

**CENTRO DE ESTUDIOS LERDO CONTEMPORANEO A.C**

**MATERIA:**

**ACTUALIZACION COMPUTACIONAL 1**

**TEMA:**

**CHATBOT**

**DOCENTE:**

**ING. JAVIER MARTINEZ BENAVENTE**

**ALUMNO:**

**JUAN MIGUEL CARREON REYES**

**FECHA:**

**17 DE JULIO DEL 2024**

**Manual: Creación de un Chatbot con Node.JS y MySQL**

**Introducción**

En este tutorial, aprenderemos cómo crear un chatbot utilizando Node.js y MySQL. El chatbot podrá responder a dos tipos de solicitudes: información general sobre una universidad y las carreras que ofrece. La información se recuperará de una base de datos MySQL.

DATOS QUE OCUPAMOS PARA ESTE CHAT BOT

* Procesar información
* Proyecto
* Actualización
* Reporte
* Capacitación

TODO CON INTELIGENCIA ARTIFICIAL Y MACHINE LEARNING

elaboraremos chatbot

EL SALUDO Y TENDRA TRES RESPUESTAS

información:

1. Ubicación
2. Historia
3. Costos
4. Carreras
5. Planes de estudio
6. Galería de fotos -- alumnos
7. Duración
8. Foros
9. Extracurriculares
10. Instalaciones (plantel en general)
11. Aula segura

Ventas:

1. Asesor (esperar a que un asesor te atienda (tiempo de espera 1 minuto))

Ver carreras (imagen):

* Lic. Derecho
* Lic. Sistemas
* Lic. Admón.
* Lic. CP
* Lic. Arquitectura

En ellos se pueden mostrar planes de estudio, costos, galería de fotos de los alumnos y proyectos.

Casos de éxito(pedir éxitos )

Hablar con un asesor: (para respuestas no reconocidas)

**Esquema de la Base de Datos**

**Tabla: university\_info**

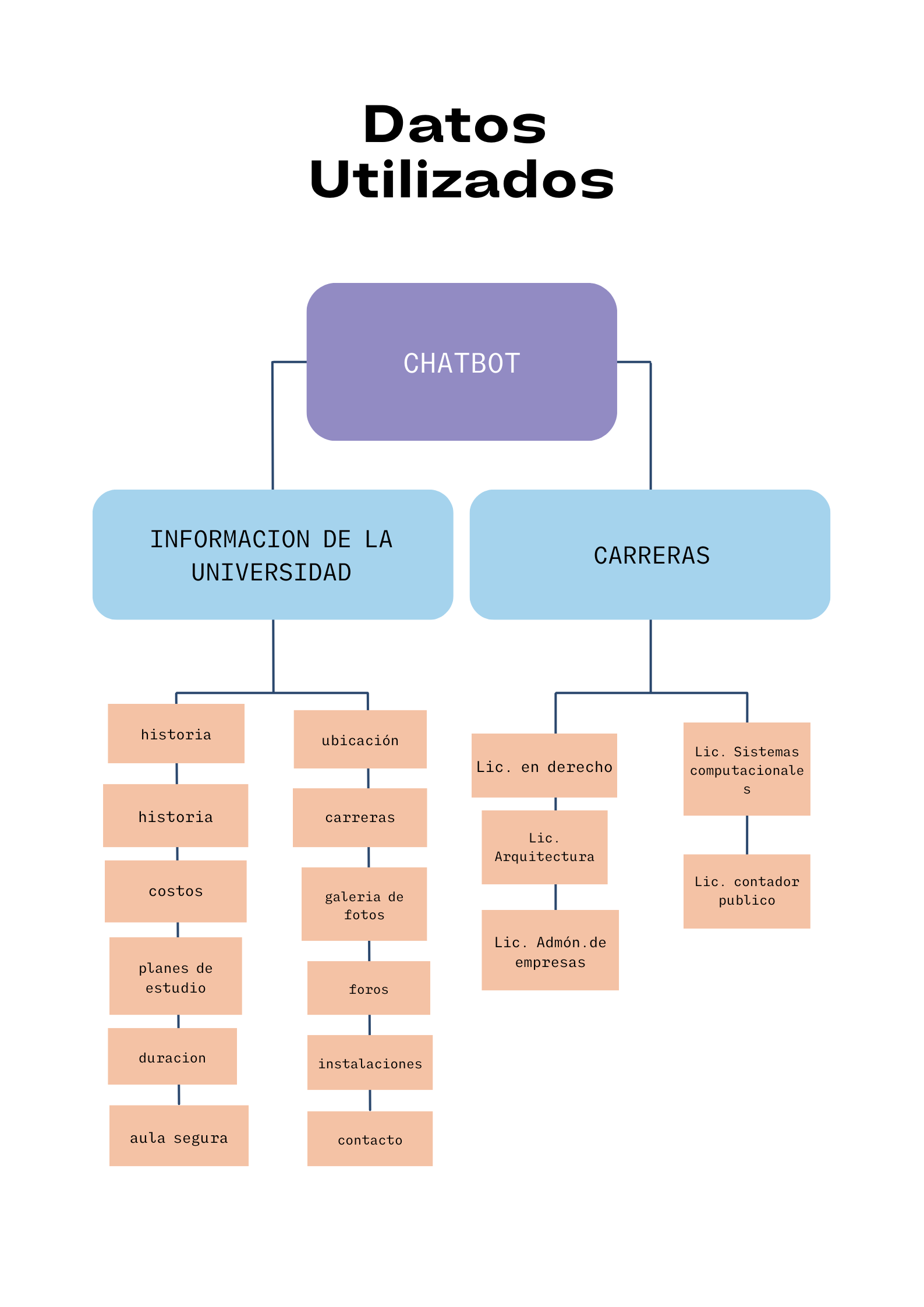
Almacena información general sobre la universidad.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Columna** | **Tipo de Datos** | **Descripción** | | id | INT (PK) | Identificador único | | location | VARCHAR(255) | Ubicación de la universidad | | history | TEXT | Historia de la universidad | | costs | VARCHAR(255) | Costos de la universidad | | study\_plans | TEXT | Planes de estudio | | gallery | TEXT | URLs de la galería de fotos | | duration | VARCHAR(255) | Duración de los programas académicos | | forums | TEXT | URLs de los foros | | extracurricular | TEXT | Actividades extracurriculares | | facilities | TEXT | Descripción de las instalaciones | | safe\_classroom | TEXT | Descripción del aula segura | | whatsapp\_contact | VARCHAR(255) | Contacto de WhatsApp | |

**Tabla: careers**

Almacena información sobre las carreras que ofrece la universidad.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Columna | Tipo de Datos | Descripción | | id | INT (PK) | Identificador único | | name | VARCHAR(255) | Nombre de la carrera | | description | TEXT | Descripción de la carrera | | image\_url | VARCHAR(255) | URL de la imagen representativa | |



Conclusión

Este manual de usuario describe los archivos esenciales de un proyecto de chatbot universitario y explica cómo funcionan juntos para proporcionar una interfaz interactiva que permite a los usuarios obtener información sobre la universidad y sus programas. Asegúrate de ajustar las configuraciones y detalles según las necesidades específicas de tu aplicación.

Nosotros decidimos subir el proyecto en un repositorio a GitHub mediante los siguientes pasos.

**Requisitos**

* Node.js instalado
* MySQL instalado y configurado(Xampp)
* Editor de texto o IDE (como Visual Studio Code)
* Navegador web

**PASO 1: CONFIGURACIÓN DEL PROYECTO**

1. **Inicializa un nuevo proyecto de Node.js:**

bash

Ingresar código

mkdir chatbotLerdo

cd chatbotLerdo

npm init -y

1. **Instala las dependencias necesarias:**

bash

Ingresar código

npm install express mysql body-parser nodemon

1. **Estructura del proyecto:**

chatbotLerdo/

├── config/

│ └── database.js

├── controllers/

│ └── chatbotController.js

├── models/

│ └── chatbotModel.js

├── public/

│ ├── index.html

│ ├── style.css

│ └── script.js

├── routes/

│ └── chatbotRoutes.js

└── app.js

**PASO 2: CONFIGURACIÓN DE LA BASE DE DATOS**

Crea una base de datos en MySQL y asegúrate de tener una tabla para la información de la universidad y otra para las carreras.

**config/database.js**

JavaScript

Ingresar código

const mysql = require('mysql');

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

database: 'your\_database'

});

connection.connect(error => {

if (error) throw error;

console.log('Database connected!');

});

module.exports = connection;

**PASO 3: MODELOS PARA ACCEDER A LOS DATOS**

**models/chatbotModel.js**

javascript

Copiar código

const db = require('../config/database');

const getUniversityInfo = () => {

return new Promise((resolve, reject) => {

db.query('SELECT \* FROM university\_info LIMIT 1', (error, results) => {

if (error) return reject(error);

resolve(results[0]);

});

});

};

const getCareersInfo = () => {

return new Promise((resolve, reject) => {

db.query('SELECT \* FROM careers', (error, results) => {

if (error) return reject(error);

resolve(results);

});

});

};

module.exports = {

getUniversityInfo,

getCareersInfo

};

**PASO 4: CONTROLADOR PARA GESTIONAR LAS SOLICITUDES**

**controllers/chatbotController.js**

javascript

Copiar código

const { getUniversityInfo, getCareersInfo } = require('../models/chatbotModel');

const chatbotController = async (req, res) => {

const { message } = req.body;

let response;

if (message === '1' || message.toLowerCase() === 'información') {

try {

const info = await getUniversityInfo();

response = {

location: info.location,

history: info.history,

costs: info.costs,

study\_plans: info.study\_plans,

gallery: info.gallery,

duration: info.duration,

forums: info.forums,

extracurricular: info.extracurricular,

facilities: info.facilities,

safe\_classroom: info.safe\_classroom,

whatsapp\_contact: info.whatsapp\_contact

};

} catch (error) {

response = 'Hubo un error al obtener la información de la universidad.';

}

} else if (message === '2' || message.toLowerCase() === 'ver carreras') {

try {

const careers = await getCareersInfo();

response = careers.map(career => ({

name: career.name,

description: career.description

}));

} catch (error) {

response = 'Hubo un error al obtener la información de las carreras.';

}

} else {

response = 'Lo siento, no entiendo tu solicitud. Por favor, intenta elegir una de las opciones: "1", "2"';

}

res.json({ response });

};

module.exports = {

chatbotController

};

**Explicación:**

* Maneja las solicitudes del chatbot y responde según el mensaje recibido.
* Obtiene información de la base de datos utilizando funciones del modelo.
* Retorna la respuesta en formato JSON.

**PASO 5: DEFINIR LAS RUTAS**

**routes/chatbotRoutes.js**

const express = require('express');

const router = express.Router();

const { chatbotController } = require('../controllers/chatbotController');

router.post('/chatbot', chatbotController);

module.exports = router;

**Explicación:**

* Define la ruta /chatbot para manejar solicitudes POST.
* Utiliza el controlador del chatbot (chatbotController) para procesar las solicitudes.

**PASO 6: CONFIGURAR EL SERVIDOR**

**app.js**

const express = require('express');

const bodyParser = require('body-parser');

const chatbotRoutes = require('./routes/chatbotRoutes');

const app = express();

const PORT = process.env.PORT || 3000;

app.use(bodyParser.json());

app.use(express.static('public'));

app.use('/api', chatbotRoutes);

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

**Explicación:**

* Configura el servidor Express.
* Utiliza body-parser para manejar datos JSON en las solicitudes.
* Define rutas estáticas para servir archivos públicos (HTML, CSS, JS).
* Inicia el servidor en el puerto especificado.

**PASO 7: CREAR LA INTERFAZ DE USUARIO**

**public/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Chatbot</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="chat-container">

<div class="chat-box" id="chat-box">

<div class="message bot-message">¡Hola! ¿En qué puedo ayudarte hoy? Puedes elegir entre las siguientes opciones:<br>1. Información<br>2. Ver carreras que ofrece la escuela</div>

</div>

<div class="input-box">

<input type="text" id="message-input" placeholder="Escribe un mensaje...">

<button id="send-button">Enviar</button>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

**Explicación:**

* Define la estructura HTML para la interfaz del chatbot.
* Incluye el archivo de estilos CSS (style.css) y el script JavaScript (script.js).

**public/style.css**

body {

font-family: Arial, sans-serif;

background-color: #f5f5f5;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.chat-container {

width: 100%;

max-width: 500px;

border: 1px solid #ddd;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

background-color: #fff;

overflow: hidden;

display: flex;

flex-direction: column;

}

.chat-box {

flex: 1;

padding: 20px;

overflow-y: auto;

display: flex;

flex-direction: column;

gap: 10px;

}

.message {

padding: 10px;

border-radius: 10px;

max-width: 80%;

}

.bot-message {

background-color: #e1f5fe;

align-self: flex-start;

text-align: start;

}

.user-message {

background-color: #c8e6c9;

align-self: flex-end;

}

.input-box {

display: flex;

border-top: 1px solid #ddd;

}

input[type="text"] {

flex: 1;

padding: 10px;

border: none;

border-radius: 0 0 0 10px;

outline: none;

font-size: 16px;

}

button {

padding: 10px 20px;

background-color: #2196f3;

color: white;

border: none;

border-radius: 0 0 10px 0;

cursor: pointer;

font-size: 16px;

}

button:hover {

background-color: #1976d2;

}

**Explicación:**

* Define los estilos para la apariencia del chatbot.
* Utiliza flexbox para organizar los elementos de la interfaz de usuario.

**public/script.js**

document.getElementById('send-button').addEventListener('click', async () => {

const message = document.getElementById('message-input').value;

const chatBox = document.getElementById('chat-box');

// Añadir mensaje del usuario al chat

const userMessageDiv = document.createElement('div');

userMessageDiv.classList.add('message', 'user-message');

userMessageDiv.textContent = message;

chatBox.appendChild(userMessageDiv);

// Enviar mensaje al servidor

const response = await fetch('/api/chatbot', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ message })

});

const data = await response.json();

let botMessageDiv = document.createElement('div');

botMessageDiv.classList.add('message', 'bot-message');

if (typeof data.response === 'string') {

// Mensaje simple

botMessageDiv.innerHTML = data.response.replace(/\n/g, '<br>');

} else if (Array.isArray(data.response)) {

// Lista de carreras

botMessageDiv.innerHTML = data.response.map(career => `

<div>

<strong>Carrera:</strong> ${career.name}<br>

<strong>Descripción:</strong> ${career.description}<br>

</div>

`).join('');

} else {

// Información de la universidad

botMessageDiv.innerHTML = `

<strong>Información:</strong> ${data.response.location}<br>

<strong>Historia:</strong> ${data.response.history}<br>

<strong>Costos:</strong> ${data.response.costs}<br>

<strong>Plan de estudios:</strong> ${data.response.study\_plans}<br>

<strong>Galería:</strong> <a href="${data.response.gallery}">${data.response.gallery}</a><br>

<strong>Duración:</strong> ${data.response.duration}<br>

<strong>Foros:</strong> <a href="${data.response.forums}">${data.response.forums}</a><br>

<strong>Extracurricular:</strong> <a href="${data.response.extracurricular}">${data.response.extracurricular}</a><br>

<strong>Instalaciones:</strong> ${data.response.facilities}<br>

<strong>Aula Segura:</strong> ${data.response.safe\_classroom}<br>

<strong>Contáctanos:</strong> <a href="${data.response.whatsapp\_contact}">${data.response.whatsapp\_contact}</a><br>

`;

}

chatBox.appendChild(botMessageDiv);

// Desplazar hacia abajo

chatBox.scrollTop = chatBox.scrollHeight;

// Limpiar el campo de entrada de mensaje

document.getElementById('message-input').value = '';

});

**Explicación:**

* Añade un event listener al botón de enviar para capturar el mensaje del usuario.
* Envía el mensaje al servidor y recibe la respuesta del chatbot.
* Añade tanto el mensaje del usuario como la respuesta del chatbot al chat.

**PASO 8: EJECUTAR EL PROYECTO**

1. **Inicia el servidor Node.js:**

bash

Copiar código

nodemon app.js

1. **Abre el navegador y navega a http://localhost:3000 para interactuar con el chatbot.**

¡Felicidades! Has creado un chatbot funcional que interactúa con una base de datos MySQL para proporcionar información sobre una universidad y las carreras que ofrece.

**Manual de Usuario: Cómo Subir un Repositorio a GitHub**

Este manual proporciona una guía paso a paso para subir tu proyecto a GitHub. GitHub es una plataforma de alojamiento de código que permite a los desarrolladores almacenar, compartir y colaborar en proyectos de software. A continuación, se detallan los pasos para crear un repositorio en GitHub y subir tu proyecto.

**Prerrequisitos**

* Una cuenta en GitHub. Si no tienes una, puedes registrarte en [github.com](https://github.com).
* Git instalado en tu sistema. Puedes descargarlo e instalarlo desde [git-scm.com](https://git-scm.com).

**Pasos para Subir un Proyecto a GitHub**

**Paso 1: Crear un Repositorio en GitHub**

1. Inicia sesión en tu cuenta de GitHub.
2. En la esquina superior derecha, haz clic en el icono de + y selecciona New repository.
3. Rellena los detalles del repositorio:
   * **Repository name**: Elige un nombre para tu repositorio.
   * **Description** (opcional): Añade una descripción para tu proyecto.
   * **Public/Private**: Elige si quieres que tu repositorio sea público o privado.
4. Opcionalmente, puedes inicializar el repositorio con un archivo README.md y/o un archivo .gitignore.
5. Haz clic en Create repository.

**Paso 2: Inicializar el Repositorio Localmente**

1. Abre tu terminal o consola.
2. Navega hasta el directorio de tu proyecto. Usa el comando cd seguido de la ruta de tu proyecto.

Código:

cd ruta/al/directorio/de/tu/proyecto

Inicializa un repositorio Git en tu directorio local:

Código:

git init

**Paso 3: Conectar el Repositorio Local con GitHub**

1. Añade los archivos de tu proyecto al repositorio:

git add .

El comando git add . añade todos los archivos y directorios en el directorio actual al área de preparación.

1. Crea un commit inicial:

git commit -m "Initial commit"

El mensaje entre comillas describe los cambios que estás confirmando.

1. Añade la URL del repositorio remoto que has creado en GitHub. Reemplaza URL-del-repositorio con la URL real de tu repositorio:

git remote add origin URL-del-repositorio

1. Verifica que el repositorio remoto se ha añadido correctamente:

git remote -v

**Paso 4: Subir el Proyecto a GitHub**

1. Sube los cambios al repositorio remoto en GitHub:

git push -u origin master

Si estás utilizando una rama principal diferente, como main, usa:

bash

Copiar código

git push -u origin main

**Paso 5: Verificar en GitHub**

1. Abre tu navegador y ve a tu cuenta de GitHub.
2. Navega hasta el repositorio que has creado.
3. Verifica que todos los archivos de tu proyecto se han subido correctamente.

**Consejos Adicionales**

* **Actualizar el Repositorio**: Cada vez que realices cambios en tu proyecto, puedes actualizar el repositorio remoto con los siguientes comandos:

git add .

git commit -m "Mensaje descriptivo de los cambios"

git push origin master

* **Colaboradores**: Si trabajas con un equipo, puedes añadir colaboradores a tu repositorio desde la configuración del repositorio en GitHub.

**Conclusión**

Este manual te ha guiado a través de los pasos necesarios para subir un proyecto a GitHub. Siguiendo estos pasos, puedes gestionar tu código de manera eficiente y colaborar con otros desarrolladores en la plataforma de GitHub.