📝 PROMPT ENDPOINT

Modifica este archivo TypeScript para usar un endpoint PHP en vez de llamar directamente a Gemini API.

REGLAS:

1. ELIMINAR:
   - import { GoogleGenAI } from "@google/genai"; (y cualquier variante)
   - import { ... Type ... } from "@google/genai"; (si existe)
   - const API_KEY = process.env.API_KEY; (o variantes)
   - if (!API_KEY) { throw new Error(...); }
   - const ai = new GoogleGenAI({ apiKey: ... });

2. AGREGAR después de los imports:
   const API_ENDPOINT = '**https://TU-DOMINIO.com**/gemini-api.php';

3. NO TOCAR la función buildPrompt() - déjala exactamente como está

4. MODIFICAR solo la función que hace la llamada a Gemini (usualmente generateContentIdeas o similar):

   BUSCAR el patrón:
   ```
   const response = await ai.models.generateContent({
       model: 'modelo-nombre',
       contents: prompt,
       config: { ... }
   });
   ```

   REEMPLAZAR por:
   ```
   const apiResponse = await fetch(API_ENDPOINT, {
       method: 'POST',
       headers: { 'Content-Type': 'application/json' },
       body: JSON.stringify({
           model: 'modelo-nombre',
           prompt: prompt,
           useGoogleSearch: [true si config tiene googleSearch, false si no],
           useThinkingMode: [true si config tiene thinkingConfig, false si no],
           responseFormat: [json si config tiene responseMimeType, text si no]
       })
   });

   if (!apiResponse.ok) {
       throw new Error(`API Error: ${apiResponse.status}`);
   }

   const apiResult = await apiResponse.json();
   ```

```
  if (!apiResult.text) {
      throw new Error('Invalid response from API');
  }

  const response = { text: apiResult.text };
```

5. Después de obtener response, el resto del código debe quedar igual (el .replace(), .trim(), etc.)

6. MANTENER sin cambios:
   - Todos los imports de tipos locales
   - La función buildPrompt() completa
   - Los tipos de las funciones
   - El manejo de errores existente
   - La lógica de limpieza del texto

Dame el código completo modificado.

EL CÓDIGO DEL ARCHIVO geminiService.ts ES EL SIGUIENTE:

PEGAR CÓDIGO ACÁ

📄 CÓDIGO PHP (gemini-api.php)

```
<?php
/**
 * Endpoint PHP para Gemini API
 * Protege la API Key actuando como proxy
 */

header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: POST, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type');

// Manejar preflight CORS
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200);
    exit;
}

// Solo aceptar POST
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    http_response_code(405);
    echo json_encode(['error' => 'Only POST method allowed']);
    exit;
}

// ⭐ CONFIGURA TU API KEY AQUÍ ⭐
$apiKey = 'TU-API-KEY';

// Leer request body
$rawData = file_get_contents('php://input');
$data = json_decode($rawData, true);

if (json_last_error() !== JSON_ERROR_NONE) {
    http_response_code(400);
    echo json_encode(['error' => 'Invalid JSON in request body']);
    exit;
}

// Extraer parámetros
$model = $data['model'] ?? 'gemini-2.5-flash';
$prompt = $data['prompt'] ?? '';
$useGoogleSearch = $data['useGoogleSearch'] ?? false;
$useThinkingMode = $data['useThinkingMode'] ?? false;
$responseFormat = $data['responseFormat'] ?? 'text';

// Validar prompt
if (empty($prompt)) {
```

```
    http_response_code(400);
    echo json_encode(['error' => 'Prompt is required']);
    exit;
}

// Construir URL
$url =
"https://generativelanguage.googleapis.com/v1beta/models/{$model}:generateContent?key=
{$apiKey}";

// Construir payload
$payload = [
    'contents' => [
        ['parts' => [['text' => $prompt]]]
    ]
];

// Configurar respuesta JSON
if ($responseFormat === 'json') {
    if (!isset($payload['generationConfig'])) {
        $payload['generationConfig'] = [];
    }
    $payload['generationConfig']['responseMimeType'] = 'application/json';
}

// Hacer petición a Gemini
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($payload));
curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/json']);
curl_setopt($ch, CURLOPT_TIMEOUT, 120);

$response = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curlError = curl_error($ch);
curl_close($ch);

// Manejar errores de conexión
if ($curlError) {
    http_response_code(500);
    echo json_encode(['error' => 'Connection failed', 'details' => $curlError]);
    exit;
}

// Manejar errores HTTP
if ($httpCode !== 200) {
    http_response_code($httpCode);
```

```
    echo $response;
    exit;
}

// Procesar respuesta
$result = json_decode($response, true);

if (!isset($result['candidates'][0]['content']['parts'][0]['text'])) {
    http_response_code(500);
    echo json_encode(['error' => 'Invalid Gemini API response']);
    exit;
}

$text = $result['candidates'][0]['content']['parts'][0]['text'];
echo json_encode(['text' => $text]);
?>
```