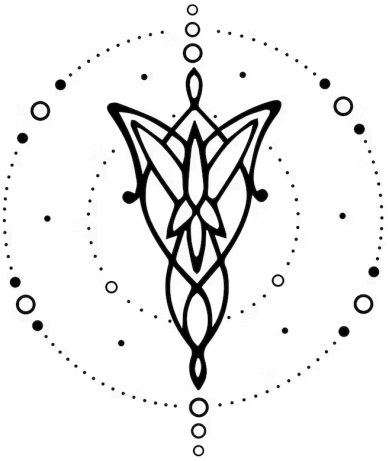

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Juan Diego	Gago	64137	jgago@itba.edu.ar
Octavio	Zacagnino	64255	ozacagnino@itba.edu.ar
Tomas	Pinausig	63167	tpinausigcastillo@itba.edu.ar
Javier	Peral	62108	jperalbelmont@itba.edu.ar

2. Repositorio

https://github.com/pi-na/tp-tla

3. Dominio

El objetivo de este proyecto es desarrollar un lenguaje y sistema que permita generar páginas web en HTML a partir de archivos JSON estructurados. El lenguaje JSON actuará como una representación de alto nivel de la estructura y contenido de la página web, permitiendo al usuario definir elementos como títulos, subtítulos, secciones, navegación y estilos básicos sin necesidad de escribir directamente código HTML.

El sistema interpretará estos archivos JSON y generará automáticamente código HTML válido y semánticamente correcto, que podrá ser visualizado en cualquier navegador moderno. Esto permitirá a diseñadores, programadores principiantes o sistemas automatizados construir páginas web de forma rápida, segura y legible.

Para reducir la complejidad del proyecto, el lenguaje JSON utilizará una estructura jerárquica con un conjunto limitado de etiquetas HTML representadas mediante el atributo `@type`. Cada elemento podrá contener atributos adicionales como `@content` (para anidar contenido), `src`, `href`, `align`, `BGCOLOR`, entre otros. No se incluirá soporte para CSS externo, JavaScript, ni atributos avanzados. El sistema solo procesará etiquetas HTML básicas como `html`, `head`, `title`, `body`, `p`, `a`, `img`, `b`, `i`, `br`, `hr`, `center`, y podrá ser extendido en futuras versiones. Cada objeto JSON deberá respetar una estructura estricta para garantizar una conversión válida y predecible a HTML.

4. Construcciones

El lenguaje JSON-HTML está diseñado para transformar una estructura JSON predefinida en un documento HTML completamente funcional. Para ello, se definen las siguientes construcciones, prestaciones y funcionalidades:

(I). Estructura Básica del Documento:

- El documento de entrada deberá iniciarse con un objeto JSON que contenga la clave "**@type**" con el valor "**html**", estableciendo la raíz de la estructura HTML.
- Se deberán definir, al menos, los elementos **head** y **body** como partes fundamentales del documento.
- La estructura podrá anidar otros elementos hasta un máximo de 4 niveles de profundidad.

(II). Definición de elementos HTML y atributos:

- Los elementos HTML se especificarán mediante la clave "**@type**", permitiendo la creación de elementos estándar (e.g., *div*, *p*, *h1*, *img*, *a*, etc.) y elementos

personalizados.

- Se podrán asignar atributos a cada elemento (por ejemplo, *style*, *class*, *id*, *src*, *href*) para definir propiedades visuales y de comportamiento.

- Los elementos podrán contener texto o a otros elementos, utilizando la clave "**@content**", pudiendo mezclarse contenido literal con objetos JSON que representen subelementos.

(III). Estructuras de control y Dinámicas:

- Se implementarán estructuras condicionales del tipo **IF-THEN-ELSE** para incluir o excluir elementos HTML según condiciones definidas en el JSON.

- Se proveerán mecanismos iterativos, como **FOR** o **WHILE**, para generar de forma dinámica conjuntos de elementos (por ejemplo, listas o tablas) a partir de datos suministrados.

- Se admitirán expresiones aritméticas y lógicas que permitan calcular valores para atributos o determinar la estructura del documento.

(IV). Parámetros y Variables:

- Se posibilitará la definición y uso de variables dentro del documento JSON para reutilizar valores (como colores, textos o direcciones) en distintas partes del HTML.

- El lenguaje permitirá incorporar parámetros de entrada, ya sea desde la entrada estándar o mediante argumentos del programa, para personalizar el contenido generado.

(V). Validación y Restricciones del Lenguaje:

- **Obligatoriedad de la clave "@type":** Cada elemento debe incluir la clave "**@type**" para identificar el tipo de etiqueta HTML a generar.

- **Unicidad de claves críticas:** No se permitirán duplicados de la clave "**@content**" en un mismo objeto, a fin de evitar ambigüedades en la interpretación.

- **Operadores y expresiones:** Se deberán utilizar únicamente operadores y funciones definidos en el lenguaje; se rechazará cualquier operador o expresión que no esté explícitamente contemplado.

- **Objetos JSON vacíos:** Se rechazará el uso de objetos vacíos, ya que pueden comprometer la correcta transformación a HTML.

- **Referencias a variables:** Todas las variables referenciadas deberán haber sido definidas previamente; de lo contrario, el programa será rechazado.

(VI). Transformación y Generación de Salida:

- El compilador se encargará de transformar el documento JSON en un código HTML válido, respetando la estructura y los atributos definidos en el mismo.

- Se garantizará la integración de estilos embebidos (CSS) y la generación de estructuras interactivas según lo definido en el JSON.

5. Casos de Prueba

Se proponen los siguientes casos iniciales (luego serán más) de prueba de **`aceptación`**:

(I). Un programa que genere una página HTML con un `title` y un párrafo simple.

(II). Un programa que genere una página HTML con una imagen centrada y un fondo de color personalizado en el `body`.

(III). Un programa que incluya un enlace (`<a>`) dentro de un párrafo.

(IV). Un programa que genere una estructura anidada de texto con negrita (`<b>`) e itálica (`<i>`).

(V). Un programa que utilice una separación visual (`<hr>`) seguida de múltiples líneas de texto con saltos de línea (`<br>`).

(VI). Un programa que combine distintos elementos como título, imagen, enlace y texto enriquecido dentro de un mismo documento.

(VII). Un programa que incluya elementos obsoletos como `center` y `align`, y se asegure de que se procesen correctamente.

(VIII). Un programa que genere múltiples secciones dentro del `body` (simulando contenedores como `divs`) con contenido variado.

(IX). Un programa que utilice atributos adicionales como `src`, `href`, `BGCOLOR`, y valide que se inserten correctamente en el HTML.

(X). Crear un HTML utilizando 1 variable provista por el usuario en tiempo de compilación

Además, los siguientes casos de prueba de **`rechazo`**:

(I). JSON malformado sintácticamente.

(II). Falta del atributo obligatorio `@type`.

(III). Tipo de elemento no soportado, ej { "@type": "video", "@content": "" }

(IV). Contenido con tipo de dato incompatible.

(V). Referencia a una variable no definida en el contexto.

6. Ejemplos

Ejemplo de pasar de un JSON a un HTML básico donde está como debería verse el HTML resultante.

```
{
  "@type": "html",
  "@content": [
    {
      "@type": "head",
      "@content": {
        "@type": "title",básico
        "@content": "Imagen centrada"
      }
    },
    {
      "@type": "body",
      "BGCOLOR": "#e0f7fa",
      "@content": {
        "@type": "center",
        "@content": {
          "@type": "img",
          "src": "imagen.jpg",
          "alt": "Descripción de la imagen",
          "@content": ""
        }
      }
    }
  ]
}
```

```
Html:
<!DOCTYPE html>
<html>
 <head>
  <title>Imagen centrada</title>
 </head>
 <body bgcolor="#e0f7fa">
  <center>
   <img src="imagen.jpg" alt="Descripción de la imagen">
```

```
      </center>
    </body>
  </html>
```

```
{
  "@type": "html",
  "@content": [
    {
      "@type": "head",
      "@content": {
        "@type": "title",
        "@content": "Bienvenidos a Mi Sitio"
      }
    },
    {
      "@type": "body",
      "BGCOLOR": "#f0f0f0",
      "color": "#333",
      "@content": [
        {
          "@type": "h1",
          "@content": {
            "var": "userName"
          }
        },
        {
          "@type": "p",
          "@content": "Este es un párrafo generado desde JSON"
        }
      ]
    }
  ]
}
```

En tiempo de compilacion se debe agregar el userName por entrada estandar

```
<!DOCTYPE html>
<html>
 <head>
  <title>Bienvenidos a Mi Sitio</title>
 </head>
```

```
  <body BGCOLOR="#f0f0f0" style="color: #333;">
    <h1>Juan Diego</h1>
    <p>Este es un párrafo generado desde JSON, con un poco más de onda. 🎉</p>
  </body>
</html>
```