Sitio Web Fundación Procamsu (Sumapaz, Bogotá)

Arquitectura de Software Pagina Web de Bajo Costo

Autor: Santiago Castro Zuluaga

Pontificia Universidad Javeriana

2025

Índice

Arquitectura de Software – Sitio Web Fundación PROCAMSU Sumapaz

1 Objetivo del Sitio Web y Propósito Institucional

El objetivo principal del sitio web es difundir la misión y las actividades de la Fundación PROCAMSU de Sumapaz de manera moderna y accesible. Este sitio institucional servirá como ventana de información sobre la historia de la fundación, su finalidad social, proyectos en curso y formas de contacto para la comunidad.

De esta manera, se busca fortalecer la presencia en línea de la fundación, facilitando que tanto la comunidad local como posibles colaboradores o donantes conozcan su labor. El sitio presentará contenido claro y organizado sobre:

- **Quiénes son** ("Nosotros"),
- **Qué hacen** (sus proyectos actuales),
- **Qué productos ofrecen** (para recaudar fondos o promover la economía local),
- **Cómo el público puede contactarlos o apoyar**.

Adicionalmente, al incluir un listado de productos con opción de carrito de compras y contacto por WhatsApp, el sitio pretende fomentar la participación y generar ingresos (por ejemplo, vendiendo productos artesanales o agroecológicos relacionados con sus proyectos).

En resumen, el sitio web institucional se enfocará en exponer la cultura organizacional de la fundación, ganando la confianza de los visitantes y alentando su involucramiento con la causa.

Al ser un sitio estático sencillo (HTML, CSS, JS) alojado en GitHub Pages, se garantiza que la fundación pueda mantener los contenidos fácilmente y a bajo costo, sin depender de infraestructura compleja. Esto es importante para una organización sin ánimo de lucro, ya que maximiza la transparencia y minimiza gastos, permitiendo que el equipo de la fundación pueda actualizar texto e imágenes (por ejemplo, agregar nuevos proyectos o noticias) con relativa facilidad.

En cuanto a la estética, el sitio buscará un diseño moderno, limpio y elegante, alineado con la identidad de la fundación, para causar una buena impresión y transmitir profesionalismo.

También se pondrá énfasis en la usabilidad: que la navegación sea intuitiva para distintos perfiles de usuario (miembros de la comunidad, investigadores, posibles voluntarios o financiadores), asegurando que todos encuentren rápidamente la información deseada.

2 Estructura de Navegación del Sitio

La estructura de navegación se basará en un menú principal presente en todas las páginas, que enlaza a las secciones clave: Inicio, Nosotros, Productos, Proyectos, Contáctanos, y el Carrito de Compras (este último posiblemente representado con un ícono de carrito en la interfaz). A continuación se describen cada una de las páginas y su contenido previsto:

Inicio: Página principal con una bienvenida y resumen de la información más destacada. Podría incluir un banner con imágenes representativas de Sumapaz o de las actividades de la fundación, una breve misión o slogan, y accesos rápidos a secciones importantes. También es buen lugar para destacar noticias recientes, proyectos destacados o productos promocionados.

Nosotros: Sección informativa sobre la fundación. Incluirá subsecciones o apartados para la Historia (reseña histórica de la fundación), la Finalidad/Misión (objetivos institucionales y valores) y la Ubicación (información geográfica, posiblemente con un mapa). Esta página dará contexto al visitante sobre quiénes son y qué hacen, fortaleciendo la confianza en la institución.

Historia: Orígenes de la fundación, hitos importantes desde su creación.

Finalidad: Misión, visión y objetivos institucionales; a quién beneficia la fundación y de qué manera.

Ubicación: Dónde opera la fundación (región de Sumapaz), con dirección de su sede o áreas de influencia. Se puede incluir un mapa embebido o imagen de mapa.

Productos: Catálogo de productos o servicios que la fundación ofrece al público (por ejemplo, artesanías elaboradas en proyectos comunitarios, productos agrícolas, souvenirs, etc.). Esta página mostrará un listado dinámico de productos con sus imágenes, descripciones y precios. El listado será dinámico en el sentido de que los productos podrán administrarse fácilmente (añadir, quitar o modificar) sin reescribir todo el HTML; por ejemplo, cargando los datos de un archivo JSON o script JS. Cada producto tendrá un botón de "Agregar al carrito" que permite seleccionar artículos para compra.

Carrito de Compras: Página que muestra los productos que el usuario ha agregado desde la sección Productos. Aquí se listarán los ítems seleccionados, sus cantidades y el total estimado. Dado que el sitio no contará con procesamiento de pagos en línea ni backend propio, el flujo de compra continuará externamente: en esta página se ofrecerá un botón o enlace para "Finalizar compra por WhatsApp". Al hacer clic, el sitio generará un mensaje con el detalle del pedido y redirigirá al usuario a una conversación de WhatsApp con la fundación, para coordinar el pago y entrega (ver más en la sección de Carrito e integración con WhatsApp).

Proyectos: Sección que presenta los proyectos actuales y actividades de la fundación. Puede mostrarse como un listado o grid de proyectos con una imagen representativa y breve descripción, permitiendo al usuario clicar para ver más detalles de cada proyecto (ya sea en la misma página con desplegables o en subpáginas si se prefiere). Aquí la fundación podrá mostrar el impacto de su trabajo, las comunidades beneficiadas, fotos de eventos, etc. Mantener

esta sección actualizada es clave para demostrar transparencia y vitalidad institucional.

Contáctanos: Página con la información de contacto de la fundación. Incluirá dirección física, correo electrónico, teléfonos de contacto y, posiblemente, enlaces a redes sociales. Además, puede integrarse un formulario de contacto sencillo (nombre, email, mensaje) aunque, al no haber backend, su funcionalidad sería limitada – por ejemplo, se podría usar un servicio de terceros o simplemente proporcionar la dirección de correo para que el usuario envíe su consulta manualmente. Dado que el sitio ofrece contacto vía WhatsApp para compras, aquí igualmente se podría mencionar el número de WhatsApp institucional. En todo caso, esta sección busca facilitar al usuario diferentes vías para comunicarse con la fundación (teléfono, email, WhatsApp, redes sociales).

Toda la navegación estará diseñada para ser clara y consistente en cada página. Se usará un menú responsive (adaptado a móviles) de manera que en pantallas pequeñas se podría colapsar en un menú tipo "hamburguesa". Además, el pie de página del sitio repetirá algunos de los enlaces de contacto e incluirá, por accesibilidad, un link al mapa del sitio.

3 Soporte de Internacionalización (Sitio Bilingüe ES/EN)

El sitio tendrá soporte multi-idioma, con idioma principal el español (orientado a la comunidad local de Sumapaz) y la opción de ver el contenido en inglés (para audiencia internacional o potenciales donantes extranjeros). La estrategia de internacionalización se puede abordar de la siguiente manera:

Selección de idioma: En la interfaz se incluirá un selector (por ejemplo, un ícono o texto "ES/EN") que permita al usuario cambiar de idioma en cualquier página. Al hacer clic, el sitio mostrará la versión traducida.

Estructura de contenidos bilingües: Inicialmente, se optará por mantener dos versiones de cada página – por ejemplo, utilizando directorios separados para cada idioma. Una posible estructura sería tener un directorio /en/ que contenga copias en inglés de los archivos HTML (por ejemplo en/index.html, en/nosotros.html, etc.), mientras que la versión en español reside en la raíz o en /es/. De este modo, al cambiar de idioma el usuario sería redirigido a la URL equivalente en el otro idioma (por ejemplo, de /productos.html a /en/productos.html para la versión en inglés). Esta aproximación evita tener que programar carga dinámica de textos en esta etapa inicial, y facilita las modificaciones, aunque sí implica mantener contenido duplicado en dos idiomas.

Contenido traducido: Se procurará que toda la información esencial esté disponible en ambos idiomas. Esto incluye menús, títulos, textos de cada sección, descripciones de productos y proyectos, y etiquetas de la interfaz (ej. el botón "Agregar al carrito" tendría su equivalente "Add to cart" en la versión en inglés). La internacionalización abarcará también atributos importantes para accesibilidad/SEO, como el atributo lang en el HTML (por ejemplo, `<html lang=.es»` en la versión española y `<html lang=.en»` en la inglesa, para indicar correctamente el idioma del contenido a navegadores y buscadores).

Alternativa con JavaScript: A futuro, si se desea evitar duplicar páginas, se podría implementar un mecanismo de carga de textos multilenguaje con JavaScript. Por ejemplo, tener archivos de recursos (JSON o JS) con cadenas traducidas para cada idioma y una función que sustituya el texto de la página según el idioma seleccionado. Esto permitiría agregar nuevos idiomas de forma más escalable sin repetir estructura HTML. Sin embargo, dado que el requisito actual es "sin frameworks externos por ahora", mantener páginas estáticas separadas en cada idioma puede ser la solución más sencilla de corto plazo.

En resumen, el sitio será bilingüe de forma que pueda atender tanto a la audiencia local (español) como a interesados internacionales (inglés), manteniendo una experiencia de usuario consistente en ambos idiomas. Se documentará claramente cómo agregar o actualizar contenido en cada versión para facilitar futuras modificaciones por parte del equipo de la fundación.

4 Uso de HTML, CSS y JavaScript (sin Frameworks)

El desarrollo del sitio se realizará utilizando HTML, CSS y JavaScript puro, sin frameworks de frontend (como React, Angular, Vue) ni bibliotecas pesadas adicionales. Esta elección se debe a que el sitio es relativamente sencillo en términos de interacción, y utilizar tecnologías nativas garantiza mejor rendimiento, menor complejidad y mayor control sobre el código. Algunas consideraciones al respecto:

HTML5 semántico: Se estructurará el contenido usando etiquetas semánticas adecuadas, lo que mejora tanto la accesibilidad como el SEO. Cada página tendrá un único `<h1>` representando su título principal (ej. "Fundación PROCAMSU Sumapaz" en Inicio, "Nosotros" en la sección correspondiente, etc.), y subtítulos con `<h2>`, `<h3>` para estructurar jerárquicamente la información.

CSS3 y diseño responsivo: Se creará una hoja de estilo CSS personalizada para reflejar la identidad visual de la fundación (colores institucionales, tipografías legibles, etc.). Se evitará por ahora el uso de frameworks CSS tipo Bootstrap para mantener el control total del estilo y no añadir código no utilizado; en su lugar, se implementarán manualmente los componentes necesarios (menú, grids, botones, etc.). El diseño será responsive, usando consultas de medios (media queries) para ajustar el layout en dispositivos móviles, tabletas y monitores grandes. Por ejemplo, el menú pasará de horizontal a un menú móvil desplegable en pantallas pequeñas. El look & feel será moderno pero simple, priorizando la usabilidad y tiempos de carga rápidos.

JavaScript vanilla: El código JS se empleará para las interacciones dinámicas necesarias: p. ej., manejo del carrito de compras (agregar/quitar ítems y almacenamiento temporal de los mismos), toggling de idioma (si se implementa vía JS), y quizás efectos menores de UI (como un carrusel de imágenes en Inicio, o desplegar/ocultar secciones de proyectos). Dado que no se usan frameworks, el código JavaScript será escrito con buenas prácticas de modularidad: se podrían definir objetos o funciones globales dentro de un único namespace para evitar colisiones, o simplemente agrupar la lógica por página en archivos separados. Se aprovechará

la API nativa del DOM para manipular elementos (por ejemplo, generar la lista de productos en la página Productos de forma dinámica).

Sin backend (sitio 100 % estático): Es importante destacar que, al no haber framework ni backend, toda la lógica ocurre del lado del cliente. Por ejemplo, si hubiera un formulario de contacto, su envío tendría que realizarse usando un servicio externo o `mailto` (pues no hay PHP/Node para procesarlo). En general, se diseñará el sitio de forma que no dependa de funcionalidades de servidor. Esto limita ciertas características (como autenticación de usuarios, procesamiento de pagos en línea, bases de datos, etc.), pero a su vez simplifica la implementación y mejora la seguridad (no hay vectores de ataque como SQL injection, etc. al no haber base de datos).

Librerías externas puntuales: Aunque no se usarán frameworks grandes, se puede evaluar el uso de alguna librería ligera si agrega valor y no rompe la filosofía de mantener el sitio simple. Por ejemplo, una librería para carruseles de imágenes o para validación de formularios podría emplearse más adelante, pero inicialmente se priorizará código propio minimalista. No se cargarán librerías innecesarias que aumenten el peso de la página; la intención es que la web cargue rápido incluso en conexiones lentas, aprovechando las ventajas de un sitio estático.

En síntesis, la arquitectura tecnológica se apoya en los estándares web básicos, lo que permite que el sitio funcione en cualquier navegador moderno sin requerir instalaciones especiales. Esta simplicidad facilita la mantenibilidad y la incorporación de futuros programadores que lean el código, ya que todo estará escrito en lenguajes web universales, con una estructura clara y comentada cuando sea necesario.

5 Mapa del Sitio – Diagrama de Navegación

Diagrama de navegación (mapa del sitio) propuesto para la Fundación PROCAMSU Sumapaz, mostrando las páginas principales del sitio y sus relaciones jerárquicas (Inicio, secciones informativas, catálogo de Productos, Carrito de Compras, Proyectos y Contáctanos), incluyendo las subsecciones internas de "Nosotros" (Historia, Finalidad, Ubicación). Este mapa ilustra la arquitectura de información del sitio web, facilitando la comprensión de cómo están organizados los contenidos y cómo puede desplazarse el usuario a través de ellos.

El Inicio actúa como eje central desde el cual se puede acceder a todas las demás secciones principales. Bajo la página de Nosotros, se observan sus tres secciones internas, reflejando la estructura lógica de ese contenido dentro de una sola página. Este tipo de representación gráfica permite identificar si la navegación será clara y equilibrada (por ejemplo, se ve que todas las páginas principales están al mismo nivel desde Inicio, evitando una estructura demasiado profunda que pueda perder al usuario).

En resumen, el diagrama de navegación confirma que el sitio tendrá una estructura simple y jerárquica, donde cada sección es fácilmente accesible y cumple una función específica en el

conjunto del portal. Esto ayudará a garantizar una experiencia de usuario positiva, ya que podrán encontrar rápidamente la información institucional, productos o formas de contacto que necesiten.

Nota: La implementación del menú reflejará este mapa: por ejemplo, al situarse sobre "Nosotros" en el menú, podría desplegar anclas para ir directamente a Historia/Finalidad/Ubicación dentro de esa página, o simplemente scroll dentro de "Nosotros". Para "Productos" y "Proyectos", es posible que en el futuro se tengan subpáginas si el contenido crece, pero inicialmente se maneja todo en una sola página cada uno.

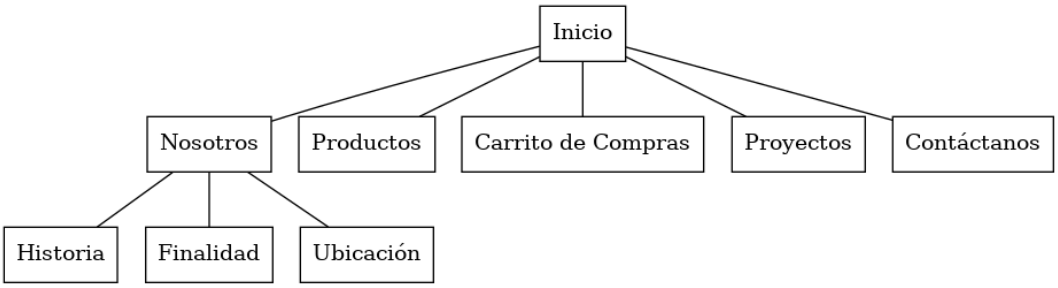

Figura 1: Diagrama de navegación del sitio web de la Fundación PROCAMSU Sumapaz.

6 Estructura de Carpetas y Componentes del Proyecto

Una buena organización de archivos es fundamental para la escalabilidad y mantenimiento del sitio. A continuación se detalla la estructura de directorios y archivos del proyecto web (repositorio GitHub Pages), siguiendo convenciones comunes en proyectos web estáticos:

Raíz del repositorio: Contendrá los archivos HTML de cada página principal del sitio – por ejemplo: index.html (Inicio), nosotros.html, productos.html, carrito.html, proyectos.html y contacto.html. Estos son los documentos HTML que se servirán al navegador para cada sección principal. Adicionalmente, en la raíz pueden existir archivos de configuración para GitHub Pages o complementarios, como un CNAME (si se usa dominio personalizado), o el archivo especial `.nojekyll` (para indicarle a GitHub Pages que no procese el sitio con Jekyll, dado que usamos HTML puro).

Directorio css/: Contiene las hojas de estilo CSS del sitio. Inicialmente, se prevé una hoja principal `styles.css` donde se incluirán todos los estilos globales del sitio (tipografías, colores, layout base, estilos de cabecera y pie, etc.). Si el CSS crece, se podría modularizar en varios archivos (por ejemplo, `home.css`, `productos.css` para estilos específicos), pero en principio un archivo consolidado es manejable. Esta separación de un directorio para estilos sigue las buenas prácticas, manteniendo el CSS independiente de la estructura HTML.

Directorio js/: Incluye los archivos JavaScript. Aquí se organiza la lógica por funcionalidad o por página. Por ejemplo: un archivo `main.js` para funciones comunes (como el código para el menú o la conmutación de idioma), `productos.js` para la lógica de carga de la lista de productos y manejo del botón "Agregar al carrito", `carrito.js` para gestionar la página del

carrito (leer los productos almacenados, calcular totales, armar el mensaje de WhatsApp) y quizás un archivo `i18n.js` en caso de implementar la carga dinámica de textos multilingües. Mantener los scripts en una carpeta dedicada facilita la ubicación del código y colabora con la separación de responsabilidades del proyecto.

Directorio assets/ (o images/): Reservado para recursos estáticos como imágenes, logos, archivos multimedia descargables, etc. Por ejemplo, aquí residiría `logo.png` (logo de la fundación) y demás imágenes utilizadas en el sitio (fotos de los proyectos, iconografía, etc.). De esta manera, las rutas de las imágenes en el HTML/CSS serán relativas a esta carpeta (e.g., `<img src=.ªssets/logo.png.ªlt="Logo»`). Mantener las imágenes en su propia carpeta ayuda a que la estructura sea modular y clara.

Directorio data/: (Opcional) Se puede incluir una carpeta de datos para almacenar archivos JSON u otros formatos que provean contenido al sitio. Un caso claro es un archivo `productos.json` que contenga un arreglo de productos (con sus nombres, precios, descripciones, URL de imagen, etc.). El script `productos.js` podría hacer `fetch` de este JSON para generar el listado dinámicamente. Esto permite que la actualización de productos se haga editando un archivo de datos en vez de tocar el HTML, lo cual es más limpio y reducirá errores. Aunque no es estrictamente necesario (pues los datos se podrían escribir dentro del JS mismo), separar la capa de datos es útil si se piensa en futuras integraciones (por ejemplo, reemplazar ese JSON por una API real más adelante).

Propuesta de estructura de directorios y archivos del proyecto (sitio estático en GitHub Pages), siguiendo una organización por separación de responsabilidades: archivos HTML en la raíz, carpetas dedicadas para CSS, JS, imágenes/assets y datos. La estructura sugerida (resumida en la imagen) refleja un modelo sencillo típico de proyectos web estáticos. Esta organización facilita la navegación del código: por ejemplo, cualquier desarrollador que se una al proyecto sabrá inmediatamente dónde buscar los estilos (/css), los scripts (/js) o recursos multimedia (/assets). Además, soporta la escalabilidad; a medida que el sitio crezca, se pueden añadir nuevos archivos en la estructura correspondiente (por ejemplo, si en el futuro se agrega una página "Noticias", se crearía `noticias.html` en raíz, y sus estilos y scripts asociados podrían residir en las carpetas ya existentes). En cuanto a control de versiones, tener los archivos bien organizados ayuda a identificar fácilmente qué partes del sitio se modifican en cada commit. En definitiva, mantener un orden lógico de carpetas y nombres descriptivos contribuye tanto a la mantenibilidad como a la colaboración en equipo.

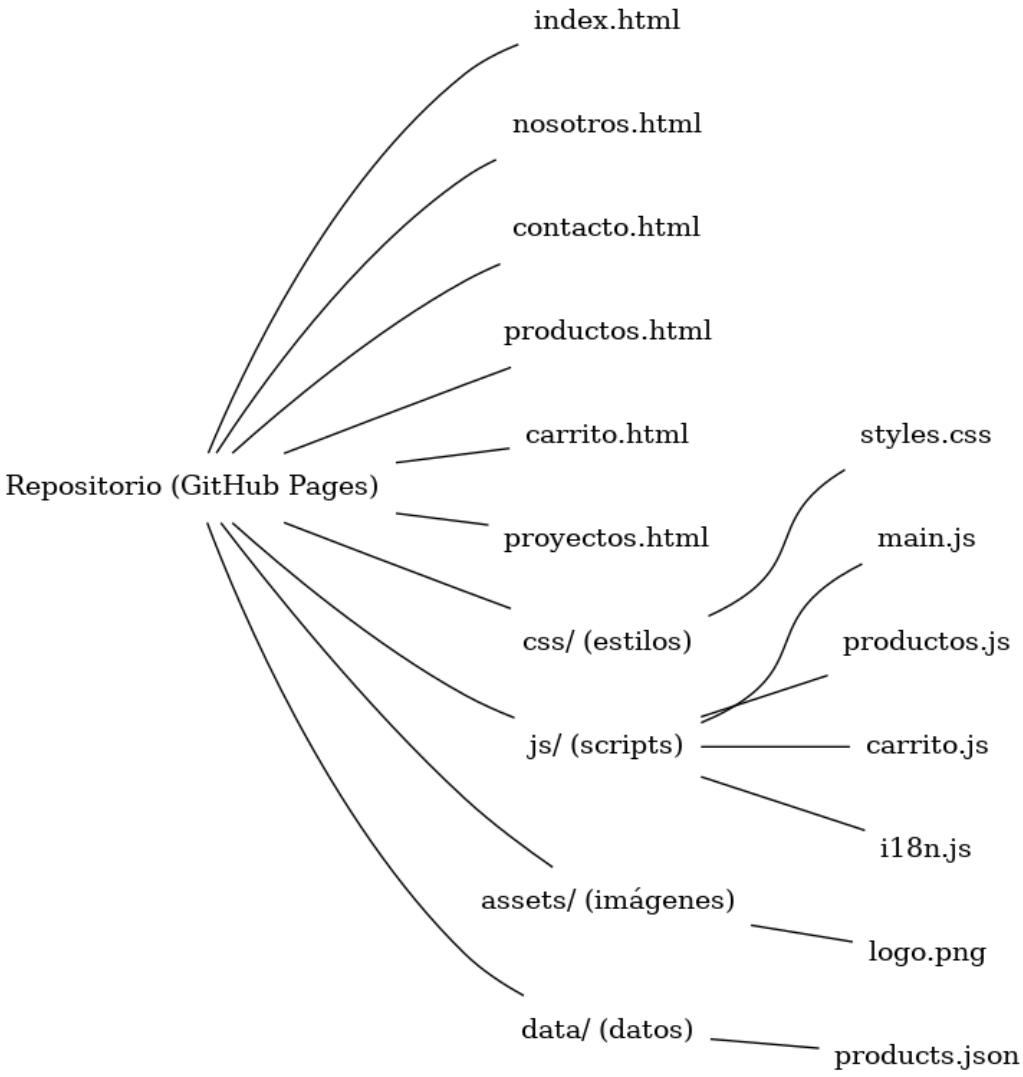

Figura 2: Estructura de carpetas y componentes del proyecto web de la Fundación PROCAMSU Sumapaz.

7 Flujo del Carrito de Compras e Integración con WhatsApp

Una característica importante del sitio es el carrito de compras y su integración con WhatsApp para completar los pedidos. A continuación, se describe el flujo previsto paso a paso, tanto desde la perspectiva del usuario como la técnica subyacente:

Diagrama de flujo de la interacción de carrito de compras e integración con WhatsApp: el usuario navega por los productos, agrega artículos al carrito, revisa el carrito y finalmente envía el pedido mediante un mensaje de WhatsApp prellenado.

El proceso inicia cuando el usuario visita la página de Productos y navega por el catálogo. Al

encontrar un producto de interés, hace clic en el botón "Agregar al carrito", lo cual ejecuta un script JavaScript que almacena la selección en el carrito. Dado que no hay backend, este almacenamiento se realiza en el propio navegador – por ejemplo, utilizando `LocalStorage` o una variable global en memoria.

LocalStorage permite guardar una lista de ítems de forma persistente (hasta que se vacíe el carrito o se cierre la sesión), de modo que aunque el usuario navegue a otra página (ej. vaya al carrito), los datos de qué productos seleccionó permanecen disponibles. Cada vez que el usuario agrega un producto, el sitio podría mostrar alguna indicación (como actualizar un contador de ítems en el ícono del carrito).

Cuando el usuario decide ver su carrito, hace clic en el enlace/ícono del Carrito de Compras, accediendo a la página de carrito (`carrito.html`). Al cargarse esta página, un script `carrito.js` recupera los ítems almacenados (p. ej., leyendo de `LocalStorage` la lista de productos seleccionados) y genera dinámicamente el listado: mostrando nombre, cantidad, precio unitario y subtotal por producto, y un total acumulado. El usuario puede revisar esta información e incluso modificar cantidades o eliminar productos (estas acciones serían manejadas también por JS actualizando los datos almacenados).

Finalmente, para realizar el pedido, el usuario hace clic en el botón "Enviar pedido por WhatsApp". Aquí ocurre la integración clave: el sitio construye un enlace especial de WhatsApp con el contenido del carrito. Usando la funcionalidad de *click-to-chat* de WhatsApp, es posible crear un URL que abra una conversación con un número específico (el de la fundación) y además pre-cargue un texto en el campo de mensaje.

Por ejemplo, se utilizará una URL del estilo:

`https://wa.me/XXXXXXXXXXX?text=Hola%20quiero%20pedir`

donde `XXXXXXXXXXX` es el número de WhatsApp de la fundación (en formato internacional, sin símbolos) y el parámetro `text` incluye el mensaje con los detalles del pedido en formato URL-encode.

El script tomará los productos del carrito y formateará un mensaje legible, por ejemplo:

`Hola, quisiera solicitar el siguiente pedido:%0A- Producto A x 2 unidades%0A- Producto B x 1 unidad%0ATotal: $YYY. Por favor, confirmarme disponibilidad.`

Al hacer clic, el navegador redirigirá a esa URL usando `window.location.href`, lo que en móviles abrirá directamente la app de WhatsApp (o en desktop, WhatsApp Web) iniciando el chat con el mensaje preparado. No se envía automáticamente el mensaje, sino que se abre para que el usuario pueda confirmar y presionar enviar desde WhatsApp.

Una vez en WhatsApp, la conversación continúa fuera del sitio: el encargado de la fundación responderá al mensaje, coordinará con el cliente los detalles de pago (por ejemplo, transferencia bancaria, pago contra entrega, etc.) y entrega o recogida del producto. Es decir, la "transacción" final ocurre mediante interacción humana vía WhatsApp, resolviendo así la limitación de no tener un sistema de pago en línea en el sitio estático.

Este enfoque tiene varias ventajas en el contexto dado:

- **(1) Simplicidad técnica** – se evita implementar pasarelas de pago o bases de datos, utilizando en cambio herramientas ya familiares (WhatsApp).
- **(2) Confianza y personalización** – muchos usuarios pueden preferir conversar con una persona de la fundación antes de cerrar una compra, lo que permite resolver dudas y genera confianza.
- **(3) Costo cero** – no requiere contratar servicios adicionales, ya que WhatsApp Business es gratuito y GitHub Pages aloja el sitio sin costo.

Consideraciones técnicas adicionales: el número de teléfono de WhatsApp se mostrará públicamente en el enlace, pero probablemente ya esté divulgado en la sección de Contacto, por lo que no es problema. Además, se debe instruir al usuario en caso de uso en escritorio: si hace clic en el enlace WhatsApp desde un PC sin WhatsApp Desktop instalado, se abrirá WhatsApp Web en el navegador.

Todo este proceso aprovecha la API oficial de WhatsApp para chat: *"clic para chatear"*, la cual no requiere que el usuario tenga guardado el número en sus contactos y funciona tanto en móviles como en web.

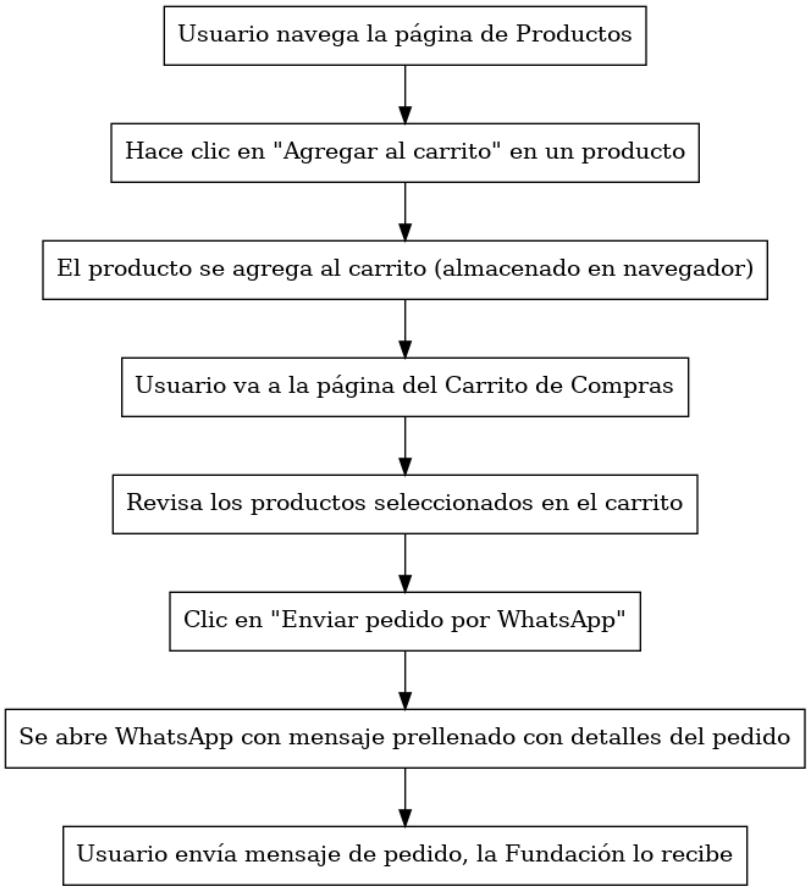

Figura 3: Flujo del carrito de compras e integración con WhatsApp en el sitio de la Fundación PROCAMSU Sumapaz.

En resumen, mediante este flujo el sitio logra ofrecer una experiencia de compra básica a pesar de ser estático: el usuario añade productos a un carrito local y luego, con un solo clic, inicia un chat de WhatsApp con la lista de su pedido. La Fundación PROCAMSU recibe el mensaje con el pedido detallado y puede continuar el proceso de manera personalizada. Este sistema es adecuado a la realidad de la fundación y su comunidad, donde WhatsApp es una herramienta ampliamente usada y confiable.

8 Diagramas UML

Con el fin de representar gráficamente la estructura y el comportamiento del sistema propuesto para el sitio web de la Fundación PROCAMSU de Sumapaz, se han elaborado tres diagramas UML que permiten visualizar tanto los componentes estructurales como las interacciones funcionales y dinámicas más relevantes.

8.1 Diagrama de Clases

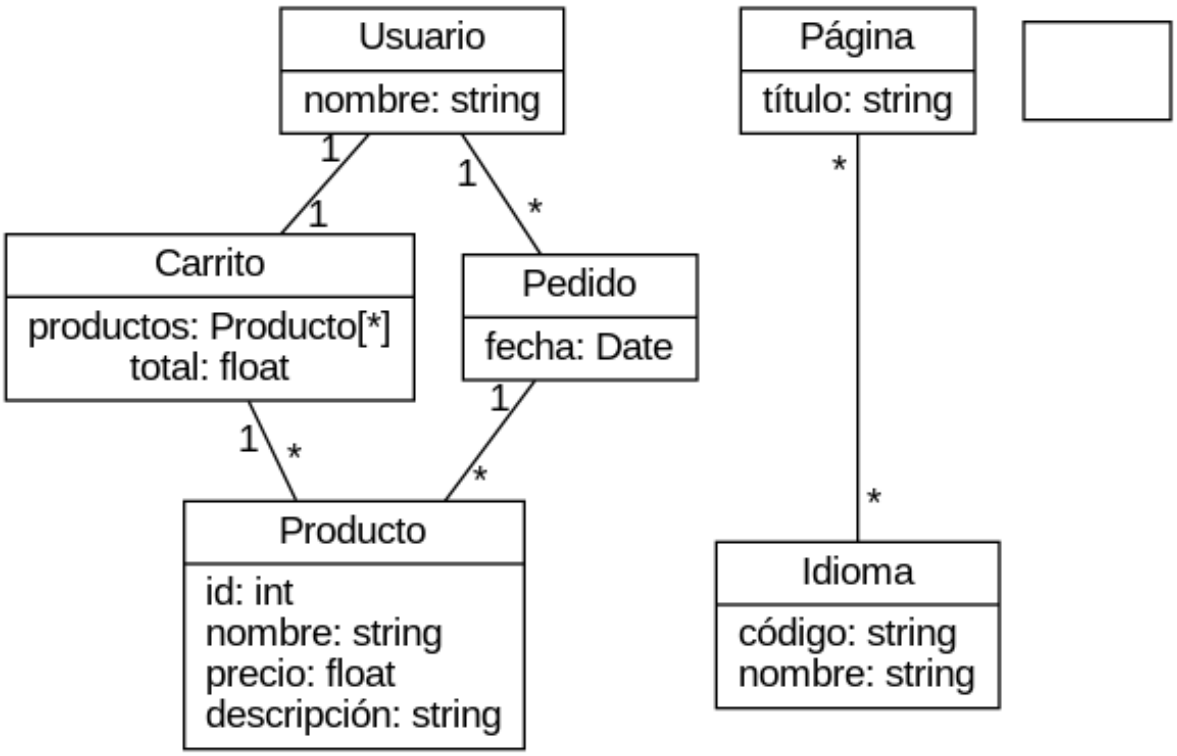

Figura 4: Diagrama de Clases del Sitio Web

Este diagrama representa las entidades principales del sistema y sus relaciones. Se destacan las siguientes clases:

- **Usuario**: Representa al visitante del sitio. Puede navegar entre páginas, agregar productos al carrito, cambiar el idioma, y enviar el pedido por WhatsApp.
- **Producto**: Define los atributos de los productos disponibles (nombre, precio, imagen, descripción). Es cargado dinámicamente en la página de productos.
- **Carrito**: Es una clase que almacena los productos seleccionados por el usuario. Contiene métodos para añadir, eliminar y listar productos.
- **Pedido**: Encapsula el resumen del carrito que será enviado mediante WhatsApp. Incluye método para generar el mensaje de texto compatible con el enlace de envío.
- **Página**: Clase abstracta que representa una vista del sitio. Se especializa en Inicio, Nosotros, Productos, Proyectos, Contacto y Carrito.
- **Idioma**: Clase encargada de gestionar el cambio de idioma (español o inglés) para los textos visibles en el sitio.

Las relaciones entre las clases reflejan composiciones y dependencias: por ejemplo, un `Carrito` contiene múltiples `Producto`, mientras que un `Usuario` puede generar un `Pedido` basado en el contenido de su `Carrito`.

8.2 Diagrama de Casos de Uso

Figura 5: Diagrama de Casos de Uso del Sitio Web

Este diagrama muestra las interacciones entre el **Actor Principal** (Usuario Visitante) y el sistema web. Los casos de uso modelados incluyen:

- **Navegar por las secciones del sitio**: Acceso a las páginas Inicio, Nosotros, Proyectos, Contacto y Productos.
- **Ver productos disponibles**: Visualización del catálogo de productos ofrecidos por la fundación.
- **Agregar producto al carrito**: Selección de productos y almacenamiento local.
- **Revisar carrito de compras**: Visualización del resumen de productos seleccionados.
- **Enviar pedido por WhatsApp**: Generación y envío del mensaje de solicitud de compra.
- **Cambiar idioma del sitio**: Alternar entre español e inglés como idioma de interfaz.
- **Contactar a la fundación**: Visualización de medios de contacto o uso de enlaces (email, teléfono o WhatsApp).

Este diagrama permite entender el alcance funcional del sistema desde la perspectiva del usuario y sirve de base para especificar los requisitos del frontend.

9 Casos de Uso Detallados

En esta sección se describen los casos de uso más representativos del sitio web de la Fundación PROCAMSU de Sumapaz, desarrollados con tecnologías estáticas (HTML, CSS y JavaScript) y alojado en GitHub Pages. Los casos de uso reflejan las funcionalidades que un usuario puede ejecutar dentro del sistema, sin requerir autenticación ni backend.

9.1 Actor Principal

Usuario Visitante: Es cualquier persona que accede al sitio web para informarse sobre la fundación, conocer sus productos o proyectos, y contactarse con la misma. Este actor puede interactuar con todas las secciones públicas del sitio.

9.2 Lista de Casos de Uso

A continuación se presentan los casos de uso identificados:

- UC01: Navegar por las secciones del sitio
- UC02: Ver listado de productos
- UC03: Agregar producto al carrito

- UC04: Visualizar y editar el carrito
- UC05: Enviar pedido por WhatsApp
- UC06: Cambiar idioma del sitio
- UC07: Ver información institucional
- UC08: Consultar proyectos activos
- UC09: Acceder a medios de contacto

9.3 Casos de Uso Detallados

9.3.1. UC01 - Navegar por las secciones del sitio

- **Actor principal:** Usuario Visitante
- **Propósito:** Permitir al usuario acceder a todas las secciones principales del sitio web mediante el menú de navegación.
- **Precondiciones:** El sitio web debe estar cargado correctamente en el navegador.
- **Flujo normal:**
  1. El usuario visualiza el menú de navegación.
  2. Selecciona una opción (Inicio, Nosotros, Productos, Proyectos, Contacto o Carrito).
  3. El navegador lo redirige a la página correspondiente.
- **Postcondiciones:** El usuario accede a la sección deseada.

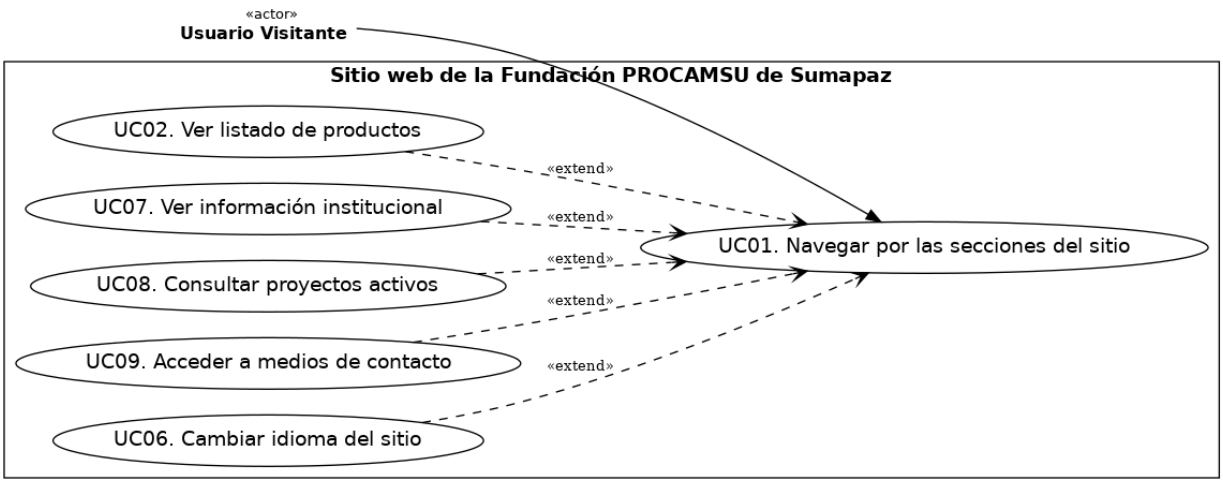

Figura 6: Diagrama de Caso de Uso UC01 - Navegar por las secciones del sitio

En este diagrama se muestra al actor **Usuario Visitante** interactuando con el sistema para recorrer las diferentes secciones disponibles del sitio web. El caso de uso principal *Navegar por las secciones del sitio* aparece dentro del límite del sistema (representado por el rectángulo rotulado como *Sitio web de la Fundación PROCAMSU de Sumapaz*).

Los demás casos de uso de secciones específicas —*Listar productos*, *Información institucional*, *Proyectos activos*, *Medios de contacto*— se modelan como extensiones (<<extend>>) del caso de uso principal de navegación.

Esto indica que, mientras navega, el usuario puede opcionalmente acceder a cada una de esas secciones desde el menú del sitio.

9.3.2. UC02 - Ver listado de productos

- **Actor principal:** Usuario Visitante
- **Propósito:** Mostrar al usuario todos los productos disponibles en formato visual y descriptivo.
- **Precondiciones:** La página de productos debe estar disponible.
- **Flujo normal:**
  1. El usuario accede a la página de productos.
  2. El sistema carga y renderiza dinámicamente el catálogo desde un archivo JS o JSON.
  3. El usuario puede visualizar nombre, precio, imagen y descripción de cada producto.
- **Postcondiciones:** El usuario visualiza los productos correctamente.

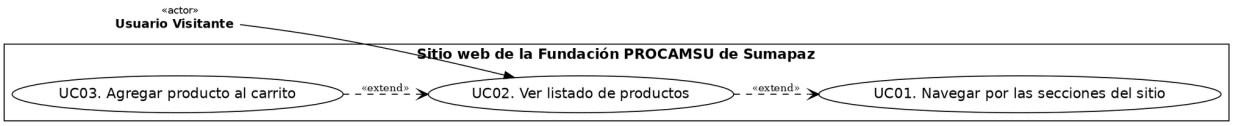

Figura 7: Diagrama de Caso de Uso UC02 - Ver listado de productos

Diagrama UML del caso de uso UC02: “Ver listado de productos”. Este diagrama muestra cómo el Usuario Visitante puede visualizar el catálogo de productos ofrecidos en el sitio. El caso de uso Ver listado de productos se representa dentro del sistema, con una relación de extensión (<<extend>>) desde el caso de uso Navegar por las secciones del sitio (UC01), lo que indica que el listado de productos es accesible al navegar por el sitio. Además, se incluye el caso de uso Agregar producto al carrito (UC03) como una extensión (<<extend>>) de Ver listado de productos, reflejando que, mientras examina el listado, el usuario puede opcionalmente agregar un producto determinado a su carrito de compra.

9.3.3. UC03 - Agregar producto al carrito

- **Actor principal:** Usuario Visitante
- **Propósito:** Permitir al usuario seleccionar productos para su compra y agregarlos al carrito.
- **Precondiciones:** El catálogo de productos debe estar cargado.
- **Flujo normal:**
  1. El usuario hace clic en "Agregar al carrito" en un producto.
  2. El sistema guarda la información del producto seleccionado en el carrito (localStorage o memoria).
  3. El sistema actualiza el ícono o contador del carrito en la interfaz.
- **Postcondiciones:** El producto queda registrado temporalmente en el carrito del usuario.

Figura 8: Diagrama de Caso de Uso UC03 - Agregar producto al carrito

Diagrama UML del caso de uso UC03: "Agregar producto al carrito". En este diagrama se representa la acción puntual de añadir un producto al carrito de compras por parte del Usuario Visitante. El caso de uso Agregar producto al carrito aparece asociado al actor y extendiendo (<<extend>>) al caso de uso Ver listado de productos (UC02). Esto significa que la función de agregar al carrito ocurre dentro del contexto de visualización de productos – es decir, el usuario, al estar viendo la lista de productos, puede elegir un producto específico y ejecutar esta acción de agregarlo al carrito. El diagrama enfatiza que "Agregar al carrito" es un flujo opcional que complementa al caso de uso de listado de productos.

9.3.4. UC04 - Visualizar y editar el carrito

- **Actor principal:** Usuario Visitante
- **Propósito:** Permitir al usuario revisar los productos seleccionados, modificar cantidades o eliminar productos.
- **Precondiciones:** El carrito debe contener al menos un producto.
- **Flujo normal:**
  1. El usuario accede a la página del carrito.

2. El sistema recupera los productos almacenados.
3. El sistema muestra una lista con los ítems seleccionados.
4. El usuario puede modificar cantidades o eliminar productos.
5. El total se recalcula automáticamente.

- **Postcondiciones:** El carrito queda actualizado según las acciones del usuario.

Figura 9: Diagrama de Caso de Uso UC04 - Visualizar y editar el carrito

Diagrama UML del caso de uso UC04: "Visualizar y editar el carrito". Este diagrama muestra cómo el Usuario Visitante puede acceder al contenido de su carrito de compras para revisarlo y modificarlo. El caso de uso Visualizar y editar el carrito está contenido dentro del sistema web e involucra al actor, indicando que el usuario puede abrir su carrito en cualquier momento para ver los productos agregados, cambiar cantidades o eliminar items. Asimismo, se incluye una relación de extensión (<<extend>>) desde el caso de uso Enviar pedido por WhatsApp (UC05) hacia Visualizar y editar el carrito – lo que refleja que una vez el usuario ha revisado/actualizado su carrito, opcionalmente puede proceder a la acción adicional de enviar el pedido mediante WhatsApp. En resumen, Enviar pedido por WhatsApp se presenta como una funcionalidad extendida que parte del escenario de visualizar el carrito listo para finalizar la compra.

9.3.5. UC05 - Enviar pedido por WhatsApp

- **Actor principal:** Usuario Visitante
- **Propósito:** Enviar el resumen del pedido a la fundación a través de un mensaje automático en WhatsApp.
- **Precondiciones:** El carrito debe contener productos válidos.
- **Flujo normal:**
  1. El usuario accede a la página del carrito y revisa el pedido.
  2. Hace clic en "Enviar por WhatsApp".
  3. El sistema genera un enlace con el texto del pedido formateado.
  4. El navegador redirige al usuario a la conversación de WhatsApp.

- **Postcondiciones:** Se inicia una conversación de WhatsApp con el pedido prellenado.

Figura 10: Diagrama de Caso de Uso UC05 - Enviar pedido por WhatsApp

Diagrama UML del caso de uso UC05: "Enviar pedido por WhatsApp". En este diagrama se representa el proceso mediante el cual el Usuario Visitante envía el pedido de los productos seleccionados a través de la aplicación WhatsApp. El caso de uso Enviar pedido por WhatsApp aparece ligado al actor y extendiendo (<<extend>>) al caso de uso Visualizar y editar el carrito (UC04). Esto indica que la acción de enviar el pedido es un flujo opcional que se ejecuta típicamente después de que el usuario haya revisado su carrito (es decir, se encuentra dentro del contexto de visualización del carrito). El diagrama deja claro que para poder enviar el pedido, el usuario previamente ha debido navegar al carrito y preparar allí el contenido del pedido, tras lo cual activa la función de envío vía WhatsApp.

9.3.6. UC06 - Cambiar idioma del sitio

- **Actor principal:** Usuario Visitante
- **Propósito:** Alternar el idioma de la interfaz entre español e inglés.
- **Precondiciones:** El selector de idioma debe estar visible en la interfaz.
- **Flujo normal:**
  1. El usuario hace clic en el botón de cambio de idioma.
  2. El sistema recarga los textos visibles con la traducción correspondiente.
  3. La interfaz se actualiza dinámicamente o redirige a otra carpeta/lenguaje.
- **Postcondiciones:** El sitio queda completamente mostrado en el idioma seleccionado.

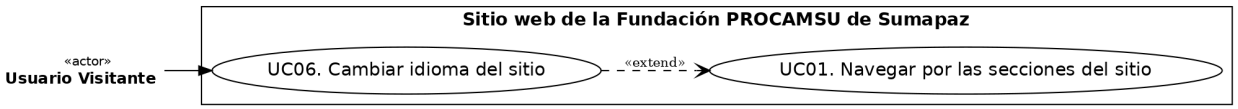

Figura 11: Diagrama de Caso de Uso UC06 - Cambiar idioma del sitio

Diagrama UML del caso de uso UC06: "Cambiar idioma del sitio". Este diagrama ilustra la funcionalidad que permite al Usuario Visitante alternar el idioma de presentación del

contenido del sitio web (por ejemplo, de español a inglés y viceversa). El caso de uso Cambiar idioma del sitio está representado dentro del sistema y asociado con el actor, mostrando que el visitante inicia esta acción. Adicionalmente, se muestra una relación de extensión (<<extend>>) desde Cambiar idioma hacia el caso de uso Navegar por las secciones del sitio (UC01). Esto refleja que la opción de cambiar el idioma está disponible durante la navegación general del sitio, siendo un comportamiento opcional que el usuario puede ejecutar en cualquier momento mientras recorre las secciones. En síntesis, el cambio de idioma ocurre en el contexto de la navegación del sitio sin interrumpirla, afectando a la visualización de todos los contenidos.

9.3.7. UC07 - Ver información institucional

- **Actor principal:** Usuario Visitante
- **Propósito:** Consultar la historia, misión y ubicación de la fundación.
- **Precondiciones:** La página "Nosotros" debe estar disponible.
- **Flujo normal:**
  1. El usuario accede a la sección "Nosotros".
  2. El sistema presenta la historia, misión/visión y ubicación geográfica.
- **Postcondiciones:** El usuario adquiere información contextual sobre la fundación.

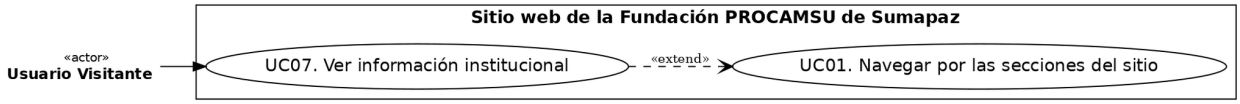

Figura 12: Diagrama de Caso de Uso UC07 - Ver información institucional

Diagrama UML del caso de uso UC07: "Ver información institucional". En este diagrama se presenta cómo el Usuario Visitante puede acceder a la sección de información institucional de la Fundación (por ejemplo, misión, visión, historia, etc.). El caso de uso Ver información institucional se muestra dentro del límite del sistema y conectado al actor, lo que indica que el usuario entra a esa sección para leer su contenido. Se ha modelado una relación <<extend>>que parte de Ver información institucional (UC07) hacia Navegar por las secciones del sitio (UC01). Esto significa que la visualización de la información institucional ocurre como parte de la navegación del sitio – es decir, el usuario llega a esta sección a través del menú o las opciones de navegación generales. La extensión <<extend>>denota que este caso de uso extiende el flujo de navegación cuando el visitante selecciona la sección institucional.

9.3.8. UC08 - Consultar proyectos activos

- **Actor principal:** Usuario Visitante
- **Propósito:** Explorar los proyectos en curso o finalizados que realiza la fundación.
- **Precondiciones:** La sección "Proyectos" debe estar habilitada.
- **Flujo normal:**
  1. El usuario accede a la página "Proyectos".
  2. El sistema muestra tarjetas o bloques con cada proyecto y su descripción.
- **Postcondiciones:** El usuario visualiza el contenido relacionado a los proyectos de la fundación.

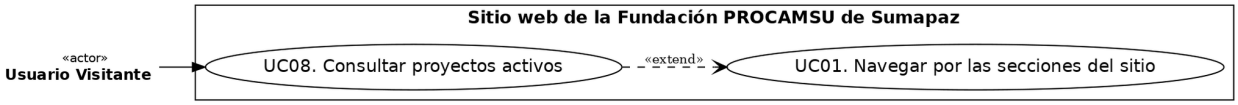

Figura 13: Diagrama de Caso de Uso UC08 - Consultar proyectos activos

Diagrama UML del caso de uso UC08: "Consultar proyectos activos". Este diagrama corresponde al caso de uso en el cual el Usuario Visitante visualiza la sección de proyectos activos de la Fundación (los proyectos actuales en curso). El caso de uso Consultar proyectos activos está representado dentro del sistema y asociado con el actor, indicando que el usuario accede a dicha sección para informarse sobre los proyectos. A su vez, se incluye una relación de extensión (<<extend>>) desde Consultar proyectos activos (UC08) hacia el caso de uso de Navegar por las secciones del sitio (UC01). Esta relación <<extend>>señala que el acceso a la información de proyectos activos ocurre en el contexto de la navegación general del sitio; en otras palabras, el usuario llega a consultar los proyectos a través del menú o sección correspondiente mientras navega por el sitio web.

9.3.9. UC09 - Acceder a medios de contacto

- **Actor principal:** Usuario Visitante
- **Propósito:** Permitir al usuario acceder a la información de contacto de la fundación.
- **Precondiciones:** La sección de contacto debe estar visible.
- **Flujo normal:**
  1. El usuario accede a la página "Contáctanos".
  2. El sistema presenta dirección, teléfono, correo, redes sociales y/o enlaces a WhatsApp.

- **Postcondiciones:** El usuario obtiene los datos necesarios para comunicarse con la fundación.

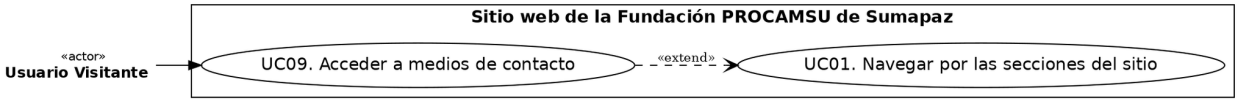

Figura 14: Diagrama de Caso de Uso UC09 - Acceder a medios de contacto

Diagrama UML del caso de uso UC09: “Acceder a medios de contacto”. En el último diagrama se muestra cómo el Usuario Visitante puede ingresar a la sección de contacto del sitio web para ver los distintos medios de comunicación disponibles (como número de teléfono, correo electrónico, redes sociales, formulario de contacto, etc.). El caso de uso Acceder a medios de contacto se representa dentro del sistema con asociación al actor, evidenciando que el visitante solicita ver la información de contacto de la Fundación. Mediante una relación de extensión (<<extend>>), Acceder a medios de contacto (UC09) extiende al caso de uso Navegar por las secciones del sitio (UC01). Esto indica que el acceso a los medios de contacto forma parte del flujo de navegación normal – el usuario llega a la sección de contacto utilizando la navegación del sitio. De esta manera, el diagrama enfatiza que la sección de contacto es una extensión opcional del recorrido general que el usuario realiza por el sitio web.

10 Escalabilidad, Mantenibilidad y Futuras Integraciones

Aunque inicialmente el sitio será simple, la arquitectura propuesta contempla principios que faciliten su escalabilidad y mantenimiento a largo plazo:

Escalabilidad de Contenido: Gracias a la estructura clara (páginas estáticas bien separadas y carpeta de datos), agregar nuevo contenido será sencillo. Por ejemplo, si en el futuro la fundación desea añadir una sección de “Noticias” o un blog, se puede crear una nueva página HTML y posiblemente un script para listarlas, sin tener que reestructurar todo el sitio. De igual forma, el listado de Productos puede crecer o incluir categorías (en tal caso, se podría reorganizar la página de productos para filtrar por categorías, manteniendo la misma lógica de cargar desde datos). La internacionalización también es escalable: el modelo de mantener contenidos en dos idiomas se puede extender a más lenguas si fuese necesario (añadiendo más carpetas o más entradas de traducción). No obstante, si el contenido creciera exponencialmente, se podría evaluar migrar a un generador de sitios estáticos (por ejemplo Jekyll, Hugo, Eleventy) que automatice la construcción de páginas repetitivas (como plantillas para noticias o proyectos) – esto sería transparente para el usuario final pero ayudaría a los editores del sitio.

Escalabilidad Funcional: Actualmente el sitio no cuenta con backend, lo cual limita ciertas funcionalidades (como se discutió en ventajas/desventajas). Si en el futuro la fundación requiriera funcionalidades más complejas – por ejemplo, un sistema de donaciones en línea, registro de usuarios/voluntarios, o un carrito de compras con pago en el sitio – habría que integrar tecnologías adicionales. Un camino evolutivo posible es adoptar un backend as a service (por ejemplo, Firebase para almacenar datos básicos o Netlify Functions/Azure Functions para pequeños endpoints) manteniendo la mayor parte del sitio estática. Otra posibilidad es migrar a una plataforma de gestión de contenido (CMS) adecuada: por ejemplo, WordPress (con hosting aparte) si se prioriza la facilidad de edición de contenido por personal no técnico, o herramientas headless CMS combinadas con nuestro frontend. Sin embargo, estas incorporaciones solo se justificarían al alcanzar cierto volumen de contenido o necesidad; mientras tanto, la configuración actual es suficiente y eficiente para el tráfico esperado.

Mantenibilidad del Código: Se está poniendo énfasis en escribir código limpio, modular y documentado. Dividir la lógica en archivos según su finalidad (como se hizo con CSS y JS) evita tener "mega archivos" difíciles de entender. Además, se seguirán convenciones de nombres consistentes (por ejemplo, clases CSS usando metodologías como BEM – Block Element Modifier – para mantener la claridad en estilos, p.ej., `.navbar__item--active`), y estructura de HTML coherente. Los comentarios se incluirán donde sea necesario para explicar partes no triviales (por ejemplo, en el código que construye el mensaje de WhatsApp, documentar el formato del link y la necesidad del encoding). Todo esto hace que un nuevo desarrollador que mire el repositorio pueda entender rápidamente cómo está construido el sitio.

Futuras Integraciones: El diseño actual deja puertas abiertas para integraciones futuras. Por ejemplo:

- Integrar Google Analytics u otra herramienta de analítica web es tan simple como agregar el script de seguimiento en la plantilla HTML (posiblemente en todas las páginas, idealmente factorizarlo para no repetirlo manualmente en cada HTML). Esto permitiría a la fundación obtener métricas de visitas, páginas más vistas, etc.
- Optimización para motores de búsqueda (SEO) ya se considera en prácticas actuales, pero se podría integrar un `sitemap.xml` auto-generado o usar las GitHub Actions para generar contenido dinámico si llegase a ser necesario (dado que GitHub Pages soporta CI/CD, se puede configurar una *action* para, por ejemplo, compilar Markdown en HTML si la fundación quisiera publicar artículos de blog escritos en Markdown).
- En el apartado de *e-commerce*: si con el tiempo se desea formalizar la venta en línea, se podría conectar el sitio a una plataforma de tienda online. Una opción ligera podría ser usar un formulario de pago embebido de Paypal o MercadoPago para ciertos productos, aunque eso requeriría programación o servicios externos. Otra opción es migrar la sección de tienda a un servicio como Shopify e integrarlo mediante enlaces o subdominio, manteniendo el resto del sitio en GitHub Pages.
- También, dado que la fundación trabaja con proyectos sociales, podría integrarse en el futuro mapas interactivos (usando APIs de Google Maps o OpenStreetMap) para mostrar las ubicaciones de actividades en Sumapaz, o incluso galerías de fotos más so-

fisticadas. Estas integraciones se pueden sumar incorporando librerías JS especializadas cuando surja la necesidad.

Performance y caching: La escalabilidad no solo se refiere a agregar funcionalidades, sino a mantener buen rendimiento con más tráfico o contenido. Un sitio estático en GitHub Pages ya es servido vía CDN, lo que otorga buena escalabilidad en términos de usuarios concurrentes (podría atender miles de visitas sin problemas de carga). A medida que crezca el contenido, se vigilará el rendimiento – por ejemplo, optimizando imágenes (usando formatos adecuados, tamaños responsivos) y minimizando CSS/JS. Se puede introducir una etapa de *build* opcional en el *workflow* de GitHub (mediante GitHub Actions) para minificar recursos en cada despliegue, si fuera necesario mejorar tiempos de carga cuando el tamaño de archivos aumente.

Seguridad: Mantener el sitio estático significa que la superficie de ataque es mínima (no hay base de datos que vulnerar ni formulario que inyectar código malicioso del lado servidor). Aun así, de crecer, se monitorearán posibles riesgos de las integraciones (p. ej., siempre usar enlaces HTTPS, considerar Content Security Policy si se incrustan recursos de terceros, etc.). GitHub Pages proporciona HTTPS gratuito, lo cual ya garantiza comunicaciones cifradas por defecto.

En síntesis, la arquitectura actual es ágil y apropiada para la etapa presente de la fundación; pero no es un callejón sin salida, sino un cimiento sobre el cual se puede construir. La adopción de estándares y buenas prácticas garantiza que, conforme surjan nuevas necesidades, el equipo pueda ampliarlas con un esfuerzo incremental y sin necesidad de reescribir desde cero. La clave está en mantener la base ordenada y simple, lista para incorporar módulos o integraciones de manera orgánica.

11 GitHub Pages – Ventajas y Desventajas de un Sitio Estático sin Backend

El sitio será hospedado mediante GitHub Pages, aprovechando que se trata de un sitio estático. A continuación, se analizan las ventajas y desventajas de esta decisión arquitectónica, especialmente en contraste con soluciones con backend:

Ventajas de GitHub Pages y sitio estático:

Hosting gratuito y confiable: GitHub Pages permite alojar sitios estáticos directamente desde un repositorio de GitHub, de forma totalmente gratuita y con altísima disponibilidad. Esto significa cero costo de hosting para la fundación, lo cual es ideal dado su presupuesto probablemente limitado, y la infraestructura de GitHub (respaldada por CDN global de GitHub) asegura tiempos de carga rápidos y servicio ininterrumpido.

Simplicidad de despliegue: La publicación de cambios es tan sencilla como hacer *push* de las actualizaciones al repositorio. Cada commit en la rama configurada (por ejemplo, `main` o `gh-pages`) provoca que GitHub Pages regenere el sitio. Esto encaja muy bien con un

flujo de trabajo de control de versiones (discutido más adelante), permitiendo incluso que colaboradores externos hagan *pull requests* para mejorar el sitio.

Velocidad y rendimiento: Los sitios estáticos suelen cargar más rápido que los dinámicos equivalentes, ya que se entregan archivos HTML preconstruidos sin necesidad de consultas a base de datos ni renderizado en servidor. Esto mejora la experiencia de usuario (especialmente en zonas rurales con internet limitado, como podría ser Sumapaz) y también es bueno para SEO. Además, al no cargar frameworks pesados, nuestro sitio tiene un peso ligero y arranca rápido en el navegador.

Seguridad: Al no tener una capa de servidor ejecutando lógica ni base de datos, se eliminan gran cantidad de vectores de ataque comunes (inyecciones SQL, exploits de autenticación, etc.). Un sitio estático en GitHub Pages es inherentemente más seguro, puesto que GitHub se encarga de la seguridad del servidor web y no hay partes dinámicas vulnerables. Por supuesto, siempre hay consideraciones (por ejemplo, proteger el repositorio, usar HTTPS que viene por defecto, etc.), pero en general el riesgo es bajísimo. Esto es importante para la fundación, ya que probablemente no cuenta con un equipo de TI dedicado a monitorear seguridad constantemente.

Menor complejidad y costo de mantenimiento: No hace falta gestionar servidores, ni configurar entornos complejos – GitHub Pages se ocupa de servir el contenido. Cualquier miembro con conocimientos básicos de HTML podría actualizar una página directamente en GitHub si fuera necesario. Esto reduce la necesidad de personal técnico para mantener la presencia web. También se pueden usar las herramientas de GitHub (Issues, Projects) para llevar registro de cambios o mejoras del sitio, centralizando todo en una misma plataforma.

Desventajas y limitaciones:

Falta de interactividad y procesamiento en servidor: Un sitio estático no puede manejar por sí mismo operaciones como formularios avanzados (por ejemplo, un formulario "Contáctanos" que guarde datos o envíe emails requeriría un servicio adicional), autenticación de usuarios, generación de contenido personalizado, etc. En nuestro caso, esto significa que implementaciones como carritos de compra reales, sistemas de login o bases de datos no son posibles de forma nativa. Hemos solventado lo del carrito parcialmente vía WhatsApp, pero es cierto que, comparado con un sitio con backend, hay menos posibilidades de ofrecer experiencias interactivas totalmente integradas.

Actualización manual del contenido: Sin un gestor de contenidos (CMS) o backend, cada cambio de contenido implica editar archivos HTML/JS/CSS en el repositorio. Para una persona sin conocimientos técnicos, esto puede ser una barrera; aunque Markdown podría ser usado con GitHub Pages (por ejemplo mediante Jekyll), hemos optado por HTML directo ahora. Si la fundación quisiera, por ejemplo, que personal administrativo actualizara noticias sin tocar código, un sitio estático puro es menos amigable. Sin embargo, dado el volumen de contenido manejable, este problema es relativo. Las actualizaciones en un sitio estático no son inmediatas a no ser que se hagan los despliegues, no hay una interfaz administrativa gráfica sencilla como en un CMS.

Integraciones limitadas: Cualquier funcionalidad más allá de presentar información re-

quiere pensar soluciones creativas o apoyarse en terceros. Por ejemplo, para tener un mapa de Google quizás haya que incluir un `iframe` o usar su API JS; para tener comentarios en un blog estático, habría que integrar algo como Disqus (un servicio externo); para recibir formularios, usar servicios tipo Formspree o Google Forms embebido, etc. Esto añade dependencias externas o capas adicionales. En un servidor propio, se podría programar todo internamente. Es decir, un sitio estático tiene funcionalidad acotada, y expandirlo implica a veces "pegar" servicios de terceros.

Escalabilidad funcional: Si bien un sitio estático escala en número de visitas fácilmente, no escala tan bien en complejidad de aplicación. Por ejemplo, manejar inventarios en tiempo real, o mostrar contenido distinto según el usuario, son cosas impracticables sin un backend. Para crecer en esas direcciones, habría que migrar a un enfoque dinámico. Por eso se suele decir que los sitios estáticos son ideales para contenidos informativos y casos sencillos, pero no para aplicaciones web complejas de gran tamaño. En nuestro caso, esta desventaja no es crítica ahora, pero es consciente.

GitHub Pages specific: Algunas limitaciones propias de GitHub Pages incluyen el tamaño máximo del repositorio (actualmente 1 GB para contenido estático, más que suficiente por ahora) y ciertas restricciones en el uso de tecnologías del lado del servidor (por ejemplo, no se puede usar directamente PHP, ni bases de datos). Además, si la fundación quisiera un dominio personalizado, puede configurarlo gratis en Pages, pero deben adquirir el dominio aparte. Otra consideración es que, si el repositorio es público, el código fuente del sitio es visible para cualquiera (lo cual normalmente no es problemático ya que es HTML/JS público de todos modos, pero cualquier información sensible no debe estar en el repo público). Se puede usar repos privado con GitHub Pro, pero eso añade costo; mantenerlo público a cambio es completamente aceptable y transparente.

En balance, la decisión de usar GitHub Pages y un sitio estático ofrece grandes ventajas en simplicidad, costo y seguridad, sacrificando ciertas funcionalidades avanzadas que, por la naturaleza del proyecto, no son prioritarias en esta etapa. Para la Fundación PROCAMSU, esto significa tener presencia en la web de forma confiable y profesional, con un mantenimiento al alcance de sus capacidades, siempre evaluando en el camino si en el futuro alguna desventaja pasa a pesar más que las ventajas (en cuyo caso se re-plantearía la arquitectura hacia una más híbrida o dinámica según corresponda).

12 Buenas Prácticas Adicionales (Accesibilidad, SEO, Modularización, Control de Versiones)

Para asegurar la calidad y longevidad del sitio, se seguirán una serie de buenas prácticas de desarrollo web en las áreas de accesibilidad, optimización para buscadores, modularidad del código y control de versiones:

Accesibilidad Web: Se diseñará el sitio bajo el principio de "accesible para todos", cumpliendo con estándares como WCAG en lo posible. Esto implica usar HTML semántico (ya

mencionado) para que los lectores de pantalla puedan interpretar correctamente la estructura del contenido. Todas las imágenes incluirán textos alternativos descriptivos mediante el atributo `alt`, de modo que usuarios con discapacidad visual puedan entender el contenido visual a través de sus lectores de pantalla. Asimismo, se cuidará el contraste de colores (para legibilidad por personas con baja visión o daltonismo) y se asegurará que toda funcionalidad sea operable vía teclado (por ejemplo, la navegación por menús debe poder hacerse con Tab/Enter, importantísimo para usuarios que no usan ratón). Componentes interactivos, como botones o links generados por JS, tendrán atributos ARIA si es necesario para anunciar su estado (por ejemplo, un botón de idioma podría tener `aria-label=Çambiar a inglés"`). Mantener un sitio accesible no solo es inclusivo, sino que beneficia al SEO, ya que muchas prácticas de accesibilidad (como proporcionar textos alternativos y estructura lógica) ayudan a los motores de búsqueda a entender mejor el contenido. En resumen: etiquetas adecuadas, formularios con `label`, evitar contenido parpadeante que pueda afectar a epilépticos, y realizar pruebas con herramientas automáticas (p. ej. extensión WAVE o Lighthouse) para detectar posibles mejoras en accesibilidad.

SEO básico (Optimización para buscadores): Desde el inicio se incorporarán las metaetiquetas y estructura necesarias para un buen SEO on-page. Cada página tendrá un título (`<title>`) descriptivo y único indicando el nombre de la fundación y la sección (por ejemplo, "Fundación PROCAMSU Sumapaz – Nosotros"), así como una meta descripción que resuma su contenido, ya que esas descripciones suelen aparecer en los resultados de búsqueda. El marcado semántico también apoya al SEO, pues los buscadores premian la estructura clara. Se asegurará que las URLs sean amigables (en nuestro caso, los nombres de archivo ya lo son, e.j. "nosotros", "contacto" en español; en la versión en inglés quizás se usen los mismos nombres o equivalentes traducidos para consistencia). Adicionalmente, GitHub Pages permite fácilmente agregar un archivo `sitemap.xml`; en caso de que la fundación lo requiera, podríamos generarlo manualmente o con una herramienta para listar todas las páginas del sitio y sus últimas modificaciones, facilitando así la indexación por Google. Otra buena práctica es utilizar enlaces internos: por ejemplo, desde la página de Inicio enlazar contenido de Proyectos destacados con "leer más" que lleva a Proyectos, etc., creando una red interna que los buscadores interpretan para asignar relevancia. Por último, se añade la etiqueta `lang=.es"` en la versión española y `lang=.en"` en la inglesa, indicando el idioma del contenido, lo que ayuda a buscadores y navegadores a entregar la versión apropiada a usuarios de distintas lenguas. Aunque el SEO es un campo amplio, implementando estas bases nos aseguramos de que el sitio sea visible y entendible por los buscadores, incrementando la probabilidad de que gente buscando términos relacionados (ej. "fundación Sumapaz proyectos") encuentre nuestra página.

Modularización y Limpieza de Código: En un proyecto que se quiere escalable y mantenible, es crucial evitar la duplicación de código y seguir la filosofía DRY ("Don't Repeat Yourself"). Por ello, se buscará reutilizar componentes en lugar de reescribirlos. Por ejemplo, si varias páginas comparten el mismo encabezado y pie, se podría hacer uso de elementos *include* en el futuro (si se emplea Jekyll o un SSG) o bien replicar manualmente asegurando actualizar en paralelo ambos idiomas. Mientras no tengamos motor de plantillas, documentaremos claramente las secciones repetidas para mantener la consistencia. En CSS, se emplearán clases reutilizables y, de ser apropiado, utilidades (ej. una clase `.hide` para esconder

elementos visualmente pero accesible solo a lectores, etc.). La modularidad también aplica a JavaScript: en vez de tener un único archivo enorme que controla todo, se segmentaron funcionalidades (como describimos en la estructura de archivos). Esto encapsula la lógica y reduce el riesgo de efectos colaterales. Por ejemplo, `productos.js` puede exponer funciones específicas pero no ensuciar el espacio global más de lo necesario. Mantener funciones puras y bien definidas hará más fácil probar cambios. Asimismo, se aplicarán buenas prácticas de codificación: nombres de variables y funciones descriptivos, comentarios donde el funcionamiento no sea obvio, y evitando "code smells" (como variables globales innecesarias, loops complicados cuando se puede usar métodos funcionales de array, etc.). Antes de cada release, se puede formatear el código (por convención, usando 2 o 4 espacios de indentación consistentemente, etc.) para que los *diffs* en Git sean limpios.

Control de Versiones con Git: Dado que el proyecto vive en GitHub, aprovecharemos al máximo las bondades de Git para asegurar un desarrollo ordenado. Todo cambio en el código estará asociado a un commit con un mensaje claro que indique qué se hizo (ejemplo de convención: prefijo del área `[WEB]`, `[CSS]`, `[FIX]`, `[FEAT]` seguido de descripción: `"[FEAT] Agregar sección de proyectos en página Proyectos"`). Esto ayuda a llevar un historial legible. Además, para funcionalidades mayores o refactorizaciones, es aconsejable crear ramas (*branches*) separadas. Por ejemplo, una rama `internationalization` para trabajar la integración del idioma inglés por completo, y luego hacer *merge* a la rama principal cuando esté listo, revisando que no rompa nada. GitHub también permite la revisión colaborativa mediante *pull requests*, de modo que si hay más de un desarrollador, se pueden usar PRs para discutir cambios antes de integrarlos. Se mantendrá el repositorio bien organizado con `tags` o `releases` si hay hitos importantes (por ejemplo, `v1.0 lanzamiento inicial`). Otra buena práctica es utilizar GitHub Issues para reportar tareas o mejoras al sitio, lo cual queda fuera del alcance del documento de arquitectura, pero cabe mencionarlo como parte de un ecosistema de buenas prácticas. En cuanto a GitHub Pages específico, cada cambio en la rama publicada dispara un despliegue automático. Se vigilará en la pestaña de Pages de GitHub que no haya errores (por ejemplo, si accidentalmente se incluye algún elemento no soportado). Un detalle técnico: agregar un archivo `.nojekyll` en la raíz para deshabilitar el procesamiento Jekyll (ya que al no usarlo, evitamos que trate de interpretar nuestros archivos; esto es especialmente necesario si tenemos carpetas con nombres que Jekyll ignora, como aquellas que empiezan con _ , aunque no es nuestro caso). Esta medida previene problemas y hace que GitHub Pages sirva exactamente lo que tenemos en la carpeta, sin modificaciones. El control de versiones y GitHub Pages van de la mano, brindando trazabilidad de cambios y facilidad de restaurar una versión previa del sitio si algo sale mal en una actualización.

Calidad y revisión: Aunque es un sitio pequeño, se procurará probarlo en distintos navegadores (Chrome, Firefox, Safari, Edge) y dispositivos para detectar cualquier inconsistencia (p. ej., estilos quebrados en pantallas muy pequeñas, o alguna funcionalidad JS que falle en un navegador específico). También, antes de hacer *deploy* de nuevos cambios, se puede probar el sitio en local (abriendo los HTML en un navegador o montando un servidor estático simple) para verificar que todo siga funcionando. Este hábito de prueba continua entra dentro de las buenas prácticas generales de desarrollo web.

En conclusión, al adherirnos a estas buenas prácticas, buscamos que el sitio de la Fundación

PROCAMSU no solo cumpla con los requisitos funcionales iniciales, sino que también sea robusto, fácil de usar y de mantener en el tiempo. La accesibilidad y SEO asegurarán que llegue al mayor público posible y cumpla su misión social; la modularización y control de versiones garantizarán que el crecimiento del sitio sea ordenado y colaborativo. Con esta base, el sitio está preparado para servir como una plataforma confiable de comunicación entre la fundación y la comunidad, hoy y en el futuro.

Referencias y Fuentes: La presente arquitectura se ha elaborado considerando principios generales de desarrollo web y las necesidades específicas de la Fundación. Se han incorporado lineamientos y ejemplos de estructura de proyectos web, recomendaciones oficiales de integración con WhatsApp, así como pautas de accesibilidad y SEO reconocidas, entre otros recursos citados a lo largo del documento. Estas referencias respaldan las decisiones técnicas tomadas y sirven como guía para la implementación detallada del sitio.