Documentación de API de Gestión de Restaurante

Daniel Calvar Cruz

31 de octubre de 2025

Índice

1. Introducción

Volver

Esta API proporciona un sistema completo para la gestión de un restaurante, incluyendo el manejo de productos del menú y pedidos de mesas.

La API está construida con Express.js y utiliza MongoDB como base de datos.

2. Estructura

2.1. Rutas Principales

La API está organizada en tres módulos principales:

- **POST**: Operaciones de creación
- **GET**: Operaciones de lectura
- **DELETE**: Operaciones de eliminación
- **PATCH**: Operaciones de actualización

2.2. Modelos de Datos

2.2.1. Modelo Producto

```
const ProductoSchema = new mongoose.Schema({
  id: { type: Number },
  nombre: { type: String },
  precio: { type: Number },
  descripcion: { type: String }
});
```

2.2.2. Modelo Mesa

```
const Mesa1Schema = new mongoose.Schema({
  pedidos: { type: Array }
});

"pedidos": array de objetos Producto.
```

2.2.3. Modelo ListaMesa

```
  const ListaMesaSchema = new mongoose.Schema({
  idMesa: {
            type: String
          },
  ocupada: {
            type: Boolean,
```

```
            default: false
          }
});
```

3. Base de Datos

Volver

MongoDB (Atlas).

Estructura

- listamesas: contiene los estados de ocupación de las mesas, un documento por mesa.
- mesaX: una colección para cada mesa, contiene todos los pedidos de esa mesa, un documento por cada ronda de pedidos.
- productos: datos de los productos de los menús, un documento por producto.

4. Endpoints

Volver

4.1. Endpoints POST

4.1.1. POST /post/createobjetomenu

Descripción: Crea un nuevo producto en el menú.

Body:

```
{
  "nombre": "string",
  "precio": "number",
  "descripcion": "string"
}
```

Respuestas:

- **200**: Documento creado exitosamente
- **400**: Error en los datos proporcionados
- **500**: Error interno del servidor

4.1.2. POST /post/mandarpedidoamesa

Descripción: Envía un pedido a una mesa específica.

Body:

```
{
  "mesa": "string",
  "pedidos": [
    {
      "id": "number",
      "nombre": "string",
      "precio": "number",
      "descripcion": "string"
    }
  ]
}
"mesa": Mesa1, Mesa2, etc.
"pedidos": array de productos.
```

Respuestas:

- **200**: Pedidos creados exitosamente
- **400**: Error en los datos del pedido
- **500**: Error interno del servidor

4.2. Endpoints GET

Volver

4.2.1. GET /get/readallmenus

Descripción: Obtiene todos los productos del menú.

Parámetros: Ninguno

Respuestas:

- **200**: Información encontrada con datos
- **404**: Documentos no encontrados
- **500**: Error interno del servidor

4.2.2. GET /get/readmesa

Descripción: Obtiene los pedidos de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")

Respuestas:

- **200**: Información encontrada
- **400**: Modelo no encontrado
- **404**: No hay datos en la colección
- **500**: Error en proceso de búsqueda

4.2.3. GET /get/readestadomesa

Descripción: Obtiene el estado de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")

Respuestas:

- **200**: Información encontrada
- **400**: Modelo no encontrado
- **404**: No hay datos en la colección
- **500**: Error en proceso de búsqueda

4.3. Endpoints DELETE

Volver

4.3.1. DELETE /delete/deletemesa

Descripción: Elimina todos los pedidos de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")

Respuestas:

- **200**: Información borrada exitosamente
- **400**: Modelo no encontrado
- **500**: Error en proceso de borrado

4.3.2. DELETE /delete/deletepedido

Descripción: Elimina un pedido de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")
- `idDocu`: Identificador del documento mongo

Respuestas:

- **200**: Información borrada exitosamente
- **400**: Modelo no encontrado
- **500**: Error en proceso de borrado

4.4. Endpoints PATCH

Volver

4.4.1. PATCH /patch/cambiarestadomesa

Descripción: Cambia el estado de ocupación de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")
- `ocupada`: booleano que identifica si la mesa está ocupada o no

Respuestas:

- **200**: Información actualizada exitosamente
- **400**: Modelo no encontrado
- **500**: Error en proceso de borrado

4.4.2. PATCH /patch/cambiarestadopedido

Descripción: Cambia el estado de servicio (ha sido servido o no a la mesa) de un pedido de una mesa específica.

Parámetros:

- `mesaId`: Identificador de la mesa (ej: "Mesa1")
- `iddocu`: Identificador del documento mongo
- `hasidoservido`: booleano que identifica si ese pedido ha sido servido a mesa o no

Respuestas:

- **200**: Información actualizada exitosamente
- **400**: Modelo no encontrado
- **500**: Error en proceso de borrado

5. Flujos de Trabajo

Volver

5.1. Agregar Producto al Menú

1. El sistema genera automáticamente un ID incremental
2. Valida que los datos del producto sean correctos
3. Guarda el producto en la colección Producto

5.2. Realizar un Pedido

1. Valida que los productos existan en la base de datos
2. Verifica la estructura de los datos del pedido
3. Guarda el pedido en la mesa correspondiente

5.3. Enviar datos al front

1. Valida que los datos existan en la base de datos
2. Verifica la estructura de los datos
3. Envia los datos

5.4. Cambio de datos en BBDD

1. Valida que los datos se hayan enviado
2. Verifica la estructura de los datos
3. Modifica los datos

6. Manejo de Errores

Volver

La API utiliza un sistema consistente de respuestas:

```
{
  "type": "success|failure",
  "message": "string descriptivo",
  "data": "object (opcional)"
}
```

7. Ejemplos de Uso

7.1. Ejemplo: Crear Producto

```
POST /post/createObjetoMenu
{
  "nombre": "Hamburguesa Clásica",
  "precio": 12.99,
  "descripcion": "Hamburguesa con queso, lechuga y tomate"
}
```

7.2. Ejemplo: Hacer Pedido

```
POST /post/mandarPedidoAMesa
{
  "mesa": "Mesa1",
  "pedidos": [
    {
      "id": 1,
      "nombre": "Hamburguesa Clásica",
      "precio": 12.99,
      "descripcion": "Hamburguesa con queso, lechuga y tomate"
    }
  ]
}
```

8. Consideraciones Técnicas

Volver

- Los IDs de productos se generan automáticamente de forma incremental
- Las mesas deben estar predefinidas en los modelos (Mesa1, Mesa2, etc.)
- Todos los precios deben ser números válidos
- La API valida la existencia de productos antes de crear pedidos

9. Códigos de Estado HTTP

- **200 OK**: Operación exitosa
- **400 Bad Request**: Error en los datos enviados
- **404 Not Found**: Recurso no encontrado
- **500 Internal Server Error**: Error del servidor

10. Stack tecnológico

- **Versión Node.js:** 22.14
- **Base de datos:** MongoDB Atlas
- **Dependencias NPM:**
  - **cors:** 2.8.5
  - **dotenv:** 16.4.7
  - **express:** 4.21.1
  - **mongodb:** 6.12.0
  - **mongoose:** 8.13.2
  - **nodemon:** 3.1.10

11. Variables de entorno (.env)

- `MONGO_URI`: URI de conexión a MongoDB Atlas o local.
- `PORT`: Puerto en el que se ejecutará el servidor.
- `DDBB_NAME`: Nombre de la base de datos en Mongo Atlas.