Deployment and Integration of Gemini AI Chatbot using Node.js

# 1. Introduction

This document provides a detailed account of the deployment and integration process of an AI-powered chatbot based on Google's Gemini language model. The backend of the chatbot was implemented using Node.js, while the frontend was embedded into a PHP-based website. The system facilitates dynamic natural language interaction and supports multiple languages.

# 2. Backend Configuration and Request Handling using gemini.js

The core backend logic resides in the gemini.js file. It starts by importing the necessary Node.js modules:

import { GoogleGenerativeAI } from "@google/generative-ai";  
import dotenv from "dotenv";  
import express from "express";  
import bodyParser from "body-parser";  
import cors from "cors";  
import languageDetect from 'languagedetect';

• dotenv is used to load the Gemini API key securely from an environment variable.  
• express is the framework that handles routing and server operations.  
• body-parser parses incoming JSON requests.  
• cors allows requests from the PHP frontend.  
• languagedetect enables detection of the user's message language.

const googleAI = new GoogleGenerativeAI(process.env.gemini\_api\_key);  
const geminiConfig = { temperature: 0.8, topP: 1, topK: 1, maxOutputTokens: 1024 };  
const geminiModel = googleAI.getGenerativeModel({ model: "gemini-1.5-flash-8b", geminiConfig });

app.use('/chat', async (req, res, next) => {  
 const userMessage = req.body.message;  
 if (userMessage) {  
 try {  
 const encoding = await geminiModel.countTokens(userMessage);  
 res.locals.inputTokens = encoding.totalTokens;  
 } catch (error) {  
 res.locals.inputTokens = null;  
 }  
 }  
 next();  
});

app.use('/chat', async (req, res, next) => {  
 const userMessage = req.body.message.toLowerCase();  
 let prompt = userMessage + "\n\n";  
 if (contactKeywords.some(keyword => userMessage.includes(keyword))) {  
 prompt += "Answer using the following contact information: ...";  
 } else {  
 prompt += "Answer this in British English, Traditional Chinese (Hong Kong)...";  
 }  
 req.body.processedPrompt = prompt;  
 next();  
});

app.post('/chat', async (req, res) => {  
 const prompt = req.body.processedPrompt;  
 const result = await geminiModel.generateContent(prompt);  
 const response = result.response;  
 let aiResponse = response.text().replace(/<br>/gi, '\n');  
 res.json({ response: aiResponse.trim(), inputTokens: res.locals.inputTokens });  
});

# 3. Frontend Integration using (chatbot\_widget.js)

The chatbot widget is injected into the web page and supports toggling, message input, and interaction:

document.addEventListener('DOMContentLoaded', function () {  
 const chatbotButton = document.getElementById('chatbot-button');  
 const chatContainer = document.getElementById('chat-container');  
 ...  
 function sendMessage() {  
 ...  
 const response = await fetch(apiEndpoint, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ message: message })  
 });  
 const data = await response.json();  
 displayMessage('ai', data.response);  
 }  
});

# 4. Embedding Chatbot Widget using PHP (chatbot\_widget.php)

The following PHP function injects the chatbot widget into the HTML layout of the website:

function chatbotWidget() {  
 $html = <<<HTML  
 <div id="chatbot-button-container" ...>...</div>  
 <div id="chat-container">...</div>  
HTML;  
 echo $html;  
}

# 5. Deployment Steps

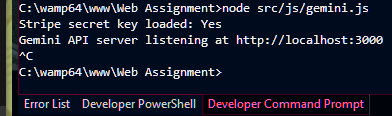
Begin by creating a suitable folder inside the web server’s root directory, ensuring the backend files remain separate from frontend assets.  
mkdir "C:\wamp64\www\Web Assignment\src\js" (You need /src/js if using Webpack)

Initialise the project to generate a package.json file, which defines metadata and dependencies. Then, install the modules necessary for the chatbot service. @latest  
npm init -y  
npm install express dotenv cors body-parser languagedetect @google/generative-ai

At the project root, create a .env file to securely store the Gemini API key. Replace GOOGLE\_API\_KEY with your actual Gemini key. Never expose this file in public repositories!  
In .env => gemini\_api\_key=GOOGLE\_API\_KEY

Create a new file named gemini.js in the same directory and paste the configured backend script.  
Ensure that the script:

* Imports the required modules.
* Defines the Gemini model and configurations.
* Handles /chat POST requests.
* Provides middleware for token logging and language-based prompt manipulation. (Optional)

Use Node or a process manager like PM2 to launch the chatbot server, this command will persist the process across terminal sessions. I added console log.  
npx pm2 start gemini.js --name gemini  
Can exit by Ctrl+C

Ensure the frontend (e.g., chatbot\_widget.js and chatbot\_widget.php) is embedded into your PHP pages or added in head.php and that the endpoint “http://localhost:3000/chat” matches the server.

**Deployment Considerations**

* Always protect .env in production.
* Use HTTPS and firewall rules in real-world hosting.
* Ensure CORS is restricted to known domains.
* Monitor token usage to avoid quota breaches.

# 6. Conclusion

The integration of Google's Gemini model into a Node.js backend and a PHP website frontend demonstrates a practical application of AI in user interaction. The setup includes translation logic, localisation handling, and token monitoring. This deployment leverages the flexibility of Node.js and the scalability of Google’s generative AI models to create a multilingual assistant experience embedded within a dynamic web platform.

# Read on GitHub

<https://github.com/Test-Plus-XD/Web-Assignment>