**Documentación del Sistema "Glosario FUPAD"**

**1. Visión General**

El sistema **Glosario FUPAD** es una aplicación web que permite consultar definiciones institucionales, sugerir nuevas palabras, solicitar definiciones faltantes y ajustar definiciones existentes. Se basa en:

* Un front-end en React que muestra una interfaz de búsqueda y gestión de términos.
* Un archivo JSON (glossary.json) como repositorio principal de definiciones.
* Una capa de servidor ligera con Google Apps Script que almacena sugerencias en una hoja de cálculo de Google Sheets, envía correos de aprobación y publica cambios al repositorio de definiciones en GitHub.

Esta documentación detalla la **arquitectura**, **componentes**, **tecnologías**, **flujo de datos** y **lógica de cada etapa** para futuras ampliaciones.

**2. Arquitectura**

graph LR

A[Usuario] --> B[React Front-end]

B --> C[Archivo glossary.json]

B --> D[Web App Google Apps Script]

D --> E[Google Sheets (Hoja "Sugerencias")]

D --> F[Correo de aprobación (MailApp)]

D --> G[GitHub API (glossary.json)]

1. **Usuario** interactúa con la UI.
2. **React** carga datos de glossary.json y envía peticiones al Web App mediante fetch.
3. **Apps Script Web App** recibe peticiones POST para registrar sugerencias en Google Sheets.
4. **Google Sheets** almacena sugerencias, y al editar cambia el estado y notifica por correo.
5. **Google Apps Script** maneja aprobaciones/rechazos vía GET, actualiza la hoja y, si es new\_term, publica el nuevo término en GitHub.

**3. Definiciones de Componentes**

|  |  |
| --- | --- |
| **Componente** | **Descripción** |
| React App | SPA que ofrece búsqueda y formularios. |
| glossary.json | Base de datos local de términos y definiciones. |
| App.css | Estilos ajustados para visualización. |
| Google Apps Script | Lógica de backend para sugerencias, aprobaciones y sincronización con GitHub. |
| Google Sheets | Almacena solicitudes: new\_term, adjust\_term, request\_definition. |
| MailApp | Servicio de Apps Script para envíos de correo de aprobación. |
| GitHub API | Permite leer y escribir el glossary.json remoto. |

**4. Tecnologías Aplicadas**

* **Frontend:**
  + React (v17+)
  + JavaScript (ES6+)
  + CSS personalizado con variables y responsive design
  + fetch API para llamadas HTTP
* **Backend (ligero):**
  + Google Apps Script (JavaScript 1.6)
  + MailApp (envío de correos)
  + UrlFetchApp (llamadas REST a GitHub API)
* **Persistencia:**
  + Google Sheets (almacén de sugerencias)
  + GitHub repository (almacén maestro de glossary.json)

**5. Detalle de Construcciones y Lógica**

**5.1 Front-end (App.js)**

1. **Estados y hooks principales:**
   * searchTerm, results, selected para búsqueda y selección.
   * showAddForm, showRequestDefinition, showAdjustmentForm controlan visibilidad de formularios.
   * Formularios usan nuevoTermino, nuevaDefinicion, nuevaFuente, adjustComment.
2. **Carga de datos:**
   * import glossaryData from './data/glossary.json'
   * Filtro en useEffect al cambiar searchTerm.
3. **Interacción:**
   * Al escribir en el input, actualiza searchTerm.
   * Si hay coincidencias, muestra lista de términos (solo títulos).
   * Al hacer click en un término, muestra cuadro con definición detallada y botón “Solicitar Ajuste”.
4. **Formularios:**
   * **Nueva palabra:** campo término/definición/fuente -> POST al Web App.
   * **Solicitud de definición:** POST con tipo request\_definition.
   * **Ajuste:** envía adjust\_term y comentario.

**5.2 Estilos (App.css)**

* Variables CSS para colores, radios y transiciones.
* Reset base y box-sizing.
* Layout centrado y responsive.
* Estilos para header sticky, search, sugerencias, formularios y botones.
* Justificación de texto en cuadros de definición y “definición actual”.
* Inclusión de icono de lupa junto a títulos de ítems.

**5.3 Backend (Google Apps Script)**

**5.3.1 doPost(e)**

* Parse de e.postData.contents JSON.
* payload.values es un array: [id, tipo, termino, definicion, sugerencia, fuente, fecha, estado].
* Se anexan a la hoja Sugerencias y a la hoja Debug para logging.
* Responde { success: true }.

**5.3.2 onEdit(e)**

* Trigger activado al editar la hoja.
* Si el estado cambió de "Pendiente" a otro distinto, envía correo de aprobación/rechazo.
* Usa MailApp.sendEmail con enlaces a la Web App (?action=approve o ?action=reject).

**5.3.3 doGet(e)**

* Maneja acciones de aprobación/rechazo desde links en el correo.
* Extrae action y id, busca fila correspondiente en la hoja.
* Si new\_term y action=approve:
  + Verifica duplicados en GitHub (validarDuplicadoGitHub).
  + Si no existe, \_pushToGitHub(entry) para agregar al JSON remoto.
  + Marca la fila con “Aprobado” (o “Duplicado”).
* Si adjust\_term y action=approve:
  + Marca la fila “Ajuste aprobado”.
  + (Opcional) Invoca \_updateOnGitHub(term, newDefinition) para sobrescribir la definición.
* action=reject marca la fila “Rechazado”.

**5.3.4 Integración con GitHub**

* **\_pushToGitHub(entry):**
  1. GET del JSON remoto.
  2. Decodifica base64 y parsea a array.
  3. arr.push(entry), codifica en base64, hace PUT con message y sha.
* **validarDuplicadoGitHub(term):** devuelve true si term ya existe (case-insensitive).
* **\_updateOnGitHub(term, newDef):** mapea el array, reemplaza item.definition, publica.

**6. Flujo de Datos**

1. **Usuario** escribe en search → React filtra localmente el JSON.
2. **Si no existe** término → opciones para “Enviar” o “Solicitar Definición” → fetch POST.
3. **Web App** registra sugerencia en Sheets → usuario recibe alerta.
4. **Editor** revisa la hoja → cambia estado “Pendiente” → onEdit envía correo.
5. **Editor** hace click en link de correo → doGet procesa acción y actualiza:
   * en Sheets (estado)
   * en GitHub (nuevo término o ajuste)
6. **Nuevo JSON** ya disponible en siguiente deploy (estático o integrado) o se recupera al refrescar React.

**7. Despliegue y Configuración**

1. **React App:** desplegar en Netlify, Vercel o similar.
2. **Google Apps Script:** desplegar como Web App (ejecutar como "Yo" y permitir acceso "Anyone, even anonymous").
3. **Google Sheets:** crear hoja Sugerencias con columnas A–H: id, tipo, termino, definicion, sugerencia, fuente, fecha, estado.
4. **Token GitHub:** configurar GH\_TOKEN en PropertiesService.
5. **Correo del destinatario:** ajustar constante DESTINATARIO.