📘 Documentación API – Consulta Externa

# 🔗 Base URL

http://<IP\_LOCAL>:5000/api/external-consultation

# 🔹 Endpoints REST

## 1. 🔍 Obtener módulos disponibles

GET /available-modules

Descripción: Lista los módulos disponibles para atención.

Respuesta:

[{"name": "Módulo 1", "currentAppointments": [ /\* lista de citas \*/ ]}]

## 2. 🚶‍♂️ Publicar llegada de paciente

POST /arrival

Descripción: Informa la llegada del paciente y notifica por WebSocket.

Body:

{  
 "id": 1,  
 "speciality": "GENERAL",  
 "officeName": "Consultorio 5",  
 "arrivalTime": "08:30:00",  
 "attentionTime": null,  
 "*endAttentionTime*": null,  
 "patientName": "Juan Pérez",  
 "patientId": "123456789",  
 "turn": "A15",  
 "attentionModule": "Módulo 1",  
 "state": "ARRIVAL"  
}

WebSocket emitido: /topic/external-consultation/arrival-confirmation-GENERAL

## 3. 🧾 Anunciar a facturación

POST /anounce-to-billing

Descripción: Envía al paciente a facturación.

Body: {"id": 1}

WebSocket emitido: /topic/external-consultation/anounce-to-billing

## 4. ⏱ Enviar a sala de espera del consultorio

POST /send-to-waiting-consult

Descripción: Notifica que el paciente está esperando en el consultorio.

Body: {"id": 1, "officeName": "Consultorio 3", "speciality": "GENERAL"}

WebSocket emitido:

- /topic/external-consultation/waiting-consult

- /topic/external-consultation/arrival-confirmation-GENERAL

## 5. 📞 Llamar a consulta

POST /call-to-consult

Descripción: (Pendiente de implementación)

## 6. ❌ Eliminar cita

DELETE /delete-appoinment/{appoinmentId}

Descripción: Elimina una cita por ID.

# 🔹 WebSocket

URL WebSocket: ws://<IP\_LOCAL>:5000/websocket-endpoint

Canales disponibles:

- /topic/external-consultation/arrival-confirmation-{ESPECIALIDAD}

- /topic/external-consultation/anounce-to-billing

- /topic/external-consultation/waiting-consult

## 🧪 Ejemplo de conexión WebSocket (JS)

const socket = new SockJS('http://<IP\_LOCAL>:5000/websocket-endpoint');  
const stompClient = Stomp.over(socket);  
  
stompClient.connect({}, () => {  
 stompClient.subscribe('/topic/external-consultation/arrival-confirmation-GENERAL', message => {  
 console.log('Actualización citas:', JSON.parse(message.body));  
 });  
  
 stompClient.send('/app/external-consultation/arrival-confirmation', {}, "Paciente llegó");  
});

import React, { useEffect, useState, useCallback, useRef } from "react";

import useSocket from "./useSocket"; // Hook personalizado para conectar a un solo canal

// Componente interno que se encarga de conectar a una especialidad

function SingleSocket({ speciality, onMessage, registerSendMessage }) {

const { data, sendMessage } = useSocket(`ExternalConsultationArrival\_${speciality}`);

useEffect(() => {

if (data) {

onMessage(data);

}

}, [data, onMessage]);

useEffect(() => {

if (sendMessage) {

registerSendMessage(speciality, sendMessage);

}

}, [sendMessage, registerSendMessage, speciality]);

return null;

}

// Componente que engloba todos los sockets activos

function MultiSocketComponents({ specialities, onMessage, registerSendMessage }) {

return (

<>

{specialities.map((spec) => (

<SingleSocket

key={spec}

speciality={spec}

onMessage={onMessage}

registerSendMessage={registerSendMessage}

/>

))}

</>

);

}

// Hook principal para múltiples especialidades

export default function useMultiSocket(specialities = [], onMessages) {

const [allMessages, setAllMessages] = useState({});

const sendMessageMap = useRef({}); // Map de funciones sendMessage

// Registrar una función sendMessage

const registerSendMessage = useCallback((speciality, fn) => {

sendMessageMap.current[speciality] = fn;

}, []);

// Enviar mensaje a una especialidad específica

const sendMessage = useCallback((speciality, message) => {

const fn = sendMessageMap.current[speciality];

if (fn) {

fn(message);

} else {

console.warn(`No hay función de envío para la especialidad: ${speciality}`);

}

}, []);

// La función onMessages pasada la usamos directamente para manejar mensajes

const handleMessage = onMessages;

return {

allMessages,

sendMessage,

MultiSocketComponents: (

<MultiSocketComponents

specialities={specialities}

onMessage={handleMessage}

registerSendMessage={registerSendMessage}

/>

),

};

}

CODIGO FACTURACION.JS  
  
import React, { useEffect, useState } from "react";

import { FaPhoneAlt, FaTrashAlt } from "react-icons/fa";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

import useMultiSocket from "./useMultiSocket";

import "./Facturacion.css";

import { API\_BASE\_URL } from "./Recepcion";

import useSocket, { DestSocket } from "./useSocket";

const MODULOS = [

  { moduleName: "Módulo 1", availableSpecialities: ["CIRUGIA\_PROGRAMADA"] },

  { moduleName: "Módulo 2", availableSpecialities: ["SOAT", "ARL"] },

  { moduleName: "Módulo 3", availableSpecialities: ["SOAT", "GENERAL"] },

  { moduleName: "Módulo 4", availableSpecialities: ["GENERAL"] },

  { moduleName: "Módulo 5", availableSpecialities: ["MEDICINA\_PREPAGADA", "PRUEBAS\_MAGISTRALES", "INFUSIONES", "RECAUDOS", "QR"] },

  { moduleName: "Módulo 6", availableSpecialities: ["GENERAL", "CURACIONES"] },

  { moduleName: "Módulo 7", availableSpecialities: ["LABORATORIOS", "ATENCION\_PREFERENCIAL"] },

  { moduleName: "Módulo 8", availableSpecialities: ["ONCOLOGIA", "ATENCION\_PREFERENCIAL"] },

  { moduleName: "Módulo 9", availableSpecialities: ["ATENCION\_PREFERENCIAL"] },

  { moduleName: "Módulo 10", availableSpecialities: ["CHATBOT", "POSTQUIRURGICO"] },

  { moduleName: "Módulo Virtual", availableSpecialities: ["APOYO"] },

];

function Facturacion() {

  const [moduloSeleccionado, setModuloSeleccionado] = useState(null);

  const [turnos, setTurnos] = useState([]);

  const [filtro, setFiltro] = useState("");

  const [nuevosTurnos, setNuevosTurnos] = useState([]);

   const [opcionesConsultorios, setOpcionesConsultorios] = useState([]);

  const [consultoriosVisibles, setConsultoriosVisibles] = useState({});

  const [table, setTable] = useState()

*// Callback para procesar mensajes entrantes de todos los sockets*

  const onMessages = (allMessages) => {

    console.log("Mensajes recibidos:", allMessages.flat()); *// <-- Aquí ves los mensajes que llegan*

    if(moduloSeleccionado === null || moduloSeleccionado === undefined){

      return[]

    }

*// Object.values(allMessages).flat().forEach((msg) => {*

   allMessages

    .flat()

    .filter(a => a.attentionModule === moduloSeleccionado.moduleName)

    .forEach((msg) => {

      if (!msg) return;

      try {

        const parsed = typeof msg === "string" ? JSON.parse(msg) : msg;

        if (parsed && parsed.id) {

          setTurnos((prevTurnos) => {

            if (!prevTurnos.some((t) => t.id === parsed.id)) {

              toast.info(`Nuevo turno: ${parsed.patientName || parsed.nombre || "Paciente"}`);

              setNuevosTurnos((prevIds) => [...prevIds, parsed.id]);

              return [...prevTurnos, parsed];

            }

            return prevTurnos;

          });

        }

      } catch (err) {

        console.error("Error procesando mensaje:", err);

      }

    });

  };

*// Desestructuramos solo lo que usamos para evitar warnings*

  const { sendMessage, MultiSocketComponents } = useMultiSocket(

    moduloSeleccionado ? moduloSeleccionado.availableSpecialities : [],

    onMessages

  );

    useEffect(() => {

    console.log(turnos)

  }, [turnos]);

*// Función para enviar el mensaje de llamada al paciente por WebSocket X*

  const enviarLlamada = async (appoinment) => {

    if (!moduloSeleccionado) {

      toast.error("Debe seleccionar un módulo primero");

      return;

    }

    console.log("ENVIANDO DATA: ", appoinment)

    try{

    const response = await fetch(`${API\_BASE\_URL}/announce-to-billing`, {

      method: 'POST',

       headers: {

      'Content-Type': 'application/json', *// <- importante*

    },

      body: JSON.stringify(appoinment)

    })

    console.log('RESPUESTA ENVIADA AL SERVIDOR PARA LLAMAR: ', response);

    toast.success(`Llamando a ${appoinment.patientName  || "Paciente"} - Turno: ${appoinment.turn}`);

    }catch(err){

      console.error('[ERROR]', err)

    }

  };

*//FUNCIÓN PARA MANEJAR ELIMINAR*

  const manejarEliminar = async (id) => {

    if (!window.confirm("¿Seguro que quieres eliminar este turno?")) return;

    try {

      const response = await fetch(`${API\_BASE\_URL}/delete-appoinment/modules/${id}`, {

      method: "DELETE",

      });

    if (!response.ok) {

      throw new Error("Error al eliminar la cita");

    }

    setTurnos((prev) => prev.filter((turno) => turno.id !== id));

    setNuevosTurnos((prev) => prev.filter((turnoId) => turnoId !== id));

    toast.success("Turno eliminado correctamente");

  } catch (error) {

    console.error("Error al eliminar turno:", error);

    toast.error("No se pudo eliminar el turno. Intenta nuevamente.");

  }

  };

*//CARGAR CONSULTORIOS DESDE LA API*

  const cargarConsultorios = async () => {

    try {

      const response = await fetch("http://192.168.17.236:5000/api/external-consultation/offices");

      const data = await response.json();

      setOpcionesConsultorios(data);

    } catch (error) {

      console.error("Error al cargar consultorios:", error);

      toast.error("No se pudieron cargar los consultorios");

    }

  };

  const turnosFiltrados = turnos.filter((turno) => {

    const filtroLower = filtro.toLowerCase();

    return (

      (turno.patientName && turno.patientName.toLowerCase().includes(filtroLower)) ||

      (turno.nombre && turno.nombre.toLowerCase().includes(filtroLower)) ||

      (turno.cedula && turno.cedula.toLowerCase().includes(filtroLower)) ||

      (turno.tipo && turno.tipo.toLowerCase().includes(filtroLower))

    );

  });

  if (!moduloSeleccionado) {

    return (

      <div className="facturacion-container" style={{ textAlign: "center" }}>

        <h1>Seleccione su Módulo</h1>

        <ul style={{ listStyle: "none", padding: 0, maxWidth: 400, margin: "30px auto" }}>

          {MODULOS.map((mod) => (

            <li key={mod.moduleName} style={{ marginBottom: 15 }}>

              <button

                className="btn-llamar"

                style={{ width: "100%", fontSize: "1.1rem" }}

                onClick={() => {

                  setModuloSeleccionado(mod);

                  setTurnos([]);

                  setFiltro("");

                  setNuevosTurnos([]);

                }}

              >

                {mod.moduleName}

              </button>

            </li>

          ))}

        </ul>

      </div>

    );

  }

  return (

    <div className="facturacion-container">

      <ToastContainer />

      <div className="facturacion-header">

        <h1>Gestión de turnos - {moduloSeleccionado.moduleName}</h1>

        <button

          onClick={() => {

            setModuloSeleccionado(null);

            setTurnos([]);

            setFiltro("");

            setNuevosTurnos([]);

          }}

          className="btn-eliminar"

          style={{ height: 40, alignSelf: "center" }}

          title="Cambiar módulo"

        >

          Cambiar módulo

        </button>

      </div>

      <input

        type="text"

        placeholder="Filtrar por nombre, cédula o especialidad..."

        value={filtro}

        onChange={(e) => setFiltro(e.target.value)}

        className="select-turno"

      />

       <table className="tabla-turnos">

        <thead>

          <tr>

            <th>Turno</th>

            <th>Nombre Completo</th>

            <th>Número de Documento</th>

            <th>Especialidad</th>

            <th>Hora Confirmación</th>

            <th>Acciones</th>

          </tr>

        </thead>

        <tbody>

          {turnosFiltrados.length === 0 ? (

            <tr>

              <td colSpan="7" className="sin-turnos">

                {turnos.length === 0

                  ? "No hay turnos confirmados aún."

                  : "No hay coincidencias con el filtro."}

              </td>

            </tr>

          ) : (

            turnosFiltrados.filter(p=>p.attentionModule===moduloSeleccionado.moduleName).map((turno) => (

              <tr key={turno.id} className={nuevosTurnos.includes(turno.id) ? "nuevo-turno" : ""}>

                <td>{turno.turn}</td>

                <td>{turno.patientName || turno.nombre}</td>

                <td>{turno.patientId}</td>

                <td>{turno.speciality}</td>

                <td>{turno.arrivalTime.split('.')[0] || "-"}</td>

                <td>

                  <button onClick={() => enviarLlamada(turno)} className="btn-llamar">

                    <FaPhoneAlt /> Llamar

                  </button>

                  <button onClick={() => manejarEliminar(turno.id)} className="btn-eliminar">

                    <FaTrashAlt /> Eliminar

                  </button>

                  <button

                    className="btn-enviar"

                    onClick={() => {

                      cargarConsultorios();

                      setConsultoriosVisibles((prev) => ({

                        ...prev,

                        [turno.id]: !prev[turno.id],

                      }));

                    }}

                  >

                    Enviar a Tamizaje

                  </button>

                  {consultoriosVisibles[turno.id] && (

                    <>

                      <input

                        list={`lista-consultorios-${turno.id}`}

                        placeholder="Escribe o selecciona un consultorio..."

                        onChange={(e) => {

                          const consultorioSeleccionado = e.target.value;

                          const consultorioObj = opcionesConsultorios.find(

                            (c) => c.nombre === consultorioSeleccionado

                          );

                          if (!consultorioObj) {

                            toast.error("Consultorio no válido");

                            return;

                          }

                          const recepcion = consultorioObj.recepcion; *// recepción indicada por backend*

                          toast.info(`Turno ${turno.turno} debe dirigirse a ${recepcion}`);

*// Envía la info por socket para que backend LO registre*

                          moduloSeleccionado.availableSpecialities.forEach((spec) => {

                            sendMessage(spec, {

                              action: "ASIGNAR\_CONSULTORIO",

                              turnoId: turno.id,

                              consultorio: consultorioSeleccionado,

                              recepcion,

                            });

                          });

                          setConsultoriosVisibles((prev) => ({

                            ...prev,

                            [turno.id]: false,

                          }));

                        }}

                        className="input-tamizaje"

                      />

                      <datalist id={`lista-consultorios-${turno.id}`}>

                        {opcionesConsultorios.map((c) => (

                          <option key={c.id} value={c.nombre} />

                        ))}

                      </datalist>

                    </>

                  )}

                </td>

              </tr>

            ))

          )}

        </tbody>

      </table>

      {*/\* Renderizamos los sockets \*/*}

      {MultiSocketComponents}

    </div>

  );

}

export default Facturacion;