Documentación de Avances – Proyecto TerappIA MVP

Resumen

Este documento presenta el registro técnico de los avances realizados en el desarrollo del MVP del proyecto *TerappIA: Prototipo de herramienta web de procesamiento de lenguaje natural para los profesionales de la salud mental*. Se detalla la configuración inicial del entorno, la creación del repositorio, y la generación de la estructura base del sistema que incluye módulos frontend y backend. El objetivo es establecer una base sólida sobre la cual continuar con el desarrollo iterativo de funcionalidades como OCR, análisis de sentimientos ABSA y gestión de expedientes clínicos.

I. Introducción

El presente documento constituye la bitácora técnica de avances del proyecto TerappIA MVP. El propósito es mantener un registro formal, detallado y profesional de las actividades realizadas, facilitando el seguimiento del progreso y asegurando la trazabilidad de las decisiones técnicas.

II. Objetivos

- Documentar los avances realizados en el proyecto hasta la fecha.
- Describir la estructura inicial del repositorio y sus componentes.
- Dejar una base metodológica para registrar futuros progresos del MVP.

III. Metodología

Se siguió un enfoque incremental y modular, priorizando la creación de un entorno de desarrollo funcional desde el inicio, con separación clara entre frontend y backend, y con control de versiones a través de GitHub.

III-A. Herramientas utilizadas

- **Sistema de control de versiones:** Git + GitHub.
- **Frontend:** React (Vite) + TailwindCSS.
- **Backend:** Node.js + Express.
- **Otros:** Git Bash como terminal principal, Visual Studio Code como IDE.

IV. Avances actuales

IV-A. Creación del repositorio en GitHub

Se creó un repositorio público denominado `terappia-mvp` en GitHub, con la descripción del proyecto y sin archivos iniciales (*README*, *.gitignore*).

IV-B. Inicialización local del proyecto

```
mkdir terappia-mvp
cd terappia-mvp
git init
git remote add origin https://github.com/TU_USUARIO/terappia-mvp.git
```

IV-C. Estructura inicial generada

Se estableció la siguiente estructura de carpetas:

```
terappia-mvp/
          frontend/    # Aplicaci n React (Vite + Tailwind)
          backend/     # API Node.js + Express
          .gitignore
          README.md
```

IV-D. Configuración del Frontend

El frontend fue inicializado con Vite:

```
cd frontend
npm create vite@latest
npm install
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Se configuró `tailwind.config.js` para incluir rutas de archivos y se añadió la importación de estilos en `index.css`.

IV-E. Configuración del Backend

El backend fue inicializado con Express:

```
cd backend
npm init -y
npm install express cors multer axios dotenv
```

Archivo `index.js` básico:

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const app = express();

app.use(cors());
app.use(express.json());

app.get("/", (req, res) => {
  res.send("Backend de TerappIA funcionando");
});

app.listen(3001, () => {
  console.log("Servidor backend en puerto 3001");
});
```

V. Implementación del Módulo de Análisis de Sentimientos con Gemini Flash

V-A. Objetivo

El objetivo de este módulo es procesar texto escrito por el paciente o el profesional de la salud mental para extraer información estructurada mediante *Aspect-Based Sentiment Analysis* (ABSA), utilizando el modelo **Gemini 1.5 Flash** provisto por Google AI. El resultado se devuelve en formato JSON para su futura integración con la base de datos del sistema TerappIA.

V-B. Entorno de desarrollo

Para la implementación se utilizó el **backend** del proyecto basado en Node.js y Express, siguiendo la estructura modular de servicios y rutas. Las herramientas empleadas fueron:

- Node.js v18+
- Librería `@google/generative-ai` para la conexión con Gemini Flash.
- Librería `dotenv` para el manejo de variables de entorno.
- Express.js para la creación de rutas HTTP.
- Python 3.10+ para la ejecución del script de prueba con la librería `requests`.

V-C. Estructura de archivos

```
backend/
 routes/
    sentiment.js        # Endpoint para análisis de sentimientos
 services/
    gemini.js           # Lógica de conexión a Gemini Flash
 index.js                # Registro de rutas y servidor
 .env                    # Variable GEMINI_API_KEY
```

V-D. Configuración de credenciales

```
GEMINI_API_KEY=tu_clave_gemini
```

Este archivo se encuentra en `.gitignore` para evitar exponer credenciales.

V-E. Implementación del servicio Gemini

```
require("dotenv").config();
const { GoogleGenerativeAI } = require("@google/generative-ai");
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

async function analyzeSentiment(text, patientId) {
  const prompt = `Eres un asistente especializado en salud mental...
  Texto: ${text}`;

  const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });
  const result = await model.generateContent(prompt);
  const rawText = result.response.text();

  const jsonStart = rawText.indexOf("{");
  const jsonEnd = rawText.lastIndexOf("}");
  const jsonString = rawText.substring(jsonStart, jsonEnd + 1);

  return JSON.parse(jsonString);
}

module.exports = { analyzeSentiment };
```

V-F. Implementación de la ruta

```
const express = require("express");
const router = express.Router();
const { analyzeSentiment } = require("../services/gemini");

router.post("/", async (req, res) => {
  const { text, patient_id } = req.body;
  if (!text || !patient_id) {
    return res.status(400).json({ error: "Faltan campos requeridos" });
  }
  try {
    const analysis = await analyzeSentiment(text, patient_id);
    res.json(analysis);
  } catch (error) {
    res.status(500).json({ error: "Error procesando el an lisis", detalle: error.
       ↪ message });
  }
});

module.exports = router;
```

V-G. Registro de la ruta en el servidor

```
const sentimentRoute = require("./routes/sentiment");
app.use("/sentiment", sentimentRoute);
```

V-H. Prueba del módulo

```
import requests, json
url = "http://localhost:3001/sentiment"
payload = {
    "text": " ltimamente ␣me␣siento␣con␣mucha␣ansiedad␣y␣sin␣ganas␣de␣salir.",
    "patient_id": "paciente001"
}
response = requests.post(url, json=payload)
print(json.dumps(response.json(), indent=2, ensure_ascii=False))
```

V-I. Ejemplo de resultado real de consulta al módulo

Al ejecutar el endpoint `/sentiment` con el texto de prueba *"Últimamente me siento con mucha ansiedad y sin ganas de salir de casa."*, el sistema respondió con el siguiente JSON estructurado, generado por el modelo **Gemini 1.5 Flash**:

Listing 1. Respuesta JSON generada por Gemini 1.5 Flash para un caso de prueba

```
{
  "patient_id": "paciente_1",
  "timestamp": "2024-10-27T12:00:00Z",
  "texto_original": " ltimamente  me siento con mucha ansiedad y sin ganas de salir
     ↪ de casa.",
  "analysis": [
    {
      "aspect": "estado de  nimo ",
      "emotion": "ansiedad",
      "sentiment": "negativo",
```

```
      "confidence": 0.9
    },
    {
      "aspect": "motivaci n",
      "emotion": "apat a",
      "sentiment": "negativo",
      "confidence": 0.8
    },
    {
      "aspect": "conducta",
      "emotion": "apat a",
      "sentiment": "negativo",
      "confidence": 0.7
    }
  ],
  "risk_indicators": {
    "suicidal_ideation": false,
    "self_harm": false,
    "substance_use": false
  }
}
```

Este resultado confirma que el módulo es capaz de:

- Identificar múltiples aspectos relevantes del estado emocional.
- Clasificar emociones y polaridades.
- Asignar niveles de confianza cuantitativos.
- Detectar indicadores de riesgo relacionados con salud mental.

VI. Integración del Frontend con el Módulo de Análisis de Sentimientos

VI-A. Objetivo

El propósito de esta integración fue permitir la interacción en tiempo real con el módulo de análisis de sentimientos implementado en el backend. Para ello, se desarrolló un componente en React que permite al usuario introducir texto, enviarlo al servidor mediante una solicitud `POST` y visualizar el resultado estructurado devuelto por el modelo **Gemini 1.5 Flash**.

VI-B. Implementación

El componente `SentimentTester.jsx` fue creado dentro de la carpeta `frontend/src/` con la siguiente funcionalidad:

- Captura del texto escrito por el usuario mediante un área de texto.
- Envío de la información al endpoint `/sentiment` del backend usando la librería `axios`.
- Recepción y visualización del JSON de respuesta formateado.

Código del componente:

Listing 2. Componente SentimentTester en React

```
import { useState } from "react";
import axios from "axios";

export default function SentimentTester() {
  const [inputText, setInputText] = useState("");
  const [result, setResult] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState("");
```

```
  const analyzeText = async () => {
    setLoading(true);
    setError("");
    try {
      const response = await axios.post("http://localhost:3001/sentiment", {
        text: inputText,
        patient_id: "paciente_prueba",
      });
      setResult(response.data);
    } catch (err) {
      setError("Ocurri  un error en el an lisis");
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="max-w-2xl mx-auto p-4 space-y-4">
      <h1 className="text-2xl font-bold">An lisis de Sentimientos (Gemini)</h1>
      <textarea
        className="w-full border rounded p-2"
        rows="5"
        placeholder="Escribe aqu  el texto a analizar..."
        value={inputText}
        onChange={(e) => setInputText(e.target.value)}
      ></textarea>
      <button
        onClick={analyzeText}
        className="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600"
        disabled={loading}
      >
        {loading ? "Analizando..." : "Analizar"}
      </button>
      {error && <p className="text-red-500">{error}</p>}
      {result && (
        <div className="bg-gray-100 p-4 rounded">
          <h2 className="text-lg font-semibold mb-2">Resultado:</h2>
          <pre className="whitespace-pre-wrap">
            {JSON.stringify(result, null, 2)}
          </pre>
        </div>
      )}
    </div>
  );
}
```

Este componente fue importado y utilizado dentro de `App.jsx` para su despliegue en la interfaz principal.

VI-C. Problemas encontrados y solución aplicada

Durante la ejecución del comando `npm run dev`, se presentaron errores relacionados con la incompatibilidad de la versión 4 de `TailwindCSS` y la configuración existente del proyecto, así como la ausencia de la librería `axios`. El mensaje de error indicaba que el plugin de `PostCSS` utilizado por `TailwindCSS` había sido movido a un paquete independiente.

Para resolverlo se optó por instalar una versión estable y compatible de `TailwindCSS`:

```
npm install tailwindcss@3
```

Además, se instaló la librería `axios` para gestionar las solicitudes HTTP desde el frontend:

```
npm install axios
```

VI-D. Resultados

Tras las correcciones mencionadas, el frontend fue capaz de:

- Enviar texto al módulo de análisis de sentimientos.
- Recibir y mostrar el JSON devuelto por el backend en tiempo real.

Esta integración constituye la primera versión funcional de la interfaz de usuario para pruebas de análisis emocional en *TerappIA*.

VI-E. Evidencia visual de la ejecución

En la Figura 1 se presenta la captura de pantalla de la primera versión funcional del frontend (*MVP v1*), donde se observa la interfaz de usuario con el campo de entrada de texto, el botón de análisis y el resultado JSON devuelto por el módulo de análisis de sentimientos.

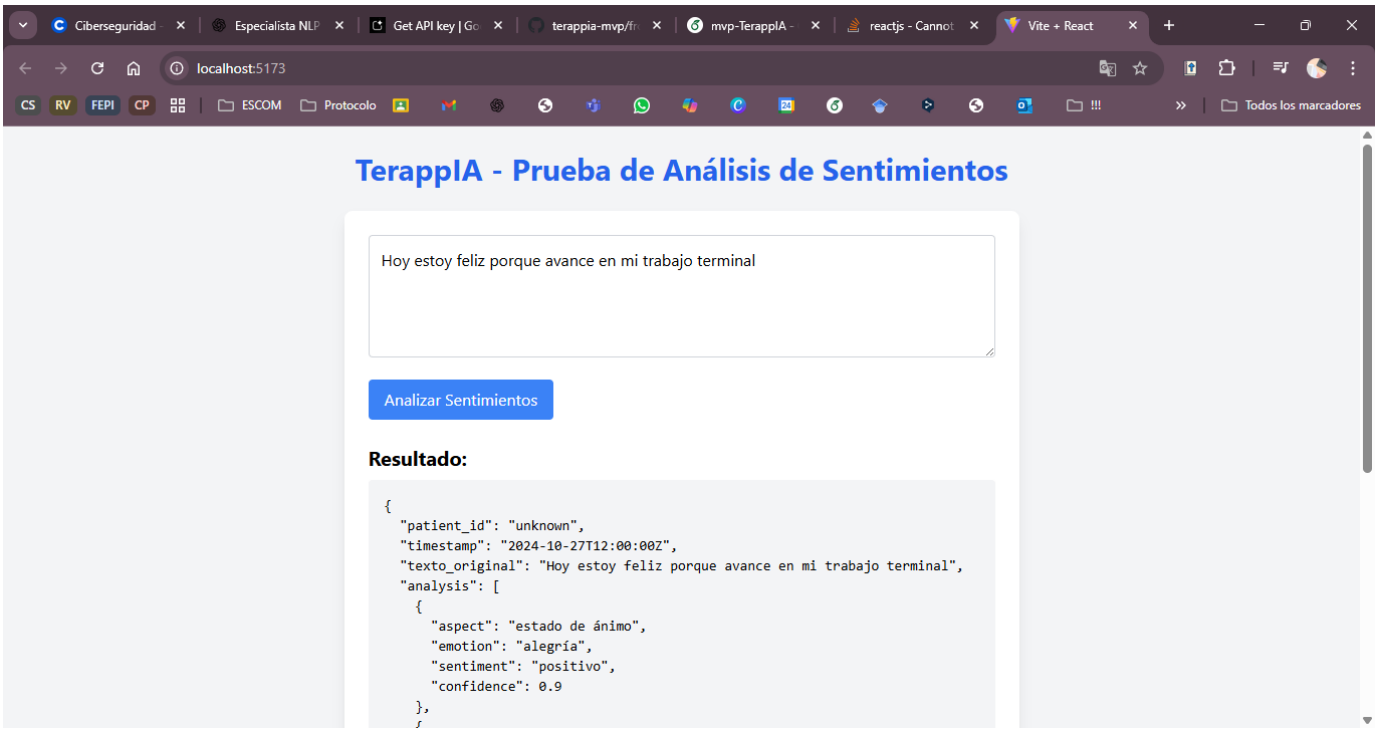

Figura 1. Ejecución del frontend conectado al módulo de análisis de sentimientos (*MVP v1*).