Centro de Estudios Lerdo Contemporáneo

**Proyecto:**

Chat bot

**Especialidad:**

Sistemas Computacionales

**Materia:**

Seminario de Actualización Computacional

**Docente:**

Ing. Javier Martínez Benavente

**Alumno:**

Ruben Canales Reyes

**Introducción**

**Chatbot Escolar**

Este proyecto es una herramienta diseñada para facilitar la comunicación entre la escuela y los estudiantes, padres y futuros alumnos. Este chatbot ha sido desarrollado para proporcionar información relevante y actualizada de manera rápida y eficiente.

**Funcionalidades Principales**

El Chatbot Escolar cuenta con varias funcionalidades clave que permiten a los usuarios obtener la información que necesitan sin necesidad de buscar manualmente en la web o contactar directamente con la escuela. Las principales funciones incluyen:

* Solicitud de Información General: El chatbot puede proporcionar respuestas a preguntas frecuentes sobre la escuela, como horarios de atención, ubicaciones de las instalaciones y eventos próximos.
* Consulta de Número de Teléfono: Los usuarios pueden solicitar el número de teléfono de contacto de la escuela para obtener asistencia directa o resolver consultas más específicas.
* Información sobre Carreras de Licenciatura: El chatbot ofrece detalles sobre las diferentes carreras de licenciatura que se imparten en la escuela. Esto incluye información sobre planes de estudio, requisitos de admisión, duración de las carreras, y oportunidades profesionales.

**Beneficios**

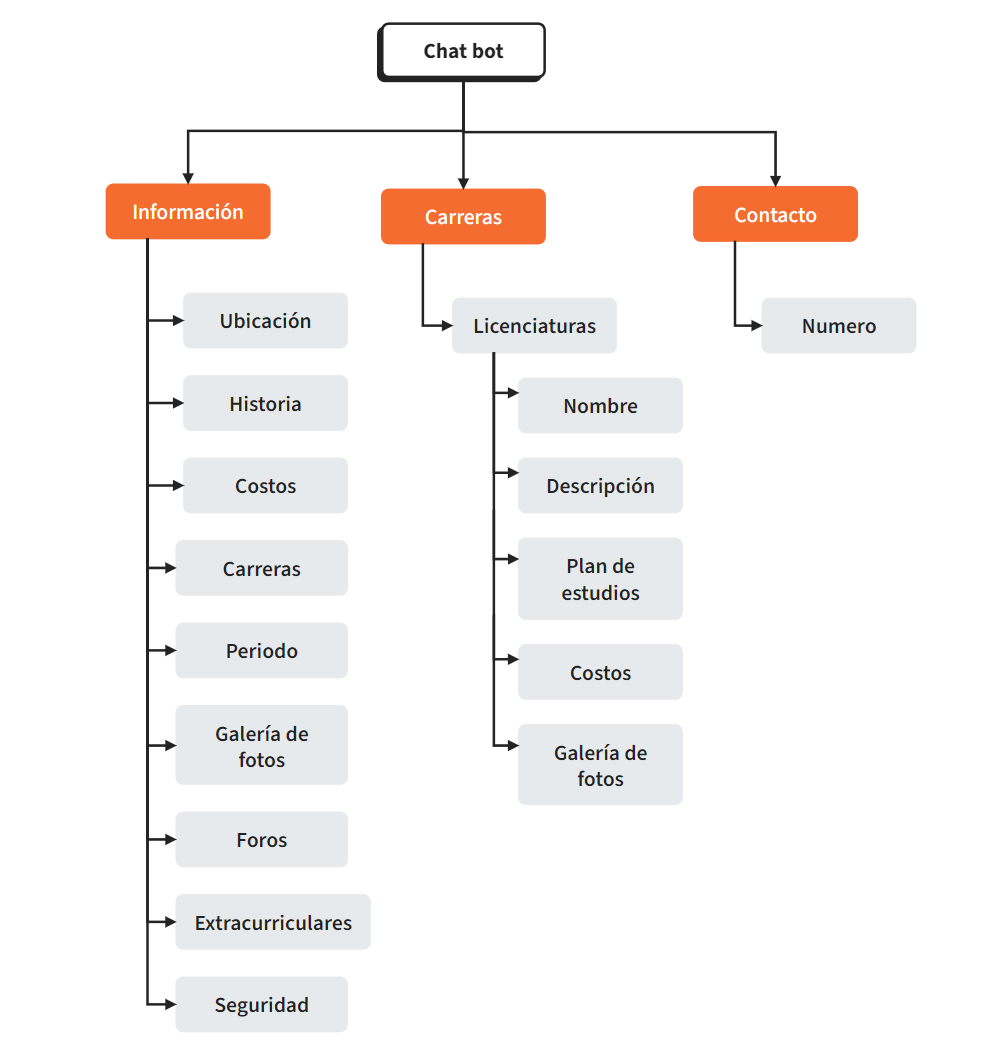
El uso de un chatbot para la comunicación escolar ofrece varios beneficios:

1. Disponibilidad 24/7: Los usuarios pueden acceder a la información en cualquier momento del día, sin depender del horario laboral del personal administrativo.
2. Respuesta Rápida: El chatbot puede proporcionar respuestas inmediatas a las consultas, mejorando la satisfacción del usuario.
3. Reducción de Carga Administrativa: Al automatizar las respuestas a preguntas frecuentes, el personal administrativo puede centrarse en tareas más complejas y personalizadas.
4. Accesibilidad y Conveniencia: Los usuarios pueden interactuar con el chatbot desde cualquier dispositivo con acceso a Internet, facilitando el acceso a la información desde cualquier lugar.

**Dependencias usadas**

* Node
* Body Parser
* Cors
* Dotenv
* Express
* Mysql
* Sequelize
* Nodemon
* XAMP

**Esquema (Chat bot)**

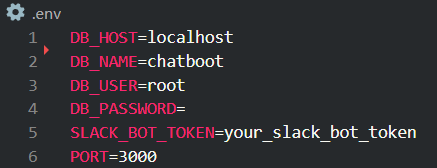


**Esquema (Programador)**

**Chatbotlerdo**

* **Config**
  + database.js
* **Models**
  + chatbotModel.js
* **Routes**
  + chatbotRoutes.js
* **Controllers**
  + chatbotController.js
* **Public**
  + index.html
  + script.js
  + style.css
  + logo.png
* .env
* app.js
* package-lock.json
* package.json

**.env**

****

DB\_HOST=localhost

Define la dirección del servidor de la base de datos. localhost indica que la base de datos está en la misma máquina que la aplicación.

DB\_NAME=chatboot

Especifica el nombre de la base de datos a la que la aplicación se conectará.

DB\_USER=root

Nombre de usuario utilizado para conectarse a la base de datos. root es el usuario administrador predeterminado en MySQL.

DB\_PASSWORD=

Contraseña del usuario de la base de datos. En este caso, está vacía, lo que significa que el usuario root no tiene una contraseña establecida. Esto es común en configuraciones locales, pero no es seguro para entornos de producción.

SLACK\_BOT\_TOKEN=your\_slack\_bot\_token

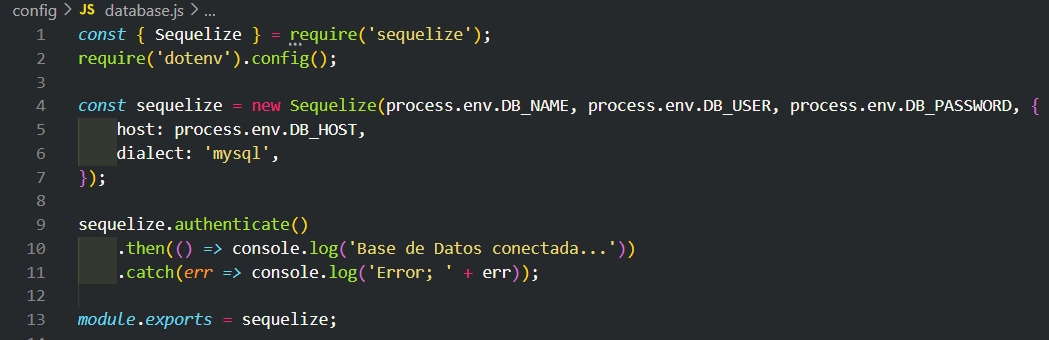
Contiene el token de autenticación para un bot de Slack. your\_slack\_bot\_token es un marcador de posición que debes reemplazar con el token real proporcionado por Slack para autenticar tu bot y permitir que interactúe con Slack.

PORT=3000

Puerto en el que la aplicación web escuchará las solicitudes.

**Config**

**Database.js**

****

**Funcionamiento general:**

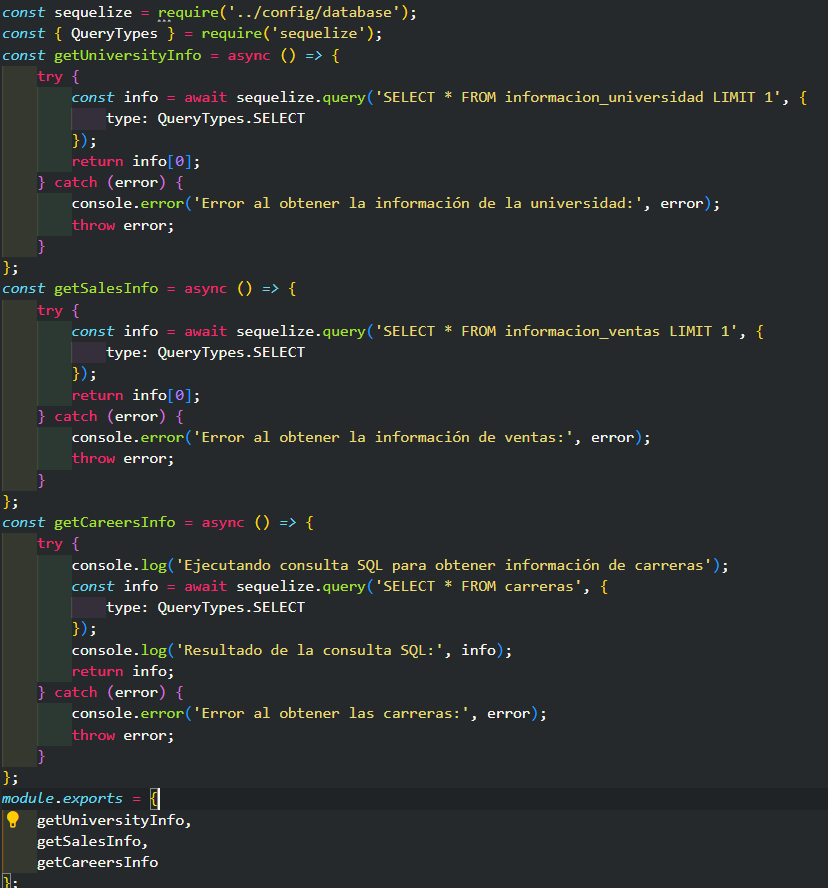
Este código establece una conexión a una base de datos MySQL utilizando Sequelize.

Utiliza variables (usadas en el archivo .env) de entorno para configurar de forma segura las credenciales de la base de datos.

La conexión se prueba mediante sequelize.authenticate(), que verifica si la conexión es exitosa o no.

**Models**

**chatbotModel.js**

****

**Funcionamiento General**

1. **Importaciones:**

const sequelize = require ('../config/database');: Importa la instancia de Sequelize configurada en otro módulo, que permite realizar consultas a la base de datos.

const { QueryTypes } = require('sequelize');: Importa QueryTypes para especificar el tipo de consulta SQL que se realizará.

1. **Obtener Información:**

* getUniversityInfo:

Consulta SQL: Realiza una consulta a la tabla informacion\_universidad y obtiene la primera fila (usando LIMIT 1).

Manejo de Errores: Si ocurre un error, se captura y se muestra un mensaje en la consola, luego se lanza el error nuevamente.

Retorno: Devuelve la primera fila de resultados de la consulta.

* getSalesInfo:

Consulta SQL: Realiza una consulta a la tabla informacion\_ventas y también limita el resultado a la primera fila.

Manejo de Errores: Captura y muestra errores de la misma manera que en la función anterior.

Retorno: Devuelve la primera fila de resultados.

* getCareersInfo:

Consulta SQL: Realiza una consulta a la tabla carreras sin límite, lo que significa que devuelve todas las filas.

Registro en Consola: Antes de ejecutar la consulta, se registra un mensaje en la consola. También se registra el resultado de la consulta para verificación.

Manejo de Errores: Captura errores y muestra un mensaje en la consola.

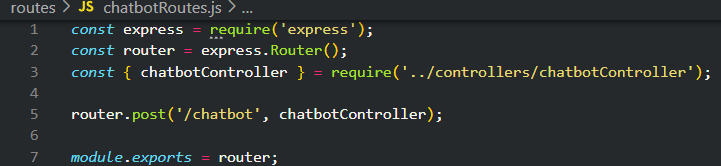
Retorno: Devuelve todas las filas de resultados.

1. **Exportación de Funciones:**

Se exportan las funciones getUniversityInfo, getSalesInfo y getCareersInfo para que puedan ser utilizadas en otros módulos, como en el controlador del chatbot.

**Routes**

**ChatborRoutes.js**

****

**Funcionamiento General**

1. **Importación de Módulos:**

const express = require ('express');: Importa el módulo Express, que es un framework para construir aplicaciones web en Node.js.

const router = express.Router();: Crea un nuevo enrutador (router) de Express, que permite definir rutas y sus manejadores.

1. **Importación del Controlador**

const { chatbotController } = require ('../controllers/chatbotController');: Importa el controlador del chatbot, que contiene la lógica para manejar las solicitudes del chatbot.

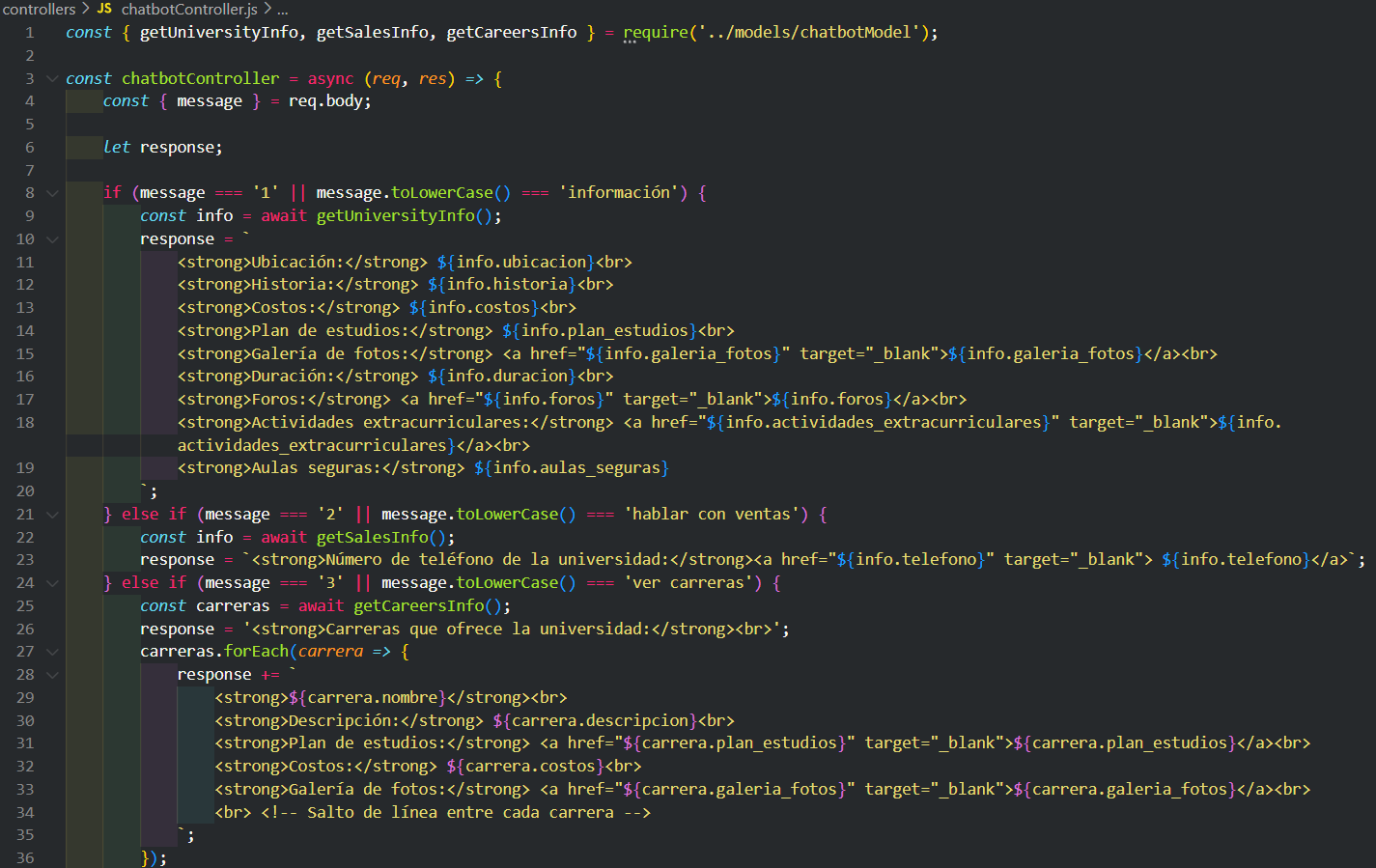
1. **Definición de la Ruta:**

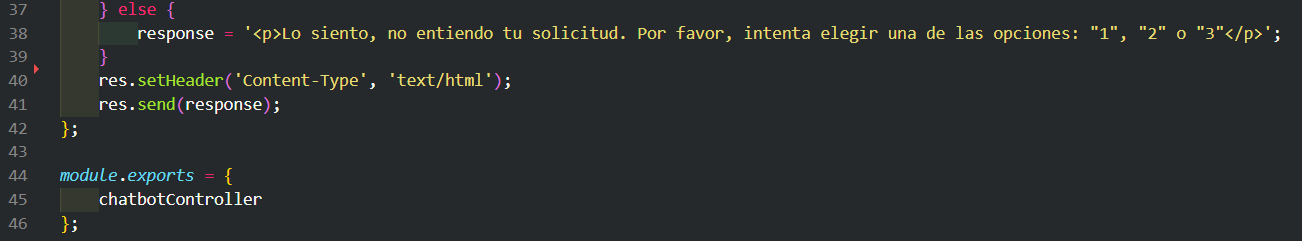
router.post('/chatbot', chatbotController);: Define una ruta que escucha las solicitudes POST en el endpoint /chatbot. Cuando se recibe una solicitud en esta ruta, se llama a chatbotController, que maneja la lógica para procesar el mensaje del usuario y generar una respuesta.

1. **Exportación del Router:**

module.exports = router;: Exporta el enrutador para que pueda ser utilizado en otros módulos, como en el archivo principal de la aplicación donde se configuran las rutas.

**Controllers**

**ChatborController.js**

****

**Funciones generales**

1. **Importación de Funciones**

Se importan las funciones “getUniversityInfo”, “getSalesInfo” y “getCareersInfo” del modelo chatbotModel. Estas funciones son responsables de obtener información de la base de datos.

1. **Definición del Controlador:**

chatbotController: Es una función que maneja las solicitudes HTTP enviadas al chatbot.

1. **Recepción del Mensaje:**

Extrae el mensaje del cuerpo de la solicitud (req.body.message).

1. **Inicialización de la Respuesta:**

Se declara la variable response que se utilizará para almacenar la respuesta que se enviará al usuario.

1. **Manejo de Opciones:**

El controlador evalúa el mensaje recibido y responde de acuerdo con las opciones disponibles:

Opción 1. Información de la Universidad:

Si el mensaje es '1' o 'información', llama a getUniversityInfo() para obtener detalles de la universidad y formatea la respuesta en HTML con información como ubicación, historia, costos, plan de estudios, etc.

Opción 2. Contacto de Ventas:

Si el mensaje es '2' o 'hablar con ventas', llama a getSalesInfo() para obtener el número de teléfono de la universidad, formateando la respuesta para incluir un enlace.

Opción 3. Ver Carreras:

Si el mensaje es '3' o 'ver carreras', llama a getCareersInfo() para obtener una lista de carreras. Itera sobre cada carrera y agrega detalles como nombre, descripción y costos a la respuesta.

Opción no valida:

Si el mensaje no coincide con ninguna opción válida, devuelve un mensaje de error indicando que la solicitud no se entiende y sugiriendo opciones.

1. **Configuración de la Respuesta:**

Establece el encabezado de la respuesta como text/html para indicar que el contenido es HTML.

1. **Envío de la Respuesta:**

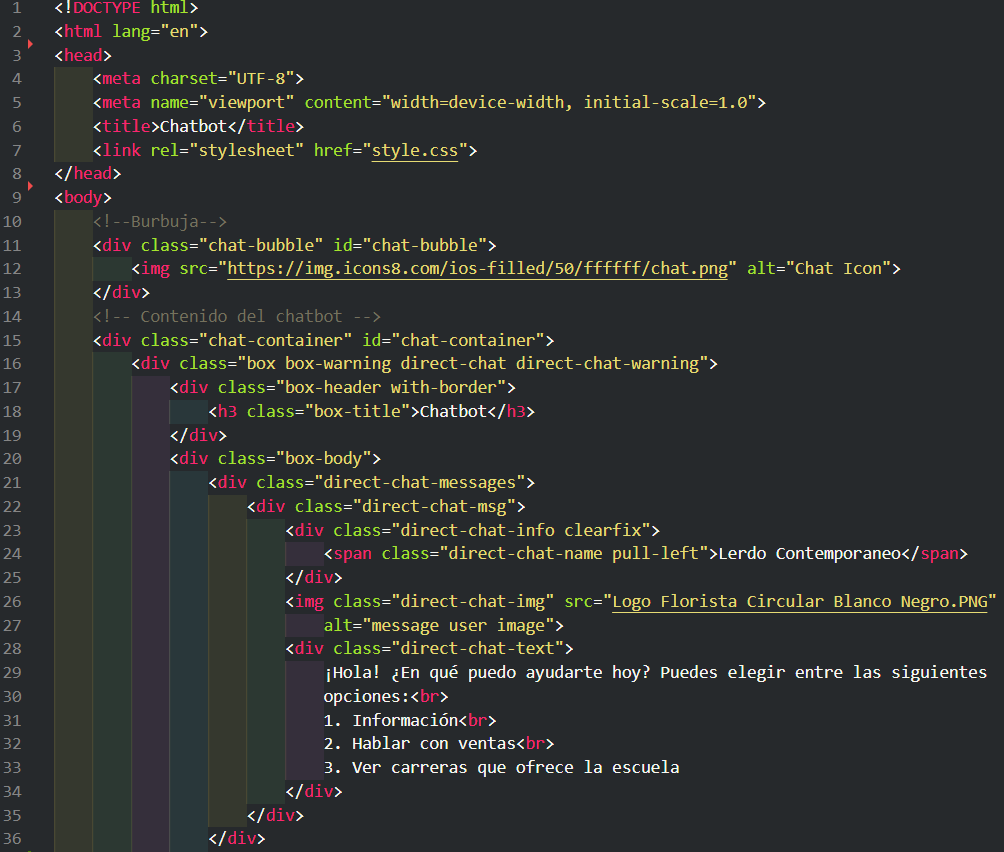
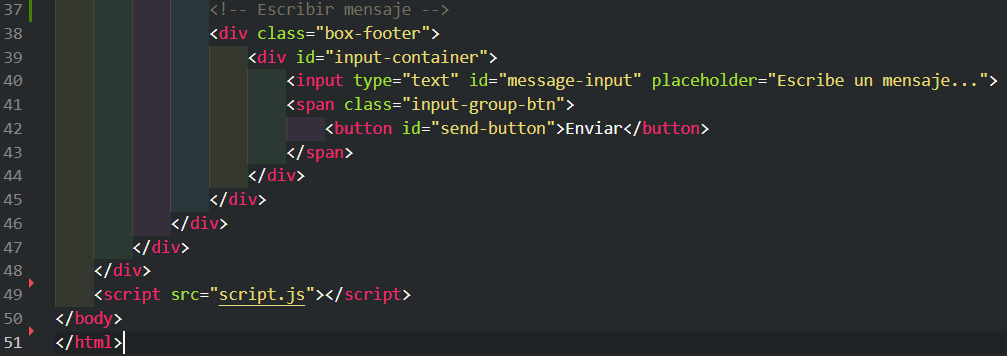
Envía la respuesta al cliente utilizando res.send(response).

1. **Exportación del Controlador:**

Se exporta el controlador para que pueda ser utilizado en otros módulos, como en las rutas de la aplicación.

**Public**

**Index.html**

****

**Funciones Generales**

1. **Estructura HTML:**

<!DOCTYPE html>: Declara el tipo de documento como HTML5.

<html lang="en">: Define el idioma del documento como inglés.

1. **Metadatos:**

<meta charset="UTF-8">: Establece la codificación de caracteres a UTF-8, permitiendo el uso de caracteres especiales.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Asegura que la página sea responsiva en dispositivos móviles.

1. **Título y Estilos:**

<title>Chatbot</title>: Define el título de la página que aparece en la pestaña del navegador.

<link rel="stylesheet" href="style.css">: Enlaza un archivo CSS externo para aplicar estilos a la página.

1. **Burbuja de Chat:**

<div class="chat-bubble" id="chat-bubble">: Crea una burbuja de chat que puede ser visible para el usuario. Contiene un ícono de chat.

<img>: Muestra un ícono gráfico que indica la disponibilidad del chatbot.

1. **Contenedor del Chatbot:**

<div class="chat-container" id="chat-container">: Contenedor principal para el chat, que alberga los mensajes y la interfaz.

<div class="box box-warning direct-chat direct-chat-warning">: Estiliza el contenedor del chat.

1. **Encabezado del Chat:**

<div class="box-header with-border">: Encabezado con el título "Chatbot".

1. **Cuerpo del Chat:**

<div class="box-body">: Contiene los mensajes del chat y la entrada del usuario.

<div class="direct-chat-messages">: Contenedor para los mensajes intercambiados en el chat.

<div class="direct-chat-msg">: Representa un mensaje individual enviado por el chatbot.

<div class="direct-chat-info clearfix">: Muestra información del remitente del mensaje (nombre del chatbot).

1. **Mensaje del Chatbot:**

<div class="direct-chat-text">: Contiene el texto del mensaje enviado por el chatbot, incluyendo las opciones disponibles para el usuario.

1. **Entrada del Usuario:**

<div class="box-footer">: Contenedor para la entrada de mensajes del usuario.

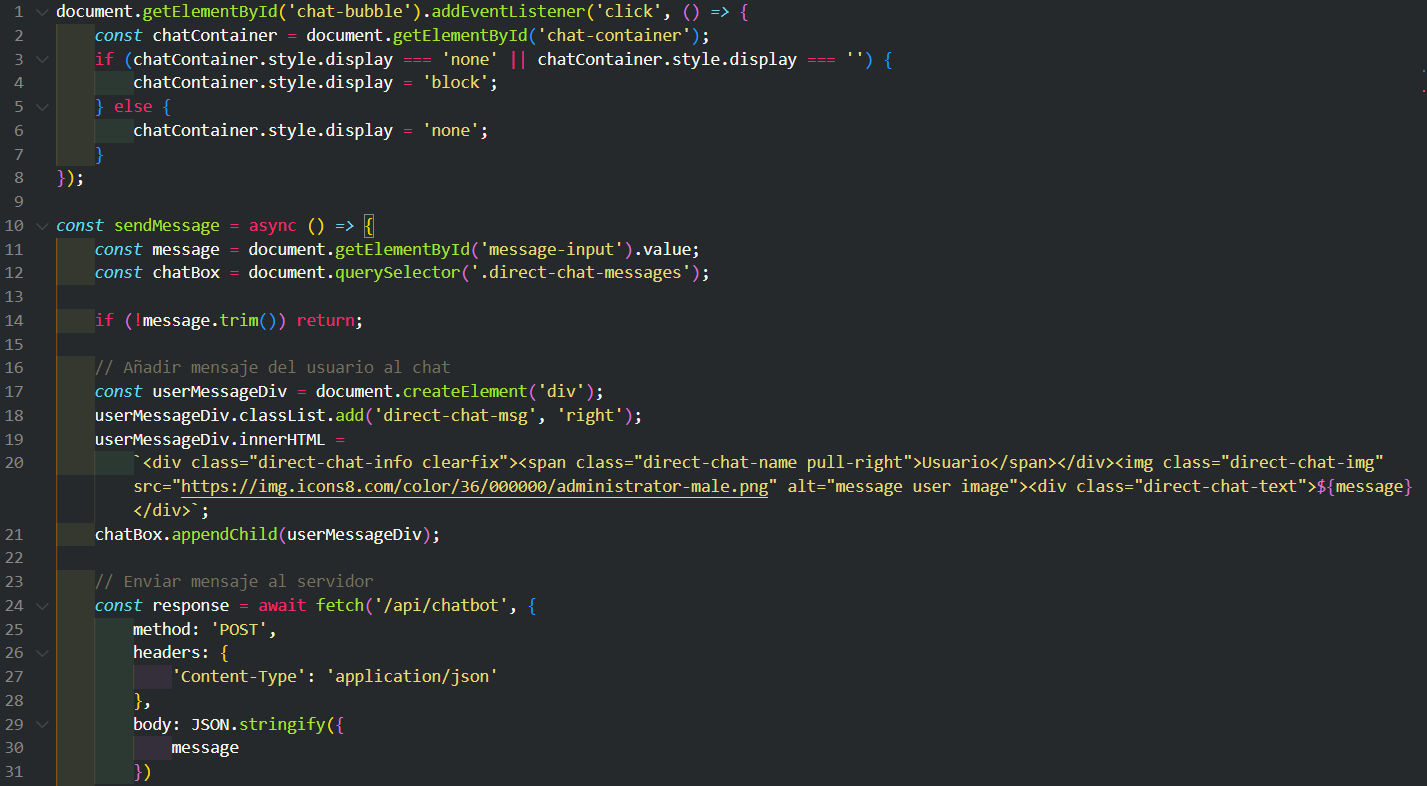
<input type="text" id="message-input" placeholder="Escribe un mensaje...">: Campo de texto donde el usuario puede escribir su mensaje.

<button id="send-button">Enviar</button>: Botón para enviar el mensaje escrito por el usuario.

1. **Integración de JavaScript:**

<script src="script.js"></script>: Enlaza un archivo JavaScript externo, donde se puede definir la lógica para manejar la interacción del usuario con el chatbot.

**Script.js**

****

**Funciones Generales**

1. **Mostrar/Ocultar Chat al Clic en la Burbuja:**

document.getElementById('chat-bubble').addEventListener('click', () => { ... }): Este evento se activa cuando se hace clic en la burbuja de chat (#chat-bubble).

const chatContainer = document.getElementById('chat-container');: Obtiene el contenedor del chat (#chat-container).

Condición de Mostrar/Ocultar: Verifica si el estilo de chatContainer es 'none' o vacío. Si es así, cambia display a 'block' (mostrar el chat); de lo contrario, lo cambia a 'none' (ocultar el chat).

1. **Enviar Mensaje al Servidor:**

const sendMessage = async () => { ... }: Esta función asincrónica se llama para enviar mensajes escritos por el usuario al servidor.

1. **Obtener Mensaje del Campo de Entrada:**

const message = document.getElementById('message-input').value;: Obtiene el mensaje ingresado por el usuario desde el campo de entrada (#message-input).

1. **Añadir Mensaje del Usuario al Chat:**

Crear Elemento HTML: Crea un nuevo elemento div que representa el mensaje del usuario.

Establecer Clases y Contenido: Define clases CSS y contenido HTML que incluye el nombre del usuario y el mensaje.

1. **Enviar y Recibir Respuesta del Servidor:**

await fetch('/api/chatbot', { ... }): Utiliza fetch para enviar una solicitud POST al servidor en la ruta /api/chatbot, con el mensaje del usuario en formato JSON.

const data = await response.text();: Espera la respuesta del servidor, que se convierte en texto.

1. **Añadir Mensaje del Bot al Chat:**

Crear Elemento HTML: Crea un nuevo elemento div que representa el mensaje del bot.

Establecer Clases y Contenido: Define clases CSS y contenido HTML que muestra el nombre del bot y la respuesta recibida del servidor.

1. **Actualizar la Interfaz del Chat:**

chatBox.scrollTop = chatBox.scrollHeight;: Desplaza automáticamentela vista hacia abajo para mostrar el mensaje más reciente.

1. **Limpiar Campo de Entrada:**

document.getElementById('message-input').value = '';: Borra elcontenido del campo de entrada después de enviar el mensaje.

1. **Interacción:**

document.getElementById('send-button').addEventListener('click', sendMessage);: Escucha el clic en el botón de enviar y llama a sendMessage.

document.getElementById('message-input').addEventListener('keypress', (event) => { ... }): Escucha la tecla presionada en el campo de entrada. Si la tecla es 'Enter', llama a sendMessage.

**Style.css**

body {

*background-color*: #959595;

*font-family*: 'Arial', sans-serif;

}

.padding {

*padding*: 3rem;

}

.box {

*position*: fixed;

*bottom*: 20px;

*right*: 5.5rem;

*width*: 300px;

*height*: 400px;

*border-radius*: 10px;

*background*: #ffffff;

*border-top*: 3px solid #d2d6de;

*box-shadow*: 0 1px 1px rgba(0, 0, 0, 0.1);

*overflow*: hidden;

}

.box-warning {

*border-top-color*: #863638;

}

.box-header {

*color*: #444;

*display*: block;

*padding*: 10px;

*position*: relative;

*border-bottom*: 1px solid #f4f4f4;

*border-top-left-radius*: 10px;

*border-top-right-radius*: 10px;

}

.box-header .box-title {

*display*: inline-block;

*font-size*: 18px;

*margin*: 0;

*line-height*: 1;

}

.box-header .box-tools {

*position*: absolute;

*right*: 10px;

*top*: 5px;

}

.btn {

*border-radius*: 3px;

*border*: 1px solid transparent;

}

.btn-box-tool {

*padding*: 5px;

*font-size*: 12px;

*background*: transparent;

*color*: #97a0b3;

}

.bg-red {

*background-color*: #863638 !important;

*color*: #fff !important;

}

.direct-chat .box-body {

*border-bottom-right-radius*: 10px;

*border-bottom-left-radius*: 10px;

*position*: relative;

*overflow-x*: hidden;

*padding*: 0;

}

.direct-chat-messages {

*padding*: 10px;

*height*: 250px;

*overflow*: auto;

*-webkit-transition*: -webkit-transform .5s ease-in-out;

*transition*: transform .5s ease-in-out;

}

.direct-chat-img {

*border-radius*: 50%;

*float*: left;

*width*: 60px;

*height*: 60px;

*overflow*: hidden;

}

.direct-chat-img img {

*width*: 100%;

*height*: auto;

}

.direct-chat-info {

*display*: block;

*margin-bottom*: 2px;

*font-size*: 12px;

}

.direct-chat-img {

*border-radius*: 50%;

*float*: left;

*width*: 40px;

*height*: 40px;

}

.direct-chat-text {

*border-radius*: 10px;

*position*: relative;

*padding*: 5px 10px;

*background*: #d2d6de;

*border*: 1px solid #d2d6de;

*margin*: 5px 0 0 50px;

*color*: #444;

*max-width*: calc(100% - 60px);

*word-wrap*: break-word;

}

.direct-chat-text:before {

*position*: absolute;

*right*: 100%;

*top*: 15px;

*border*: solid transparent;

*border-right-color*: #d2d6de;

*content*: ' ';

*height*: 0;

*width*: 0;

*pointer-events*: none;

*border-width*: 6px;

*margin-top*: -6px;

}

.direct-chat-text:after {

*position*: absolute;

*right*: 100%;

*top*: 15px;

*border*: solid transparent;

*border-right-color*: #d2d6de;

*content*: ' ';

*height*: 0;

*width*: 0;

*pointer-events*: none;

*border-width*: 5px;

*margin-top*: -5px;

}

.right .direct-chat-img {

*float*: right;

}

.direct-chat-warning .right>.direct-chat-text {

*background*: #863638;

*border-color*: #863638;

*color*: #fff;

}

.right .direct-chat-text {

*margin-right*: 50px;

*margin-left*: 0;

}

.box-footer {

*border-top*: 1px solid #f4f4f4;

*padding*: 10px;

*background-color*: #fff;

*border-top-left-radius*: 0;

*border-top-right-radius*: 0;

*border-bottom-right-radius*: 10px;

*border-bottom-left-radius*: 10px;

}

.input-group-btn {

*position*: relative;

*font-size*: 0;

*white-space*: nowrap;

}

.input-group-btn:last-child>.btn {

*z-index*: 2;

*margin-left*: -1px;

}

.btn-warning {

*color*: #fff;

*background-color*: #863638;

*border-color*: #863638;

}

.chat-bubble {

*position*: fixed;

*bottom*: 20px;

*right*: 20px;

*width*: 60px;

*height*: 60px;

*background-color*: #863638;

*border-radius*: 50%;

*display*: flex;

*justify-content*: center;

*align-items*: center;

*cursor*: pointer;

*box-shadow*: 0 4px 8px rgba(0, 0, 0, 0.2);

*z-index*: 1000;

}

.chat-bubble img {

*width*: 30px;

*height*: 30px;

}

.chat-container {

*display*: none;

*position*: fixed;

*bottom*: 90px;

*right*: 20px;

*z-index*: 999;

*margin-bottom*: 10px;

}

#input-container {

*display*: flex;

*align-items*: center;

*margin-top*: 10px;

}

#message-input {

*flex*: 1;

*padding*: 10px;

*border*: 1px solid #d2d6de;

*border-radius*: 5px;

*margin-right*: 10px;

*font-size*: 14px;

*font-family*: 'Arial', sans-serif;

}

#send-button {

*padding*: 10px 15px;

*background-color*: #863638;

*color*: white;

*border*: none;

*border-radius*: 5px;

*cursor*: pointer;

*font-size*: 14px;

*font-family*: 'Arial', sans-serif;

}

#send-button:hover {

*background-color*: #6b2a2a;

}

::-webkit-scrollbar {

*width*: 8px;

}

::-webkit-scrollbar-thumb {

*background-color*: #babac0;

*border-radius*: 10px;

}

::-webkit-scrollbar-track {

*background*: #f1f1f1;

}

::-webkit-scrollbar-thumb:hover {

*background*: #888;

}

::-webkit-scrollbar-thumb:active {

*background*: #555;

}

**Funciones Generales**

1. **Estilo General:**

Fondo y Tipografía: Define un fondo gris (#959595) y utiliza la fuente Arial para todo el cuerpo.

1. **Padding Adaptable:**

Clases de Padding: Aplica diferentes niveles de padding a la clase .padding según el tamaño de la pantalla usando media queries.

1. **Contenedor del Chat:**

Posicionamiento Fijo: La clase .box se posiciona en la esquina inferior derecha de la pantalla, con un diseño limpio y borde redondeado.

Sombras y Bordes: Añade un efecto de sombra y un borde superior para diferenciarlo visualmente.

1. **Encabezado del Chat:**

Diseño de Encabezado: La clase .box-header estiliza el encabezado del chat con un color de texto y un borde inferior

1. **Estilos de Mensajes:**

Diseño de Mensajes: Las clases .direct-chat-messages, .direct-chat-text, y .direct-chat-img controlan la apariencia de los mensajes y las imágenes de perfil, permitiendo un diseño de burbujas.

Burbujas de Mensaje: Las burbujas de mensajes se diseñan con bordes redondeados y flechas que apuntan al remitente.

1. **Interacción de Usuario:**

Efecto Hover en Botón: Cambia el color del botón de envío cuando el usuario pasa el mouse sobre él.

1. **Estilo del Chatbot:**

Burbuja de Chat: .chat-bubble es una burbuja circular que aparece en la esquina inferior derecha, que al hacer clic despliega el contenedor del chat.

1. **Contenedor de Entrada de Mensajes:**

Estilo del Input y Botón: Se utiliza un diseño flexible para el contenedor de entrada, asegurando que se vea bien y sea fácil de usar.

1. **Scrollbars Personalizados:**

Estilos para Scrollbars: Personaliza la apariencia de las barras de desplazamiento, mejorando la usabilidad y la estética.

**Resultado**

****

**Repositorio**

<https://github.com/RubenCanalesReyes/chartbootl>