

# **EXPLICACION DE CODIGO EVALUACION PROGRAMACION WEB**

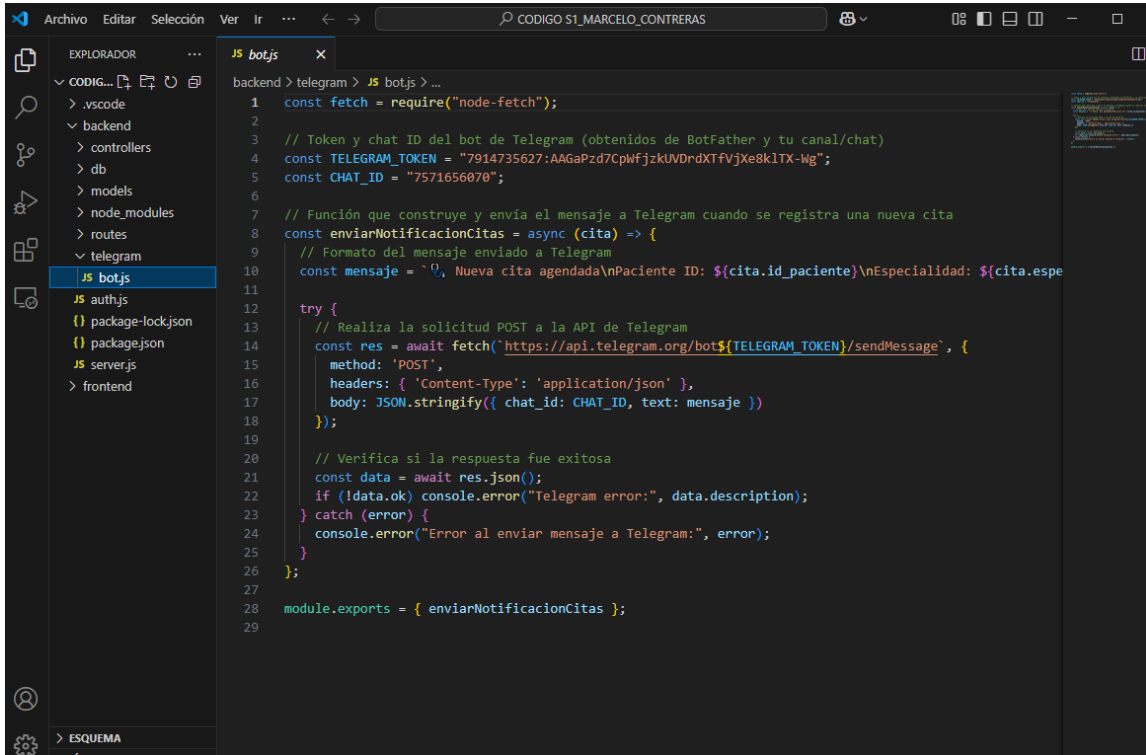


## **Informe Técnico Detallado**

**MARCELO ALEJANDRO CONTRERAS LAFLOR**

# BOT.JS

## 1. Archivo bot.js



```
1 const fetch = require("node-fetch");
2
3 // Token y chat ID del bot de Telegram (obtenidos de BotFather y tu canal/chat)
4 const TELEGRAM_TOKEN = "7914735627:AAGaPzd7CpWfjzkUVDrdXTfVjXe8klTX-Wg";
5 const CHAT_ID = "7571656070";
6
7 // Función que construye y envía el mensaje a Telegram cuando se registra una nueva cita
8 const enviarNotificacionCitas = async (cita) => {
9   // Formato del mensaje enviado a Telegram
10  const mensaje = `\\, Nueva cita agendada\\nPaciente ID: ${cita.id_paciente}\\nEspecialidad: ${cita.espe
11
12  try {
13    // Realiza la solicitud POST a la API de Telegram
14    const res = await fetch(`https://api.telegram.org/bot${TELEGRAM_TOKEN}/sendMessage`, {
15      method: 'POST',
16      headers: { 'Content-Type': 'application/json' },
17      body: JSON.stringify({ chat_id: CHAT_ID, text: mensaje })
18    });
19
20    // Verifica si la respuesta fue exitosa
21    const data = await res.json();
22    if (!data.ok) console.error("Telegram error:", data.description);
23  } catch (error) {
24    console.error("Error al enviar mensaje a Telegram:", error);
25  }
26 };
27
28 module.exports = { enviarNotificacionCitas };
29
```

**Nombre del módulo:** enviarNotificacionCitas

### Descripción:

Este módulo encapsula la lógica de integración con Telegram, permitiendo al sistema enviar automáticamente una notificación cada vez que se registra una nueva cita médica. Se realiza mediante una petición HTTP POST a la API de Telegram.

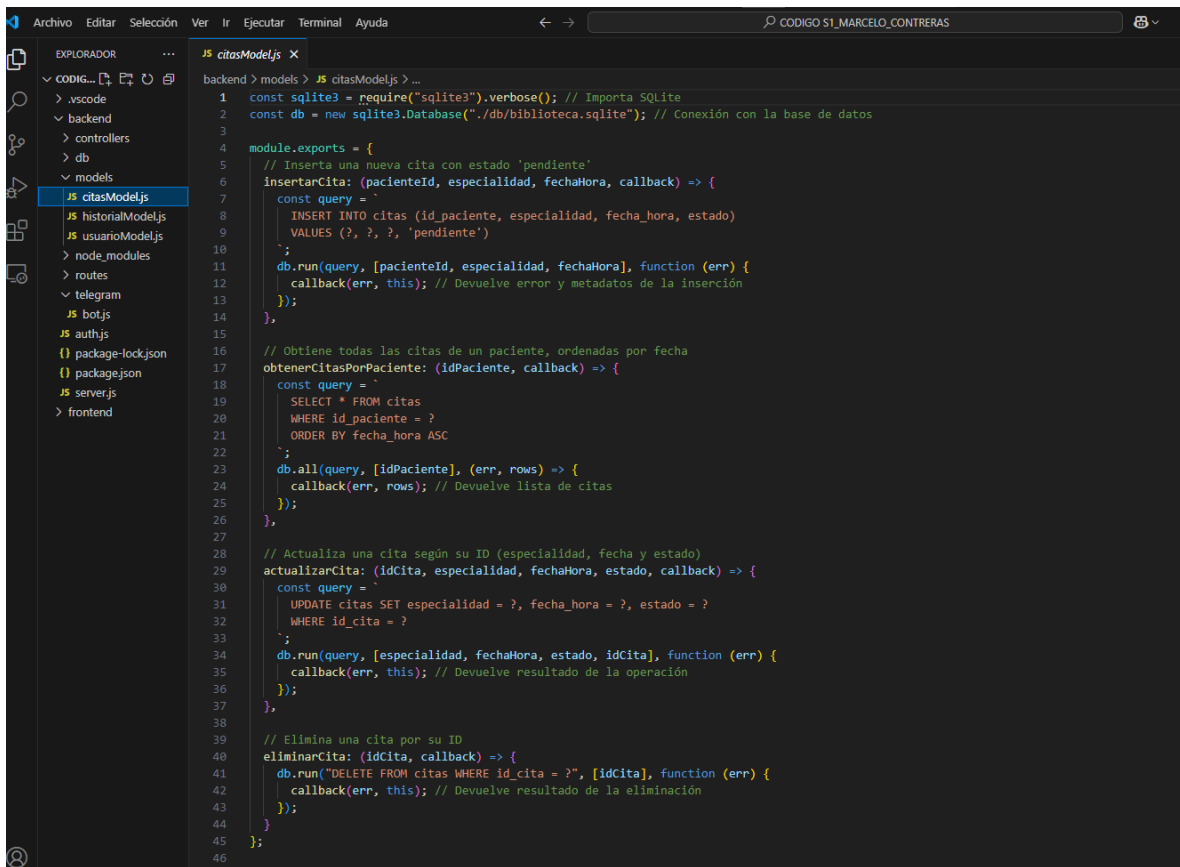
### Componentes principales:

- **fetch:** Módulo de Node.js para realizar solicitudes HTTP.
- **TELEGRAM\_TOKEN y CHAT\_ID:**  
Valores que identifican al Bot y el canal/chat donde se debe enviar el mensaje.
- **enviarNotificacionCitas(cita):**  
Función que construye el mensaje con los datos de la cita (ID del paciente, especialidad, fecha y estado), y lo envía al chat de Telegram configurado.

### Flujo de ejecución:

1. Construcción del mensaje con formato predefinido.
2. Envío del mensaje mediante fetch a  
<https://api.telegram.org/bot<TOKEN>/sendMessage>.
3. Manejo de posibles errores: errores de red o respuesta fallida de la API.

## citasModels.JS



```
1 const sqlite3 = require("sqlite3").verbose(); // Importa SQLite
2 const db = new sqlite3.Database("./db/biblioteca.sqlite"); // Conexión con la base de datos
3
4 module.exports = {
5   // Inserta una nueva cita con estado 'pendiente'
6   insertarCita: (pacienteId, especialidad, fechaHora, callback) => {
7     const query = `
8       INSERT INTO citas (id_paciente, especialidad, fecha_hora, estado)
9       VALUES (?, ?, ?, 'pendiente')
10    `;
11    db.run(query, [pacienteId, especialidad, fechaHora], function (err) {
12      callback(err, this); // Devuelve error y metadatos de la inserción
13    });
14  },
15
16   // Obtiene todas las citas de un paciente, ordenadas por fecha
17   obtenerCitasPorPaciente: (idPaciente, callback) => {
18     const query = `
19       SELECT * FROM citas
20       WHERE id_paciente = ?
21       ORDER BY fecha_hora ASC
22    `;
23    db.all(query, [idPaciente], (err, rows) => {
24      callback(err, rows); // Devuelve lista de citas
25    });
26  },
27
28   // Actualiza una cita según su ID (especialidad, fecha y estado)
29   actualizarCita: (idCita, especialidad, fechaHora, estado, callback) => {
30     const query = `
31       UPDATE citas SET especialidad = ?, fecha_hora = ?, estado = ?
32       WHERE id_cita = ?
33    `;
34    db.run(query, [especialidad, fechaHora, estado, idCita], function (err) {
35      callback(err, this); // Devuelve resultado de la operación
36    });
37  },
38
39   // Elimina una cita por su ID
40   eliminarCita: (idCita, callback) => {
41     db.run("DELETE FROM citas WHERE id_cita = ?", [idCita], function (err) {
42       callback(err, this); // Devuelve resultado de la eliminación
43     });
44   }
45 };
46
```

## 2. Archivo citasModel.js

**Nombre del módulo:** Modelo de Acceso a la Base de Datos para Citas

### Descripción:

Este módulo contiene todas las funciones necesarias para interactuar con la tabla citas de la base de datos biblioteca. SQLite. Representa la **capa de persistencia** del sistema, encargada de acceder, insertar, actualizar y eliminar datos.

### Funciones incluidas:

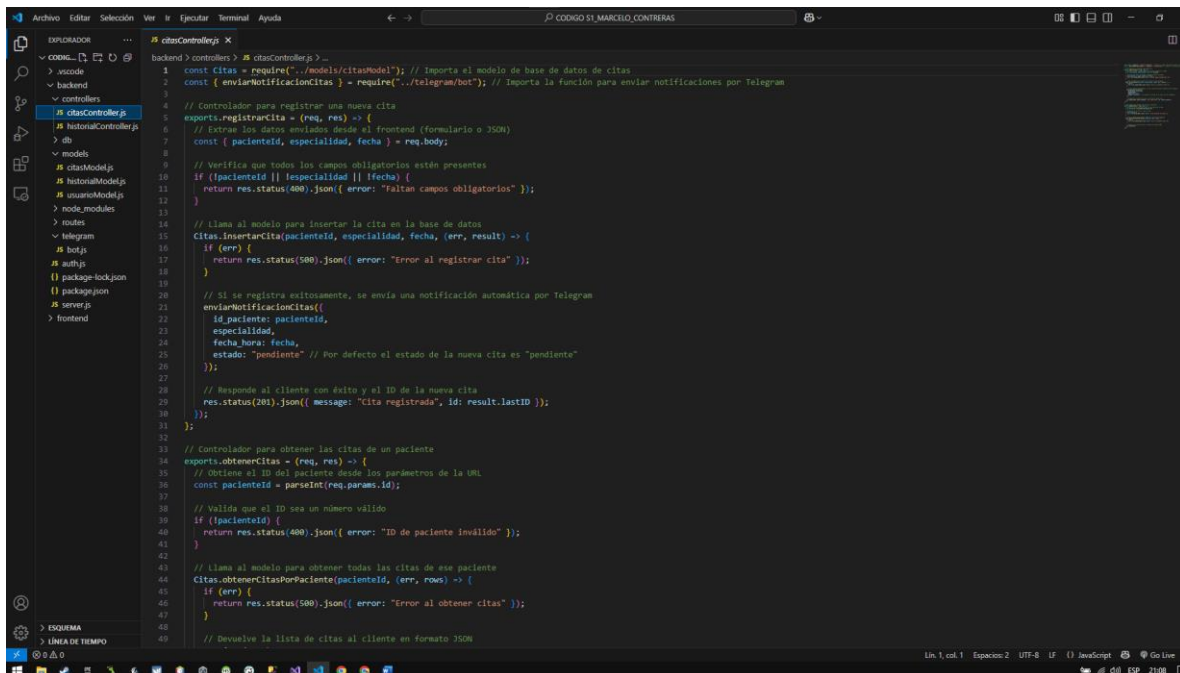
1. **insertarCita(pacienteId, especialidad, fechaHora, callback)**  
Inserta una nueva cita médica en la base de datos con estado "pendiente".
2. **obtenerCitasPorPaciente(idPaciente, callback)**  
Devuelve todas las citas asociadas a un paciente, ordenadas cronológicamente por fecha y hora.

3. **actualizarCita(idCita, especialidad, fechaHora, estado, callback)**  
Permite modificar una cita existente en función de su ID.
4. **eliminarCita(idCita, callback)**  
Elimina permanentemente una cita médica específica de la base de datos.

**Estructura de la tabla citas:**

Columna	Tipo de dato	Descripción
id_cita	INTEGER	Identificador único de la cita
id_paciente	INTEGER	Referencia al paciente
especialidad	TEXT	Especialidad médica (Ej: Cardiología)
fecha_hora	TEXT (ISO)	Fecha y hora programadas de la cita
estado	TEXT	Estado de la cita (pendiente, atendida, etc)

# citasController.js



```
backend > controllers > citasController.js
1 const citas = require("../models/citasModel"); // Importa el modelo de base de datos de citas
2 const { enviarNotificacionCitas } = require("../telegram/bot"); // Importa la función para enviar notificaciones por Telegram
3
4 // Controlador para registrar una nueva cita
5 exports.registrarCita = (req, res) => {
6   // Extrae los datos enviados desde el frontend (formulario o JSON)
7   const { pacienteId, especialidad, fecha } = req.body;
8
9   // Verifica que todos los campos obligatorios estén presentes
10  if (!pacienteId || !especialidad || !fecha) {
11    return res.status(400).json({ error: "Faltan campos obligatorios" });
12  }
13
14  // Llama al modelo para insertar la cita en la base de datos
15  citas.insertarCita(pacienteId, especialidad, fecha, (err, result) => {
16    if (err) {
17      return res.status(500).json({ error: "Error al registrar cita" });
18    }
19
20    // Si se registra exitosamente, se envía una notificación automática por Telegram
21    enviarNotificacionCitas({
22      id_paciente: pacienteId,
23      especialidad,
24      fecha_hora: fecha,
25      estado: "pendiente" // Por defecto el estado de la nueva cita es "pendiente"
26    });
27
28    // Responde al cliente con éxito y el ID de la nueva cita
29    res.status(201).json({ message: "Cita registrada", id: result.lastID });
30  });
31 }
32
33 // Controlador para obtener las citas de un paciente
34 exports.obtenerCitas = (req, res) => {
35   // Obtiene el ID del paciente desde los parámetros de la URL
36   const pacienteId = parseInt(req.params.id);
37
38   // Valida que el ID sea un número válido
39   if (!pacienteId) {
40     return res.status(400).json({ error: "ID de paciente inválido" });
41   }
42
43   // Llama al modelo para obtener todas las citas de ese paciente
44   citas.obtenerCitasPorPaciente(pacienteId, (err, rows) => {
45     if (err) {
46       return res.status(500).json({ error: "Error al obtener citas" });
47     }
48
49     // Devuelve la lista de citas al cliente en formato JSON
50   });
51 }
```

## 📁 3. Archivo citasController.js

**Nombre del módulo: Controlador de Citas**

### Descripción:

Este archivo actúa como **punto entre el cliente y la base de datos**. Contiene las funciones que procesan las solicitudes HTTP entrantes relacionadas con la gestión de citas. También es el punto de conexión con el bot de Telegram.

### Funciones principales:

#### 1. registrarCita(req, res)

- Extrae los campos pacienteId, especialidad, y fecha desde req.body.
- Verifica que todos los campos estén presentes. Si falta alguno, retorna un error 400.
- Inserta la nueva cita usando insertarCita.
- Llama a enviarNotificacionCitas para notificar por Telegram.

- Devuelve una respuesta 201 con el ID de la cita creada.

## 2. **obtenerCitas(req, res)**

- Obtiene el ID del paciente desde req.params.id.
- Llama al modelo para recuperar todas sus citas.
- Devuelve el listado en formato JSON o un error si algo falla.

## **Integración:**

Este archivo importa:

- citasModel.js para manejar la base de datos.
- bot.js para las notificaciones automáticas.

## **Ejemplo de flujo:**

1. Cliente realiza POST a /citas con datos de la cita.
2. registrarCita() valida y guarda la cita.
3. Se dispara automáticamente una notificación a Telegram.

## Telegram

### Ingreso de datos:

### Clínica Médica Vitalis

[Inicio](#)[Agendar Cita](#)[Historial](#)[Atencion](#)[Cerrar Sesión](#)

#### Agendar Cita Médica

Otorrino

Fecha:

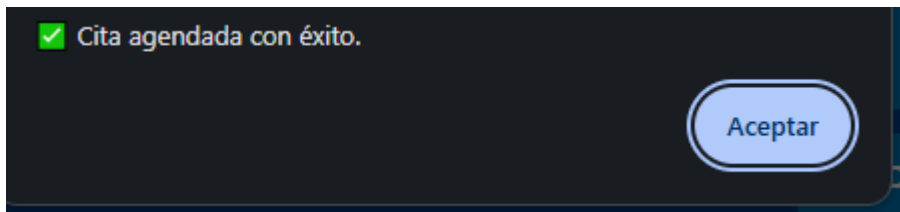
11 - 05 - 2025

Hora:

16:45

Agendar

### Mensaje en Telegram:







marcedrkk\_bot  
bot

Paciente ID: 200  
Especialidad: Otorrino  
Fecha: 2025-05-05 16:15  
Estado: pendiente 20:36

Nueva cita agendada  
Paciente ID: 111  
Especialidad: Otorrino  
Fecha: 2025-05-14 08:00  
Estado: pendiente 20:42

Nueva cita agendada  
Paciente ID: 200  
Especialidad: Otorrino  
Fecha: 2025-05-04 12:45  
Estado: pendiente 20:44

Nueva cita agendada  
Paciente ID: 226  
Especialidad: Otorrino  
Fecha: 2025-05-02 12:30  
Estado: pendiente 20:45

Nueva cita agendada  
Paciente ID: 227  
Especialidad: Otorrino  
Fecha: 2025-05-24 11:30  
Estado: pendiente 20:47

Nueva cita agendada  
Paciente ID: 288  
Especialidad: Otorrino  
Fecha: 2025-05-01 16:45  
Estado: pendiente 20:48

Nueva cita agendada  
Paciente ID: 288  
Especialidad: Otorrino  
Fecha: 2025-05-11 16:45  
Estado: pendiente 21:12

