Juan Diego Trejo Sandoval

9B Ingenieria en Software

Unidad 2

Arquitectura Orientada a Servicios

Fued Alejandro Majul Ramirez

Lunes 18 de agosto de 2025

Contenido

Objetivo

Objetivo general

Implementar un chatbot de WhatsApp para un restaurante que permita a los usuarios explorar el menú por categorías, añadir platillos al pedido, confirmar/cancelar, y registrar el pedido en la base de datos, cumpliendo los limites de UX de WhatsApp (Listas y botones) y respondiendo siempre dentro de los tiempos del webhook.

Objetivos específicos

- Mostrar categorías y platillos mediante listas interactivas de WhatsApp, con paginación acorde al limite de 10 filas.
- Gestionar una sesion conversional por usuario con un flujo de estados claro.
- Validar entradas (ej. cantidad) y manejar errores/entradas no validas con mensajes guiados.
- Crear/actualizar registros en BD: Usuarios, pedidos y pedidoItems.
- Enviar confirmación de pedido con total calculado.

Fuera del alcance (Versión base)

- Pagos en WhatsApp, delivery tracking y cálculos de envio.
- Autenticación avanzada; se identifica al usuario por su telefono.
- Persistencia de sesion en cache externo.

Flujo conversacional base

El recorrido del usuario se compone de una secuencia de estados controlador por el backend.

WELCOME: cuando llega el primer mensaje del usuario, el bot le da la bienvenida y le muestra un botón con la opción “Ver Menú”. No se realizan consultas a BD en este punto; solo se invita a iniciar.

MAIN_MENU: si el usuario pulsa el botón o escribe algo que contenga “ve menú”, el sistema consulta las categorías (Menu.findAll) y las presenta como una lista interactiva paginada, respetando los límites de WhatsApp (Max. 10 filas por lista). Desde aquí se pasa a la sección de categoría.

SELECT_CATEGORY: el usuario puede: “ver más” para paginar o elegir una categoría ya sea tocando el item de la lista o escribiendo un numero/nombre valido. Si la entrada no coincide con ninguna categoría y no es una respuesta de lista, el bot envia un mensaje de ayuda y permanece en este estado. Al seleccionar correctamente, se guarda categoryId, se cargan los platillos de esa categoría con paginación y se avanza.

SELECT_DISH: La dinámica es análoga a categorías: el usuario puede pedir mas resultados o seleccionar un platillo por respuestas de lista, numero o nombre. Si la entrada es invalida y no proviene de la lista, se responde con una guía y se mantiene el estado. Cuando la elección es válida. Se guarda dishId y se solicita la cantidad.

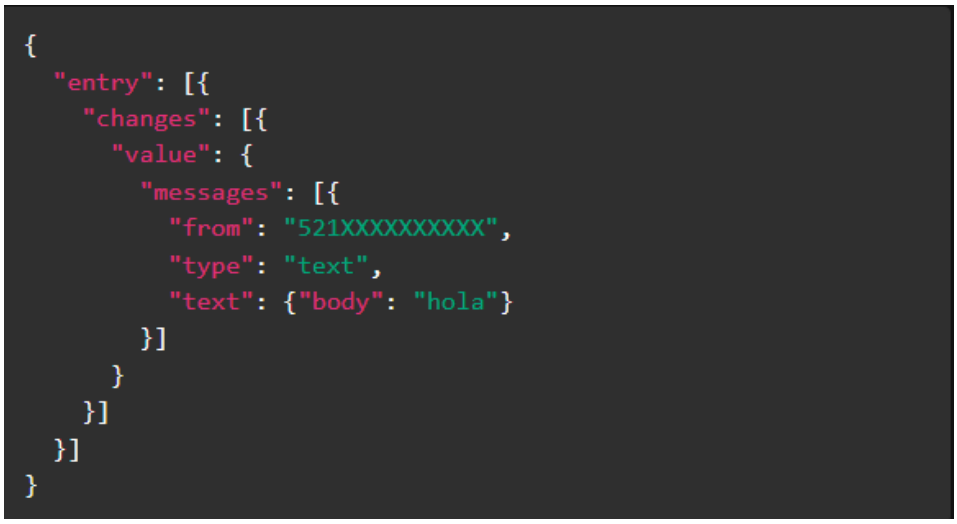
ASK_QUANTITY: El bot espera un numero mayor a 1. Si el valor no es válido, se pide corregirlo. Con un numero valido, el platillo se añade a los ítems de la sesion y se informa: “Agregado….¿Agregar otro?(si/no)”.

ADD_MORE: Si la respuesta del usuario empieza con “s” (si), se regresa al listado de categorías para seguir agregando. En caso contrario, el bot arma un resumen del pedido (líneas por platillo y total) y preguntando por la confirmación final.

CONFIRM: si el usuario confirma (respuesta inicial con “s”), el sistema asegura la existencia del Usuario, crea el pedido con el total y persiste cada PedidoItem. Luego envia el numero de pedido y el total. Si el usuario no confirma, el bot cancela el flujo y avisa. En ambos casos, la sesion se limpia.

Estructura de la API Y DB

API HTTP

- GET /webhook
  - Uso: verificación del webhook.
  - Query params: hub.mode, hub.verify_token, hub.challenge.
  - Respuesta: 200 con hub.challenge si verify_token coincide; 403 en caso contrario.
- POST/ webhook
  - Uso: recepción de evento de WhatsApp
  - 

  - Salida: siempre responder 200 rápidamente al webhook. El bot envia mensajes a través de los helpers y la API de WhatsApp.
  - Reglas de negocio:
    - MAIN_MENU: mostrar categorías paginadas.
    - SELECT_CATEGORY: detectar cat_more, cat_{id}, indice o nombre.
    - SELECT_DISH: detectar dish_more, dish_{id}, indice o nombre.
    - ASK_QUANTITY: Validar número mayor a 1.
    - CONFIRM: persistir pedido y limpiar sesion.

Modelo de datos

- Menú
  - Menu.ts M

src > models > Menu.ts > ...

```ts
import { Column, DataType, HasMany, Model, Table } from "sequelize-typescript";
import Platillos from "./Platillos";

@Table({
    tableName:'Menu'
})

class Menu extends Model{
    @Column({
        type:DataType.STRING(100)
    })
    declare nombre:string

    @HasMany(()=>Platillos)
    declare platillos:Platillos[]
}

export default Menu
```

- Platillos
  - Platillos.ts

src > models > Platillos.ts > Platillos > platillo

```ts
import { BelongsTo, Column, DataType, ForeignKey, Model, Table } from "sequelize-typescript";
import Menu from "./Menu";

@Table({
    tableName:'Platillos'
})

class Platillos extends Model{
    @Column({
        type:DataType.STRING(100)
    })
    declare platillo:string
     @Column({
        type:DataType.INTEGER
    })
    declare precio:number

    @ForeignKey(()=>Menu)
    @Column({type:DataType.INTEGER})
    declare menuId:number

    @BelongsTo(()=>Menu)
    declare menu:Menu


}

export default Platillos
```

- Usuarios
  - Usuarios.ts

src > models > Usuarios.ts > Usuario

```ts
import { AllowNull, Column, DataType, HasMany, Model, Table } from "sequelize-typescript";
import Pedido from "./Pedido";

@Table({tableName:'Usuarios'})
export default class Usuario extends Model{
    @Column({
        type:DataType.STRING(20),
        allowNull:false,
        unique:true
    })
    declare telefono:string

    @HasMany(()=>Pedido)
    declare pedidos: Pedido[]
}
```

- Pedidos
  - Pedido.ts

src > models > Pedido.ts > Pedido

```ts
import { AllowNull, BelongsTo, Column, DataType, ForeignKey, HasMany, Model, Table } from "sequelize-typescript";
import PedidoItem from "./PedidoItem";
import Usuario from "./Usuarios";

@Table({tableName:'Pedidos'})
export default class Pedido extends Model{
    @Column({
        type:DataType.DATE,
        defaultValue:DataType.NOW
    })
    declare fecha:Date
    @Column({
        type:DataType.FLOAT,
        allowNull:false
    })
    declare total:number

    @ForeignKey(()=>Usuario)
    @Column({type:DataType.INTEGER,allowNull:false})
    declare usuarioId:number

    @BelongsTo(()=>Usuario)
    declare usuario : Usuario


    @HasMany(()=>PedidoItem)
    declare items:PedidoItem[]
}
```

- PedidosItems

PedidoItem.ts

src > models > PedidoItem.ts > ...

```ts
import { AllowNull, BelongsTo, Column, DataType, ForeignKey, Model, Table } from "sequelize-typescript";
import Pedido from "./Pedido";
import Platillos from "./Platillos";

@Table({tableName:'PedidoItems'})
export default class PedidoItem extends Model{

    @ForeignKey(()=>Pedido)
    @Column({type:DataType.INTEGER,allowNull:false})
    declare pedidoId:number

    @ForeignKey(()=>Platillos)
    @Column({type:DataType.INTEGER,allowNull:false})
    declare platilloId:number

    @Column({type:DataType.INTEGER,allowNull:false})
    declare cantidad:number

    @BelongsTo(()=>Pedido)
    declare pedido:Pedido
    @BelongsTo(()=>Platillos)
    declare platillo:Platillos
}
```