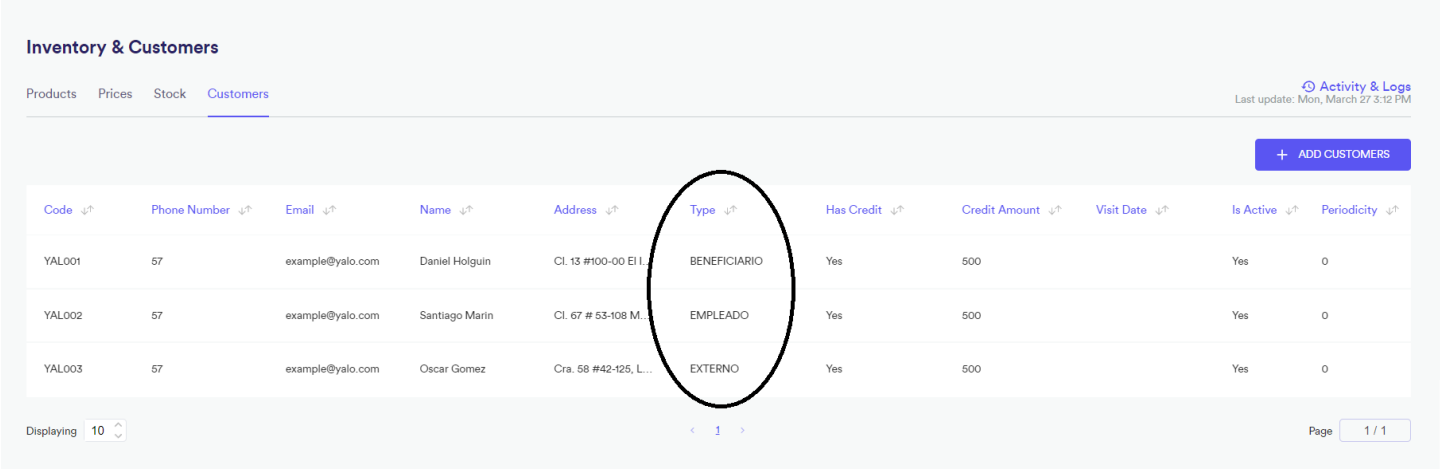


Figura 52: Tabla de customers

Descripción campos CSV para cargar **clientes** en la base de datos de YALO

No	Columna / Dato	Tipo	Notas
01	customerCode	String	Identificador de clave para cada cliente, este campo debe ser único entre todos los clientes.
02	phoneNumber	String	Número de teléfono del cliente.
03	email	String	Correo electrónico del cliente
04	name	String	Es el nombre del cliente que será mostrado en los flujos conversacionales
05	address	String	Domicilio del Cliente.
06	type	String	Categoría del cliente. Este campo se utiliza para referenciar precios. Si los precios se gestionan por tipo de cliente este campo debe ser obligatorio
07	hasCredit	Boolean	Permite configurar si un cliente podrá tener activada o desactivada la opción de realizar un pedido a crédito, conforme lo siguiente: hasCredit=true, El cliente puede hacer un pedido a crédito hasCredit=false, El cliente no tiene la posibilidad de hacer un pedido a crédito.
08	creditAmount	Decimal	Monto de crédito disponible para el cliente. Será posible utilizar el punto decimal, sin embargo, no deberá utilizarse el separador de comas para expresar cantidades. Por defecto 0.

09	visitDate	String	Indica la fecha que se utilizará para calcular/determinar la fecha promesa de entrega que se mostrará al momento de realizar una orden/pedido, esta fecha deberá ser proporcionada por el Negocio enel siguiente formato: **YYYY-MM-DD (Año, Mes, Día)**
10	isActive	Boolean	Indica si un cliente está activo o no. En el servicio de donde se obtiene este dato, el valor debe ser un número entero de valor true o false, lo que significa falso para el primero y cierto para lo último. Este campo obligatorio podría no existir, en tal caso será configurado con el valor predeterminado de verdadero.

11	lockedStoreCode	String	Atributo que permite asociar a un cliente con un determinado Distribuidor. Deberá existir una correlación entre este atributo y el valor del atributo **Code** que pertenece al archivo de **Stores**.
12	zipcode	String	Código postal.

Descripción campos CSV para cargar **productos** en la base de datos de YALO

No	Columna / Dato	Tipo	Notas
01	sku	String	Identificador de código de producto, este valor debe ser único entre todos los productos.
02	name	String	Nombre del producto que se mostrará.
03	category	String	Categoría de producto, esto se utilizará para mostrar la lista de categorías en el WebView.
04	priority	Int	Posición del producto cuando se muestra en el WebView. Los números menores se mostrarán primero.
05	description	String	Descripción del producto que se mostrará
06	tags	String	Etiquetas de producto. Para varias etiquetas, el valor debe ser una cadena separada por comas. por ejemplo: "etiqueta1, etiqueta2, etiqueta3"
07	size	String	Tamaño del producto si es que aplica
08	imageURL	String	URL de la imagen que se mostrará en la vista web. Si el valor es nulo ovacío es posible que el Webview no funcione.
09	packageType	String	Tipo de presentación/empaquetado del producto. por ejemplo, caja o unidades.

10	unitsPerPackage	Number	Unidades por caja para el producto, si el paquete del producto está en cajas. Se deberá capturar la cantidad de piezas por caja. Este es un dato informativo.
11	unitDivision	Number	Este es un dato informativo, está pendiente de implementar para futuras actualizaciones en la plataforma de Yalo, por el momento, no es necesario que se capture
12	divisionsByUnit	Number	Este es un dato informativo, está pendiente de implementar para futuras actualizaciones en la plataforma de Yalo, por el momento, no es necesario que se capture.
13	isActive	Boolean	Indica si el producto debe marcarse como activo, Predeterminado: True. Si el valor es False el producto se eliminará de forma lógica, es decir no de forma definitiva. Este atributo podrá ser utilizado para los casos en que un producto no cuente con stock disponible, pueda ser marcado como no activo y por consiguiente no podrá ser adquirido a través de la solución conversacional.
14	brand	String	Nombre de la marca a la cual pertenece el producto, Ejemplo: Marca Libre.

Descripción campos CSV para cargar **precios** en la base de datos de YALO

No	Columna / Dato	Tipo	Notas
01	sku	String	Identificador de código de producto. Este valor debe coincidir con un SKU de un producto existente en el archivo llamado Productos.
02	price	Decimal	Precio neto del producto que será asignado en la plataforma de Yalo
03	key	String	Identificador para agrupar el sku del producto y su precio, con un type de cliente. Para que esto ocurra, el valor de este campo debe corresponder con el type del cliente.
04	isActive	Boolean	Indica si el precio debe marcarse como activo, Predeterminado: True. Si el valor es False el precio se eliminará de forma lógica, es decir no de forma definitiva.

Descripción campos CSV para cargar **stocks** en la base de datos de YALO

No	Columna / Dato	Tipo	Notas
01	sku	String	Identificador del producto al cual se le asignará un determinado stock disponible, el valor de este atributo deberá existir como identificador en el archivo de productos.
02	key	String	Identificador para determinar a qué store pertenece el producto y stock definido, el valor de este atributo deberá existir en el archivo de stores.
03	stock	Number	Cantidad de stock disponible en el C-Commerce, expresado en números enteros, aplicable al producto y store definido.
04	isActive	Boolean	Permite activar/desactivar un stock conforme lo siguiente: isActive = True, define que el stock está activado isActive = False, define que el stock está desactivado

Publicidad y promociones (Advertising & promos)

Esta sección ofrece dos herramientas para ejecutar estrategias de mercadeo, la cuales son *banners* (imágenes llamativas de portada) y *promotions* (distintos tipos de promociones para los productos y tipos de usuario) como se ve en la siguiente figura

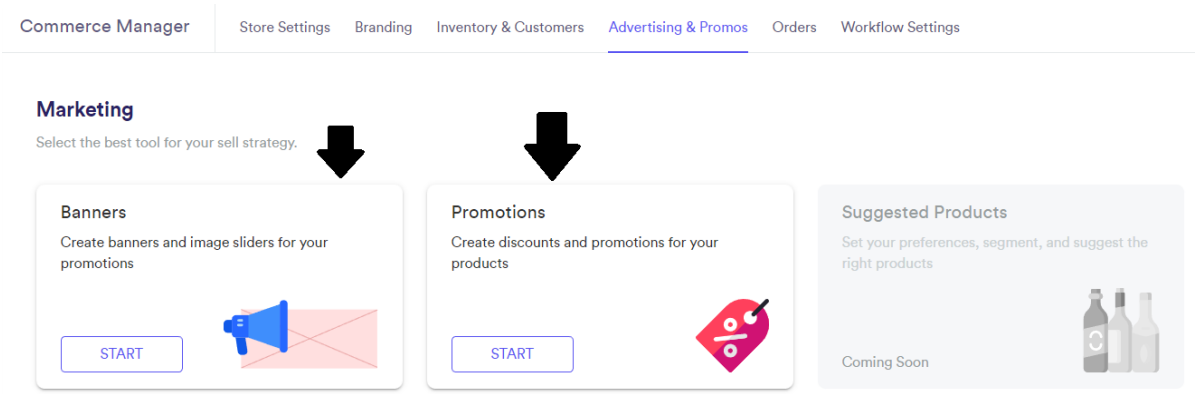


Figura 53: Interfaz de marketing

Para el apartado de Banners la interfaz es muy similar la de marca (*branding*) donde podemos cargar las portadas y ligarla a una categoría o producto particular y se ve de la siguiente forma

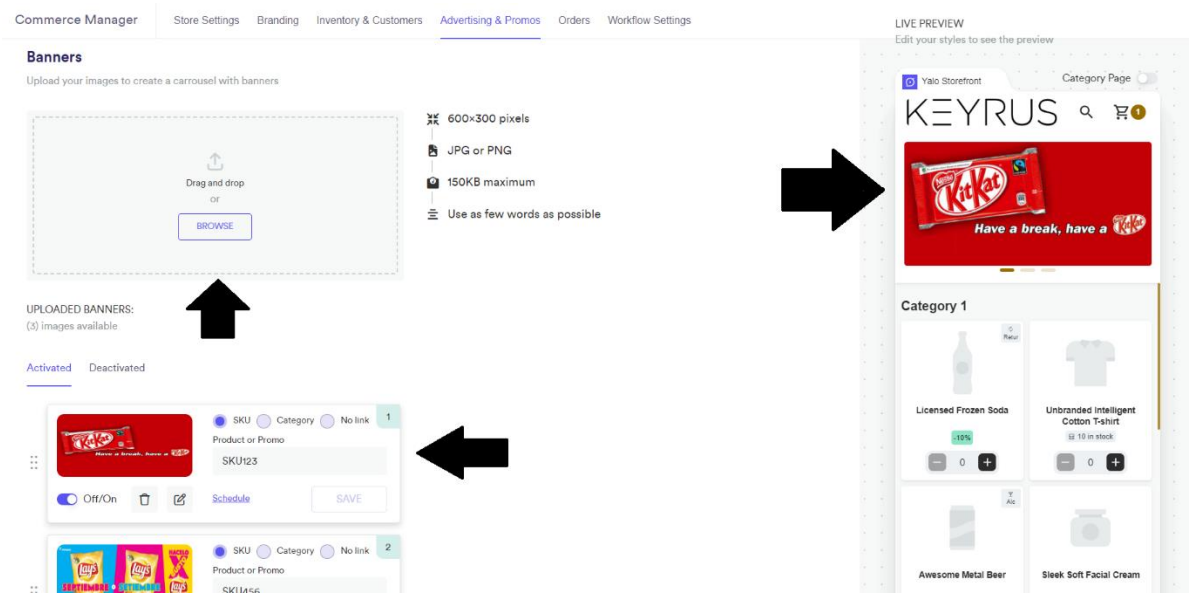


Figura 54: Interfaz de banners

Por el lado de promociones tenemos una interfaz similar a la carga de productos. Para cargar las promociones se hace mediante un archivo CSV

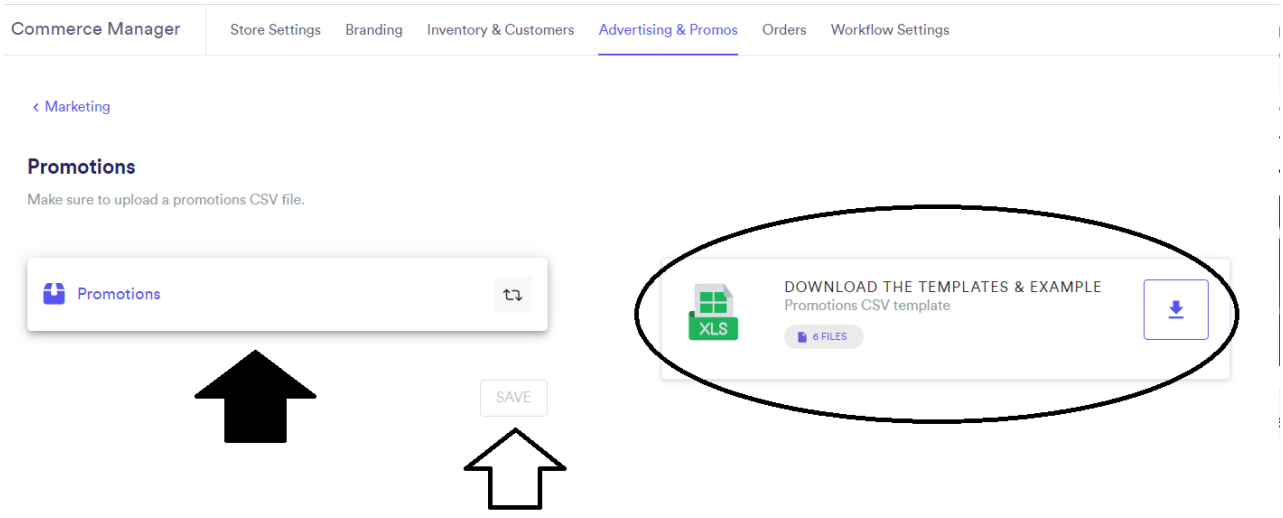


Figura 55: Ejemplo interfaz para subir promociones

Descripción campos CSV para cargar **promociones** en la base de datos de YALO

No	Columna / Dato	Tipo	Notas
01	type	String	Define el tipo de promoción que se aplicará, los posibles valores son: **DIRECT:** Promoción directa, Ejemplo: 10% de descuento en un determinado SKU, aplicable a todos los clientes. **VOLUME:** Promoción con base a volumen de compra, Ejemplo: 7% de descuento al comprar un mínimo de 15 productos de un determinado SKU **COMBO:** Promoción tipo Combo, Ejemplo: Compra un producto con el SKU01 y te obsequiamos otro producto delSKU02.
02	customerType	String	Indica el tipo de cliente para el cual será aplicable la promo definida, el valor de este atributo deberá existir en el archivo de Clientes dentro del atributo "type" Si este valor está en Blanco, la promoción será aplicada a todos los clientes.
03	customerBuys_quantity	Integer	Indica cuántos ítems del valor definido en el atributo "customerBuys_items" se requieren adquirir para que aplique la promoción. Ejemplo: 3 items
04	customerBuys_amount	Decimal	Indica el monto/importe mínimo de compra que se deberá adquirir con respecto al valor definido en el atributo "customerBuys_items" para que aplique la promoción. Ejemplo: 1,500 pesos
05	customerBuys_minimum	Integer	Se utiliza para definir la cantidad mínima de ítems que se deberán adquirir con respecto al valor definido en el atributo "customerBuys_items" para que aplique la promoción. Ejemplo: 5, implica que se deben adquirir como mínimo 5 ítems para que la promo sea aplicada.
06	customerBuys_items	String	Indica el SKU que debe ser añadido al carrito de compras para habilitar la Promoción. Debe coincidir con el formato del SKU proporcionado en el Catálogo de Productos.
07	customerGets_quantity	Integer	Indica la cantidad de SKUs especificados en "customerGets_items" que serán añadidos al carrito una

Órdenes

La interfaz de ordenes permite ver la ordenes realizadas en la storefront.

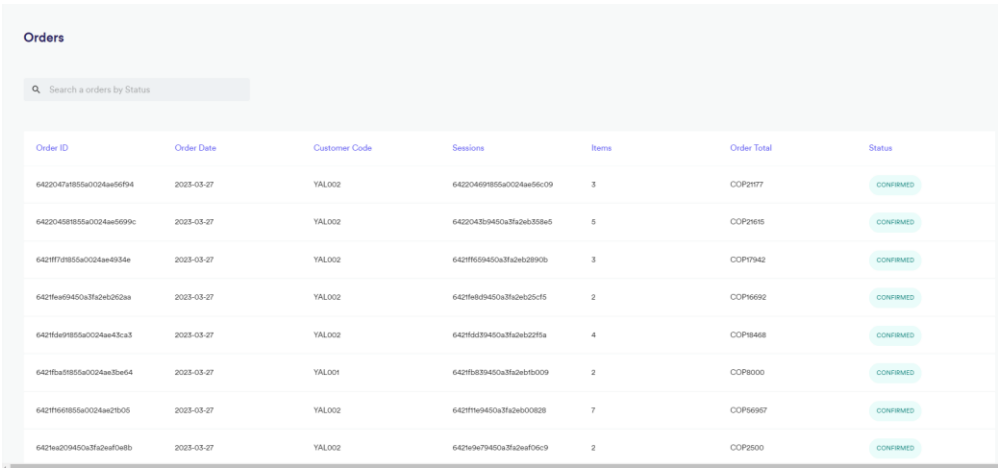


Figura 56: Lista de ordenes

Configuraciones de flujo de trabajo

Esta interfaz permite conectar el flujo de trabajo con la *storefront*, se conecta mediante un *webhook* previamente creado en el flujo de trabajo.

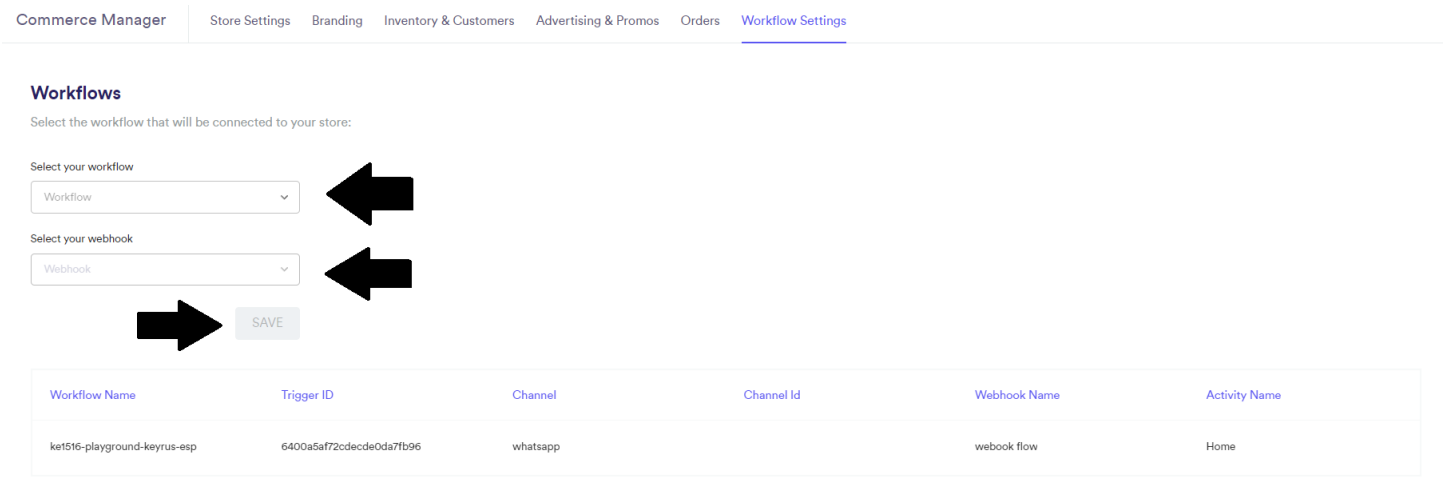


Figura 57: Configuración del workflows

De esta manera una vez el usuario termine la interacción con la tienda, se dispara el *webhook* para siga el flujo deseado.

Código Lua storefront:

1. **Para iniciar una sesión de comercio y generar una URL válida para el usuario se hace por medio del siguiente código Lua.**

```
--se toma el nombre de la tienda establecido en la sección de variables
storefrontName = Global.get("commercename")
```

```
--Este url también está declarado en la sección de variables y se lo proporciona el equipo de Yalo. SIN EMBARGO, NO ES NECESARIO PARA INICIAR LA TIENDA NI GENERAR EL URL, SOLO CON commerce.init(storefrontName) FUNCIONA

storefrontUserUrl = Global.get("storefrontUserUrl")

--La librería commerce permite inicializar el comercio usando el .init y pasándole el nombre de la tienda y url de usuario.
initData = Commerce.init(storefrontName,storefrontUserUrl)



--Se guarda la data de inicio en el contexto
Context.set("initData", initData)

--Se toma el número de identificación ingresado por el usuario previamente
userDocument = Context.get("NroDocumentoCommerce")
--Este número/identificación tiene que ser igual al campo denominado code o phoneNumber en caso de que la sesión se cree con phoneNumber

tipo = "code" --Puede ser con phoneNumber

--Se crea la sesión de comercio con los datos del customer registrado en el comercio
sessionData = Commerce.sessionCreate(tipo, userDocument)

--Se guarda la data de sesión en el contexto
Context.set('sessionData', sessionData)

--Si el usuario existe en el comercio, entra al condicional y genera la url para ingresar
if sessionData.status == 'ok' then
    local baseUrl = "https://commerce.yalochat.com/"
--Los .. sirven para concatenar en Lua

    url = baseUrl .. storefrontName .. "/" .. sessionData.data.id

    nombreuser = sessionData.data.customer.name
    address = sessionData.data.customer.address

--Se guarda la Url en el contexto y demás datos de interés

    Context.set('rawUrl', url)
    Context.set('CustomerName', nombreuser)
    Context.set('address', address)
    Workflow.branch(54)
```

```
--Sino se encuentra el usuario en el comercio el estatus marca 404
elseif sessionData.status == 404 then
    Workflow.branch(57)
end
```

2. Código para obtener los datos del carrito de la storefront

```
   --Se guarda la data del carrito en el contexto
   cartData = Commerce.cartGet()
   Context.set("cartData", cartData)

   --Si la sesión de es correcta, entra al condicional
  if cartData.status == 'ok' then
   --Guarda los artículos que el usuario compró en el contexto
    Context.set("CartItems", cartData.data.items)

   --Si hay productos con promociones activas, se guarda en el contexto también para
   mostrarlos con el precio correcto en carrito
     Context.set("activePromotions", cartData.data.activePromotions)
       if Context.get("activePromotions") == "[]" then
               Workflow.branch(58)
       else
            Workflow.branch(103)
       End
  else
   --Expira la sesión si no cumple el condicional
      sessionResponse=Commerce.sessionExpire()
      Debug.print(sessionResponse)
  End
```

Nota: El JSON que arroja ***Commerce.cartGet()*** tiene una estructura un poco compleja con respecto a la relación de los productos que cuentan con descuento, pues está como un arreglo aparte de ***cartData.data.items.*** Esto dificulta la obtención del precio del artículo que tiene descuento para mostrarlo en el mensaje del carrito. Un ejemplo de posible solución para mostrar el precio correcto de los artículos con descuento es el siguiente:

Branch (103):

```
--Se toma el arreglo de promociones activas del contexto guardado en el anterior código
promocionesActivas = JSON.decode(Context.get("activePromotions"))

--Se toma el arreglo de artículos del carrito del contexto guardado en el anterior código
ItemsCarro = JSON.decode(Context.get("CartItems"))

--En Lua existe una función llamada .insert, pero en Yalo no funciona, por lo que se
recurrió a hacerlo con un index creciente.
```

```
index = 1

--Arreglo o tabla vacía donde guardaremos los productos con el precio final.
products={}

for i=1,#ItemsCarro do
    valorConDescuento=-1
    porcentaje = 0
  for j=1, #promocionesActivas do

--Si el producto del carrito está en el arreglo de promociones activas se guarda el precio del articulo con la promoción incluida

    if ItemsCarro[i].sku == promocionesActivas[j].promotion.customerGets.items[1] then
          valorConDescuento=promocionesActivas[j].pricePerItem
          porcentaje = promocionesActivas[j].promotion.customerGets.value.percentage*100

--Si el tipo de promoción es tipo combo el precio con descuento se calcula de manera diferente
            if promocionesActivas[j].promotion['type'] == "COMBO" then
                valorConDescuento=promocionesActivas[j].promotion.customerGets.value.perc
entage*ItemsCarro[i].price
            end

            if porcentaje == 100 then
                valorConDescuento = 0
            end
        end
    end
                products[index]={
                ["sku"]=ItemsCarro[i].sku,
                ["name"]=ItemsCarro[i].name,
                ["quantity"]=ItemsCarro[i].quantity,
                ["valor"]=ItemsCarro[i].price,
                ["valorConDescuento"]=tostring(valorConDescuento),
                ["porcentajeDescuento"]=porcentaje
            }
            index = index+1
end

--Se guardan todos los productos, tanto los que tienen promoción como los que no tienen. Los que no tienen en el campo “valorConDescuento” queda con -1 y porcentajeDescuento = 0.

Context.set("articulosPromo", JSON.encode(products))
Workflow.branch(104)
```

3. **Código handlebar para mostrar la información del carrito en el chat cuando hay artículos con promoción.**

```
Branch(104):
{{#each articulosPromo}}

{{#ifEquals this.valorConDescuento '-1'}}

{{this.name}} {{this.quantity}} Unid(s) - ${{this.valor}} CRC c/u

{{/ifEquals}}

{{#ifNotEquals this.valorConDescuento '-1'}}
{{this.name}} {{this.quantity}} Unid(s) - ${{this.valor}} ${{this.valorConDescuento}} CRC c/u (

{{this.porcentajeDescuento}}% de descuento 🏷️)

{{/ifNotEquals}}

{{/each}}

💰 TOTAL: ${{cartData.data.total}} CRC
```

4. **Código handlebar para mostrar la información del carrito en el chat cuando no hay artículos con promoción.**

```
Branch(58):
{{#each CartItems}}

{{this.name}} {{this.quantity}} Unid(s) - ${{this.price}} CRC c/u

{{/each}}

💰 TOTAL: ${{cartData.data.total}} CRC
```

5. **Código para crear la orden**

```
orderCreateResponse = Commerce.orderCreate()
Context.set('orderCreateResponse', orderCreateResponse)

if orderCreateResponse.status == 'ok' or orderCreateResponse.status == 200 then
    Workflow.branch(61)
End
```

6. **Código para confirmar la orden**

```
orderCreatedResponse = Context.get("orderCreateResponse")
orderCreatedId = JSON.decode(orderCreatedResponse).data.id
```

```
orderConfirmedResponse = Commerce.orderConfirm(orderCreatedId)
Context.set("orderConfirmedResponse", orderConfirmedResponse)

if orderConfirmedResponse.status == 'ok' then
    --Se guarda el id en el contexto para proporcionárselo al usuario en el chat
    Context.set("orderId",orderCreatedId)
    --Una vez confirmada la orden se expira la sesión creada
    sessionResponse=Commerce.sessionExpire()
    Workflow.branch(63) --Branch que envía el mensaje de la orden y la dirección
else
    Workflow.branch(62) --Branch que envía un mensaje de error en caso de que la orden
no haya sido confirmada exitosamente
End
```

7. **Código para traer los clientes de la storefront**

```
storefrontName = Global.get('commercename')
session = JSON.decode(Context.get('sessionData'))

headlessAdminUrl = "" --debe solicitarse al equipo de Yalo

query = string.format([[
  query GetCustomers($storefrontName: String!, $filter: FilterCustomerInput) {
  customers(storefrontName: $storefrontName, filter: $filter) {
    id
    type
    code
    phoneNumber
    email
    name
    address
    createdAt
    updatedAt
    hasCredit
    creditAmount
    customFields
    isActive
  }
}]])

variables = string.format([[
    {
      "storefrontName": "%s"
    }
```

```
]], storefrontName)

payload = {
    query = query,
    variables = JSON.decode(variables)
}

response = HTTP.post(headlessAdminUrl, payload)
responseData = response.data.data
responseError = response.data.errors
```