

Manual Técnico y Funcional - Entregable Final Frontend (Gambito Soluciones)

TUTOR

John Olarte

Integrantes del Grupo

Diego Alejandro Gamboa Beltrán

POLITÉCNICO GRANCOLOMBIANO

2025

Tabla de contenido

1. Descripción general del proyecto	3
2. Estructura del repositorio	3
3. Requerimientos técnicos	4
4. Instalación y despliegue	4
4.1. Ejecución local:	4
4.2. Despliegue en GitHub Pages:	4
5. Arquitectura funcional del sistema	5
5.1. Visión general	5
5.2. Estructura general	5
5.3. Módulos principales	6
5.4. Módulos de scripts (JS)	6
5.5. Flujo funcional	7
5.6. Interacciones entre módulos	7
6. Flujo de navegación	7
6.1. Descripción General del Flujo	8
6.1.1. Inicio (Index.html)	8
6.1.2. Catálogo de Productos (catalogo.html)	8
6.1.3. Carrito de Compras (carrito.html)	9
6.1.4. Inicio de Sesión (login.html)	10
6.1.5. Sección “Acerca de” y Formulario de Contacto (acerca.html)	11
6.1.6. Bandeja de Mensajes (bandeja.html)	12
6.1.7. Navegación Dinámica y Navbar Persistente (main.js)	12
6.1.8. Comportamiento Técnico	12
6.1.9. Resumen del Flujo	13
7. Mantenimiento y actualizaciones	13
7.1. Mantenimiento Preventivo	13
7.2. Mantenimiento Correctivo	14
7.3. Mantenimiento Evolutivo	14
7.4. Proceso de Actualización en GitHub Pages	14
7.5. Buenas Prácticas Recomendadas	15
7.6. Actualización de Documentación	15
8. Especificación técnica de funciones (por módulo y archivo)	15
8.1. Archivo: main.js	15
8.2. Archivo: auth.js	16
8.3. Archivo: login.js	16

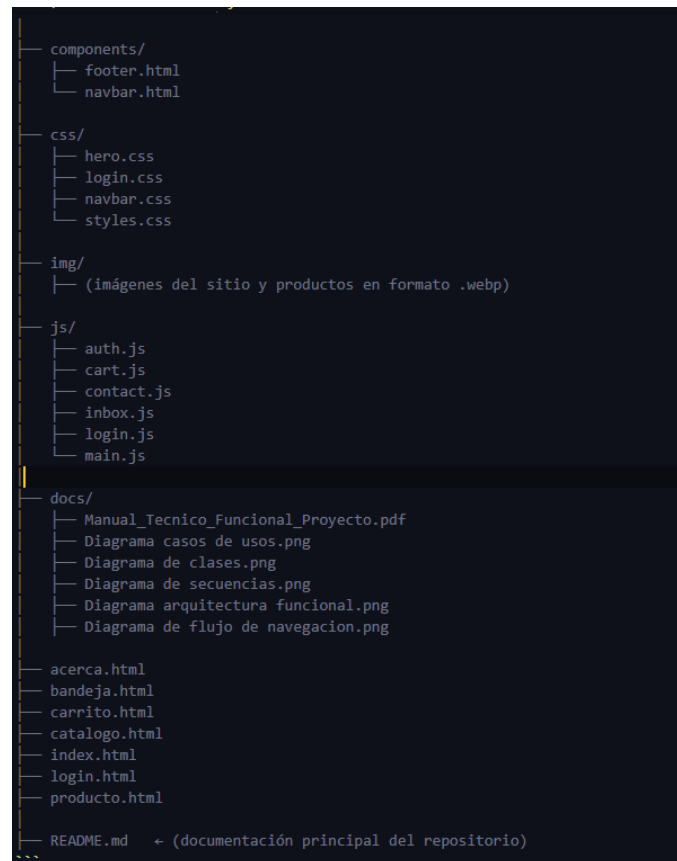
8.4.	Archivo: cart.js	16
8.5.	Archivo: contact.js	17
8.6.	Archivo: inbox.js.....	17
8.7.	Archivo: catalogo.html (interacción).....	18
8.8.	Archivo: bandeja.html.....	18
8.9.	Archivo: acerca.html.....	18
8.9.1.	Resumen General.....	19
9.	Diagramas UML del Sistema	19
9.1.	Diagrama de casos de uso.....	19
9.2.	Diagrama de Clases	20
9.3.	Diagrama de Secuencia	21
9.4.	Diagrama de Arquitectura Funcional.....	21
9.5.	Diagrama Complementario: Flujo de Navegación Funcional.....	22

1. Descripción general del proyecto

El presente documento detalla el desarrollo, estructura y funcionamiento del proyecto web 'Gambito Soluciones'. Este sistema fue diseñado como un entorno web moderno, adaptable y funcional, construido con tecnologías de frontend como HTML5, CSS3, JavaScript y Bootstrap 5.3. El sitio se encuentra desplegado en GitHub Pages.

2. Estructura del repositorio

Repositorio: <https://github.com/DiegoGamboa25/Entregable-2-frontend->



3. Requerimientos técnicos

Software requerido	Librerías
Visual Studio Code	Bootstrap 5.3
Git y GitHub Pages	Bootstrap Icons
Navegador compatible con HTML5	JavaScript (ES6)

4. Instalación y despliegue

4.1. Ejecución local:

1. Clonar el repositorio con: `git clone https://github.com/DiegoGamboa25/Entregable-2-frontend-.git`
2. Abrir en Visual Studio Code.
3. Ejecutar con Live Server.

4.2. Despliegue en GitHub Pages:

1. Subir cambios con `git add .`, `git commit` y `git push`.

2. Configurar GitHub Pages desde Settings → Pages.
3. URL pública: <https://diegogamboa25.github.io/Entregable-2-frontend/>
4. Bandeja de mensajes: <https://diegogamboa25.github.io/Entregable-2-frontend-/bandeja.html>

5. Arquitectura funcional del sistema

5.1. Visión general

La arquitectura del sistema Gambito Soluciones está basada en un modelo Frontend modular estático, apoyado completamente en HTML, CSS, JavaScript y Bootstrap 5, sin necesidad de un backend real.

El almacenamiento y flujo de información se simula a través del LocalStorage del navegador, permitiendo mantener los datos de usuario, productos y mensajes entre las diferentes vistas.

La comunicación entre los módulos se logra mediante:

- Importación dinámica de componentes (navbar y footer) cargados por main.js.
- Persistencia local (datos guardados en el navegador: usuario, carrito, bandeja).
- Scripts especializados por función (cada módulo JS atiende una parte de la aplicación).

5.2. Estructura general

El proyecto se organiza en cuatro capas principales:

Capa	Descripción	Archivos claves
Presentación (Frontend)	Contiene las páginas HTML accesibles al usuario final. Cada vista es independiente y carga dinámicamente los componentes comunes como la barra de navegación y el pie de página.	index.html, catalogo.html, carrito.html, login.html, acerca.html, bandeja.html, producto.html
Lógica de Aplicación (Scripts JS)	Gestiona la funcionalidad dinámica de la interfaz, interacción con el almacenamiento local, autenticación simulada y renderizado de elementos.	main.js, auth.js, login.js, cart.js, contact.js, inbox.js
Recursos Estáticos	Contiene estilos, imágenes y fragmentos HTML reutilizables.	css/, img/, components/

Persistencia Simulada (LocalStorage)	Actúa como almacenamiento persistente en el navegador. Se utiliza para mantener sesiones, productos añadidos al carrito y mensajes enviados desde el formulario de contacto.	Sin archivos físicos (almacenamiento interno del navegador)
--------------------------------------	--	---

5.3. Módulos principales

Módulo / Página	Función principal	Interacción
index.html	Página de inicio. Presenta la empresa y redirige al catálogo.	Usa main.js para insertar el navbar/footer y mantener coherencia visual.
catalogo.html	Muestra los productos o servicios disponibles.	Usa catalogo.js y cart.js para gestionar el evento “Agregar al carrito”.
carrito.html	Visualiza los productos añadidos. Permite eliminar o simular la compra.	cart.js controla renderizado, cálculo de totales, vaciado y simulación de compra.
login.html	Simula inicio de sesión con usuarios de prueba.	login.js valida y guarda el usuario en localStorage.
acerca.html	Presenta la empresa y el formulario de contacto .	contact.js maneja el envío del formulario y guarda mensajes en localStorage.
bandeja.html	Simula una bandeja de entrada con los mensajes enviados desde “Acerca de”.	inbox.js recupera los mensajes desde localStorage y los muestra en tabla.
producto.html	Página opcional para ampliar información de un producto.	Puede reutilizar datos del catálogo o cargarse con localStorage.
components/navbar.html	Barra de navegación común.	Cargada dinámicamente en todas las páginas por main.js.
components/footer.html	Pie de página común.	Cargado dinámicamente en todas las páginas.

5.4. Módulos de scripts (JS)

Archivo	Descripción técnica
Main.js	Carga e inserta dinámicamente navbar.html y footer.html según la ruta (local o GitHub Pages). También inicializa el contador del carrito (updateCartCount()).
Auth.js	Detecta si hay un usuario activo (guardado en localStorage). Si existe, cambia “Iniciar sesión” por “Cerrar sesión” en el menú.
login.js	Simula un sistema de autenticación con usuarios de prueba. Si el correo y contraseña coinciden, almacena el objeto usuario en localStorage.

cart.js	Gestiona todo el flujo del carrito: agregar, eliminar, vaciar, calcular totales y simular una compra. También actualiza el contador global del carrito en el navbar.
Contact.js	Controla el formulario de contacto en “acerca.html”. Guarda los mensajes enviados en localStorage bajo la clave messages.
Inbox.js	Recupera los mensajes almacenados y los muestra en “bandeja.html” con una tabla dinámica. También puede eliminar registros si se desea.

5.5. Flujo funcional

Flujo de navegación principal:

Inicio → Catálogo → (Agregar al carrito) → Carrito → (Simular compra)

 ↘ Inicio de sesión → login → (Enviar) → (Simular login)

 ↘ Acerca de → (Enviar formulario) → Bandeja de mensajes

5.6. Interacciones entre módulos

1. main.js → todas las páginas
 - Inserta el encabezado y pie, además de activar el contador del carrito.
2. catalogo.js + cart.js
 - El primero crea los objetos producto y el segundo gestiona el almacenamiento y visualización.
3. login.js + auth.js
 - login.js crea el usuario y auth.js modifica la interfaz en función de la sesión.
4. contact.js + inbox.js
 - contact.js guarda los mensajes del formulario en localStorage y inbox.js los lista en “bandeja.html”.

6. Flujo de navegación

El flujo de navegación define el recorrido que realiza el usuario dentro del sistema Gambito Soluciones, desde el ingreso inicial hasta la interacción con el catálogo, el carrito, el formulario de contacto y la bandeja de mensajes.

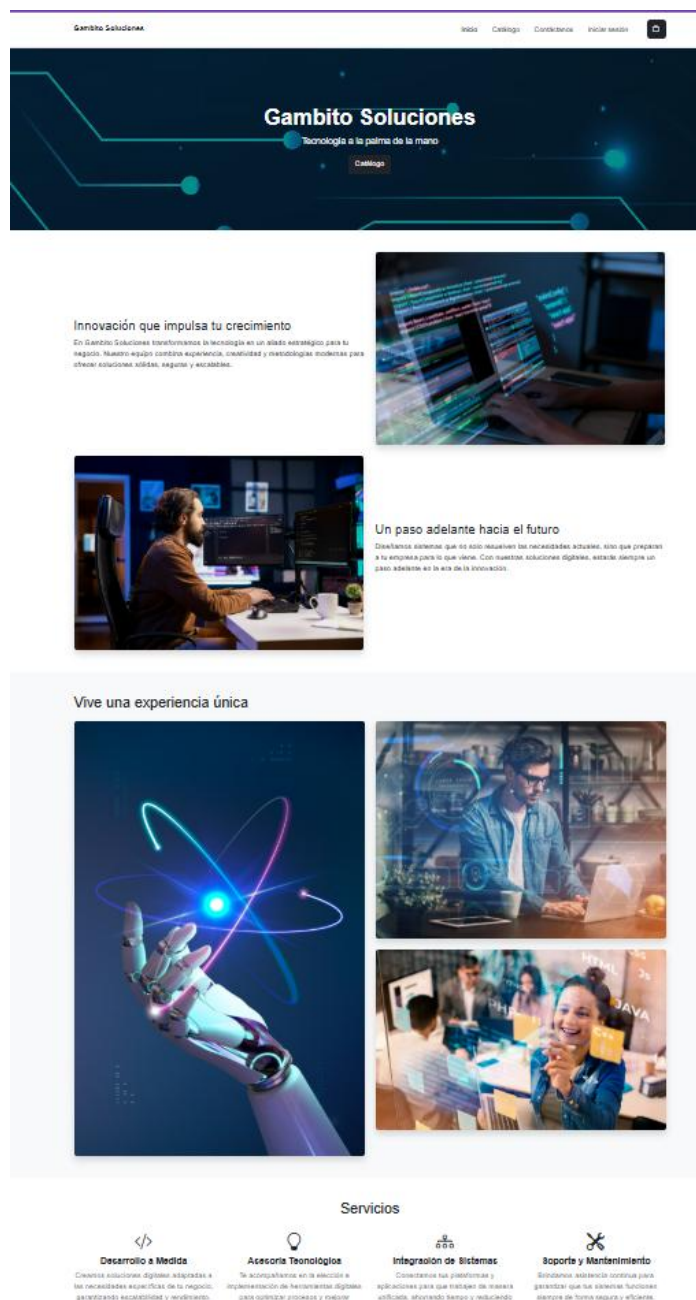
Este flujo está completamente implementado en el Frontend, haciendo uso de la manipulación del DOM, eventos y almacenamiento en LocalStorage, lo que permite simular un entorno de aplicación web completa sin servidor.

6.1. Descripción General del Flujo

6.1.1. Inicio (Index.html)

El usuario ingresa al sitio principal, donde se presentan los servicios y una vista general de la empresa.

Desde aquí puede acceder al catálogo o a las secciones de información.









6.1.2. Catálogo de Productos (catalogo.html)

El usuario puede visualizar los productos y agregar ítems al carrito mediante el botón “Agregar al carrito”.

Cada producto se almacena en localStorage junto con su nombre, precio e imagen



Catálogo

 <p>SISTEMA DE FACTURACIÓN ELECTRÓNICA GAMBITO SOLUCIONES</p> <p>Sistema de Facturación Electrónica Plataforma completa para emitir, recibir y almacenar facturas electrónicas con soporte para la DIAN.</p> <p>\$2.500.000</p> <p>Agregar al carrito</p>	 <p>PÁGINA WEB CORPORATIVA GAMBITO SOLUCIONES</p> <p>Página Web Corporativa Sitio web institucional con hasta 5 secciones (Inicio, Servicios, Nosotros, Contacto, Blog) optimizado para SEO y responsive.</p> <p>\$1.200.000</p> <p>Agregar al carrito</p>	 <p>TIENDA VIRTUAL (E-COMMERCE) GAMBITO SOLUCIONES</p> <p>Tienda Virtual (E-commerce) Plataforma de comercio electrónico con carrito de compras, pasarela de pagos integrada y gestión de inventario.</p> <p>\$3.500.000</p> <p>Agregar al carrito</p>
 <p>SISTEMA DE GESTIÓN DE INVENTARIOS GAMBITO SOLUCIONES</p> <p>Sistema de Gestión de Inventarios Software para registrar productos, movimientos de entrada/salida, alertas de stock y reportes en tiempo real.</p>	 <p>APLICACIÓN WEB DE RESERVAS GAMBITO SOLUCIONES</p> <p>Aplicación Web de Reservas Herramienta para gestionar citas o reservas (ideal para médicos, barberías, hoteles, etc.), con calendario y notificaciones.</p>	 <p>CHATBOT DE ATENCIÓN AL CLIENTE GAMBITO SOLUCIONES</p> <p>Chatbot de Atención al Cliente Bot automatizado para responder preguntas frecuentes en WhatsApp, Messenger o web, con inteligencia básica de respuestas.</p>

6.1.3. Carrito de Compras (carrito.html)

Desde esta vista se muestran los productos añadidos.

El usuario puede:

- Eliminar productos.
- Vaciar el carrito.
- Simular una compra (mensaje de confirmación).
- El carrito se actualiza en tiempo real mediante la función `renderCart()` y conserva su estado en `localStorage`.

Gambito Soluciones Inicio Catálogo Contáctanos Iniciar sesión

Tu Carrito de Compras

Imagen	Producto	Precio	Cantidad	Subtotal	Acción
	Página Web Corporativa	\$1.200.000	1	\$1.200.000	
	Sistema de Facturación Electrónica	\$2.500.000	1	\$2.500.000	

Vaciar carrito Total: \$3.700.000

Finalizar compra Seguir comprando

© 2025 Gambito Soluciones. Todos los derechos reservados.

6.1.4. Inicio de Sesión (login.html)

El usuario puede autenticarse mediante un login simulado con usuarios ficticios (almacenados en login.js).

Usuarios ficticios:

- email: "diego@example.com", password: "1234"
- email: "admin@example.com", password: "admin"

Inicie sesión en su cuenta

Continuar con Google

o inicie sesión con correo electrónico

Email

diego@example.com

Password

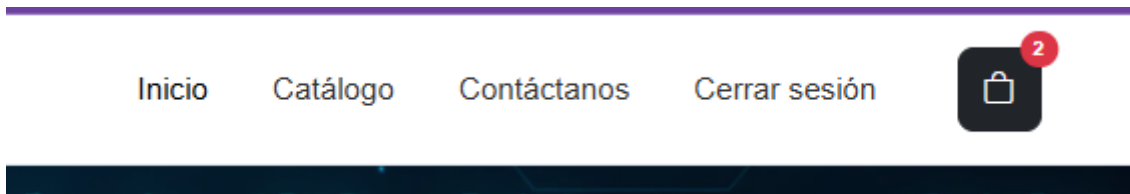
....

☐ Recuérdame [¿Olvidó su contraseña?](#)

Iniciar sesión

¿Aún no estás registrado? [Crear una cuenta](#)

Si inicia sesión correctamente, la opción cerrar sesión aparece en el navbar, administrado por auth.js.

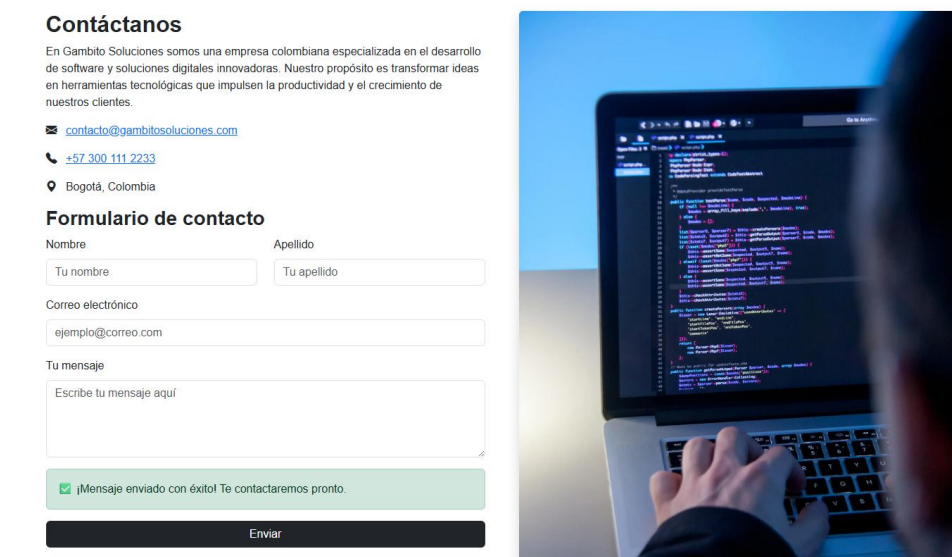
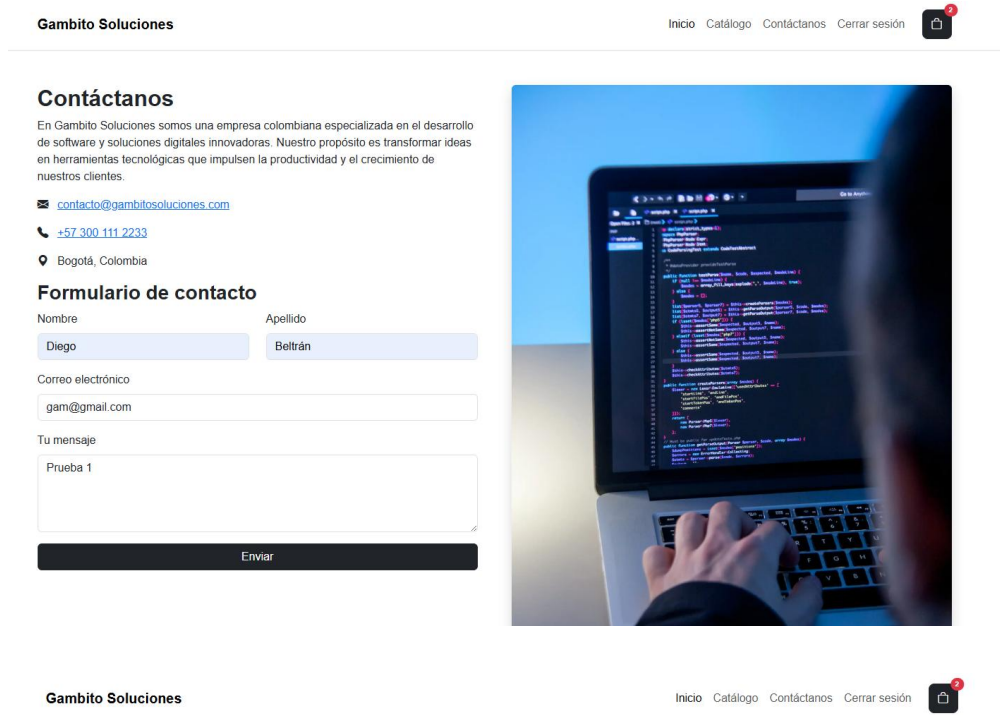


6.1.5. Sección “Acerca de” y Formulario de Contacto (acerca.html)

Contiene un formulario donde el usuario puede enviar un mensaje.

Los datos se almacenan localmente mediante contact.js, simulando un envío exitoso.

Luego, estos mensajes se muestran en la bandeja de mensajes (bandeja.html).





6.1.6. Bandeja de Mensajes (bandeja.html)

Utiliza el script inbox.js para leer los mensajes guardados por contact.js y mostrarlos en una tabla dinámica.

De esta forma, se simula un panel administrativo de contacto.

Enlace: <https://diegogamboa25.github.io/Entregable-2-frontend-/bandeja.html>

 Bandeja de mensajes recibidos			
Fecha	Nombre	Email	Mensaje
6/10/2025, 9:51:51 p. m.	Diego Beltrán	gam@gmail.com	Prueba 1
 Borrar todos los mensajes			

6.1.7. Navegación Dinámica y Navbar Persistente (main.js)

El navbar y footer se cargan de manera dinámica en todas las páginas mediante fetch().

Gracias a esto:

- El contador del carrito (updateCartCount()) se mantiene sincronizado.
- Se conserva la sesión activa.
- El acceso al carrito y la bandeja siempre está visible.
-

Gambito Soluciones	Inicio	Catálogo	Contáctanos	Cerrar sesión	
--------------------	--------	----------	-------------	---------------	---

6.1.8. Comportamiento Técnico

Evento	Acción ejecutada	Script responsable
Click en “Agregar al carrito”	Agrega producto a LocalStorage y actualiza contador	cart.js
Cargar página del carrito	Renderiza productos almacenados	cart.js
Click en “Vaciar carrito”	Limpia LocalStorage y actualiza vista	cart.js
Envío de formulario de contacto	Guarda mensaje en LocalStorage	contact.js
Carga de bandeja	Recupera mensajes guardados y los muestra	inbox.js
Inicio de sesión	Verifica usuario y guarda sesión activa	login.js
Cierre de sesión	Borra datos de sesión	auth.js
Carga de página	Inserta navbar y footer, detecta entorno GitHub/local	main.js

6.1.9. Resumen del Flujo

- El usuario inicia en index.html.
- Se mueve hacia el catálogo, donde puede interactuar con los productos.
- Añade ítems al carrito, que se mantiene sincronizado en todo el sitio.
- Puede iniciar sesión o enviar mensajes desde la vista acerca de.
- Los mensajes quedan disponibles en bandeja.html como si fuera un panel de administración.
- Finalmente, puede realizar una simulación de compra, mostrando una experiencia completa sin backend.

7. Mantenimiento y actualizaciones

El sistema Gambito Soluciones – Entregable Final Frontend está diseñado con una estructura modular y componentes reutilizables, lo que facilita su mantenimiento y evolución sin afectar la estabilidad del proyecto.

Dado que se basa en tecnologías de cliente (HTML, CSS, JavaScript), su mantenimiento se centra principalmente en tres áreas: actualización de contenido, ajustes funcionales y optimización visual.

7.1. Mantenimiento Preventivo

Este tipo de mantenimiento busca prevenir errores futuros y asegurar la compatibilidad del proyecto con nuevas versiones de navegadores o librerías.

Tarea	Descripción	Frecuencia sugerida
Verificar integridad de enlaces	Asegurar que los enlaces a scripts, imágenes y páginas sean válidos (especialmente al desplegar en GitHub Pages).	Mensual
Comprobar compatibilidad de Bootstrap y dependencias	Revisar si existen versiones actualizadas de Bootstrap o iconos (bootstrap-icons).	Cada 3 meses
Limpiar LocalStorage	Eliminar datos obsoletos o de pruebas que puedan generar errores en el flujo.	Según pruebas
Validar carga dinámica de componentes (main.js)	Confirmar que navbar y footer se rendericen correctamente tras actualizaciones del servidor.	Después de cada actualización

7.2. Mantenimiento Correctivo

Consiste en corregir errores detectados durante las pruebas o el uso real.

Ejemplos de ajustes frecuentes en este sistema incluyen:

- Corrección de errores de lógica en los scripts (cart.js, auth.js, login.js).
- Ajustes en los eventos de clic o envío de formularios (contact.js).
- Actualización de rutas en main.js si cambia el nombre de carpetas o archivos.
- Reparación de referencias a recursos eliminados (imágenes o estilos).
- Cada corrección debe probarse localmente antes de subir los cambios al repositorio remoto.

7.3. Mantenimiento Evolutivo

Implica añadir nuevas funciones o mejorar las existentes sin alterar la estabilidad de la aplicación.

Gracias al diseño modular, es posible ampliar la funcionalidad con facilidad. Ejemplos:

Nueva función posible	Modificación necesaria
Integrar autenticación real con API	Sustituir login.js y auth.js por peticiones fetch() hacia un backend real.
Envío de correos reales desde el formulario	Reemplazar contact.js por una API de envío (por ejemplo, EmailJS o backend PHP).
Conexión de carrito con base de datos	Sustituir la persistencia en localStorage por una API REST.
Estadísticas o panel administrativo real	Crear un dashboard y conectar inbox.js a un servidor que guarde mensajes.

7.4. Proceso de Actualización en GitHub Pages

Para mantener la aplicación actualizada en producción:

- 1. Actualizar los archivos locales.**
 - Modifica o agrega los cambios en el entorno local (VS Code o equivalente).
- 2. Verificar en entorno local.**
 - Prueba la funcionalidad en http://localhost con Live Server o XAMPP.
- 3. Subir los cambios al repositorio.**

```
git add .
git commit -m "Actualización de funciones o contenido"
git push origin main
```

- 4. Desplegar en GitHub Pages.**

- GitHub actualiza automáticamente la versión visible en <https://diegogamboa25.github.io/Entregable-2-frontend-/>

5. Validar funcionamiento online.

- Revisa que los enlaces y los scripts funcionen correctamente.

7.5. Buenas Prácticas Recomendadas

- Mantener comentarios explicativos en cada función para facilitar el mantenimiento futuro.
- Evitar el uso de rutas absolutas; siempre emplear rutas relativas o detección de entorno (main.js).
- Nombrar los archivos y variables de manera descriptiva y coherente.
- Respalidar el proyecto antes de aplicar grandes modificaciones.
- Probar cada módulo de forma independiente antes de integrar los cambios globales.

7.6. Actualización de Documentación

Cada vez que se modifique o agregue una funcionalidad, debe actualizarse:

- El presente Manual Técnico y Funcional.
- El archivo README.md del repositorio.
- Los diagramas UML si cambia la estructura lógica o el flujo de navegación.

Esto garantiza que la documentación se mantenga sincronizada con el estado real del sistema.

8. Especificación técnica de funciones (por módulo y archivo)

El sistema Gambito Soluciones está compuesto por diversos módulos JavaScript organizados de forma modular para separar la lógica de cada componente de la interfaz.

A continuación, se detalla la descripción técnica de todas las funciones implementadas en cada archivo:

8.1. Archivo: main.js

Propósito:

Carga de componentes dinámicos (navbar, footer), detección del entorno (local o GitHub Pages) y actualización del contador de carrito en todas las páginas.

Función	Descripción	Parámetros	Retorno
DOMContentLoaded	Evento que inicializa la carga dinámica del navbar y footer.		

Función	Descripción	Parámetros	Retorno
updateCartCount()	Calcula el total de productos en el carrito y actualiza el contador en el ícono del navbar.		Actualiza el elemento #cart-count del DOM.

8.2. Archivo: auth.js

Propósito:

Gestionar la sesión activa del usuario y modificar el enlace de inicio/cierre de sesión en el navbar.

Función	Descripción	Parámetros	Retorno
DOMContentLoaded	Detecta si hay un usuario guardado en localStorage. Si existe, muestra su nombre y cambia el enlace a "Cerrar sesión".		Actualiza el DOM.
loginLink.addEventListener("click")	Elimina los datos de sesión (localStorage.removeItem("usuario")) y redirige a index.html.	event	Redirección al inicio.

8.3. Archivo: login.js

Propósito:

Simular el proceso de autenticación con usuarios ficticios.

Función	Descripción	Parámetros	Retorno
DOMContentLoaded	Espera la carga del formulario y agrega un listener para capturar el evento de envío.		
form.addEventListener("submit")	Captura correo y contraseña, compara con los usuarios válidos y guarda los datos del usuario autenticado en localStorage.	event	Guarda usuario y redirige a index.html.

Usuarios simulados:

```
{ nombre: "Diego", email: "diego@example.com", password: "1234" },
{ nombre: "Admin", email: "admin@example.com", password: "admin" }
```

8.4. Archivo: cart.js

Propósito:

Administrar todas las operaciones relacionadas con el carrito de compras.

Función	Descripción	Parámetros	Retorno
getCart()	Obtiene los productos almacenados en localStorage.		Array de productos.
saveCart(cart)	Guarda el arreglo de productos en localStorage.	cart (array)	
addToCart(product)	Agrega un producto al carrito o aumenta su cantidad si ya existe.	product (objeto con id, name, price, image)	Muestra alerta y actualiza contador.
removeFromCart(productId)	Elimina un producto específico del carrito.	productId (número)	
clearCart()	Vacía completamente el carrito y actualiza la vista.		
renderCart()	Renderiza el contenido del carrito en la tabla HTML (carrito.html).		Actualiza el DOM dinámicamente.
updateCartCount()	Actualiza el contador de productos en el navbar.		
simulatePurchase()	Simula la compra mostrando confirmación, limpia el carrito y redirige al inicio.		Mensaje y redirección.

8.5. Archivo: contact.js

Propósito:

Simular el envío de mensajes desde el formulario de contacto y almacenarlos en localStorage.

Función	Descripción	Parámetros	Retorno
DOMContentLoaded	Captura el formulario y agrega un listener al evento de envío.		
form.addEventListener("submit")	Obtiene los datos (nombre, apellido, email, mensaje) y los guarda en localStorage bajo la clave "mensajes".	event	Mensaje de confirmación y limpieza del formulario.

8.6. Archivo: inbox.js

Propósito:

Mostrar los mensajes almacenados por contact.js en la bandeja de entrada

Función	Descripción	Parámetros	Retorno
DOMContentLoaded	Carga los mensajes guardados desde localStorage al cargar la página.		
renderInbox()	Recorre los mensajes y los inserta en una tabla HTML con nombre, correo y mensaje.		Actualiza el DOM.
clearInbox() (opcional)	Permite limpiar la bandeja de mensajes simulada.		Limpia localStorage.

8.7. Archivo: catalogo.html (interacción)

Propósito:

Permitir al usuario visualizar los productos disponibles e interactuar con el carrito mediante los botones Agregar al carrito.

Elemento	Descripción	Archivo vinculado
Botón .add-to-cart	Contiene los atributos data-id, data-name, data-price, data-image usados por cart.js.	cart.js

8.8. Archivo: bandeja.html

Propósito:

Mostrar los mensajes enviados desde el formulario de contacto (como panel administrativo).

Elemento	Descripción	Archivo vinculado
Tabla #inbox-messages	Contenedor dinámico donde se cargan los mensajes almacenados.	inbox.js
Botón “Borrar mensajes”	Permite vaciar el almacenamiento de mensajes.	inbox.js

8.9. Archivo: acerca.html

Propósito:

Contiene el formulario de contacto, gestionado por contact.js.

Elemento	Descripción	Archivo vinculado
Formulario <form> con campos #nombre, #apellido, #email, #mensaje	Captura los datos del usuario.	contact.js

8.9.1. Resumen General

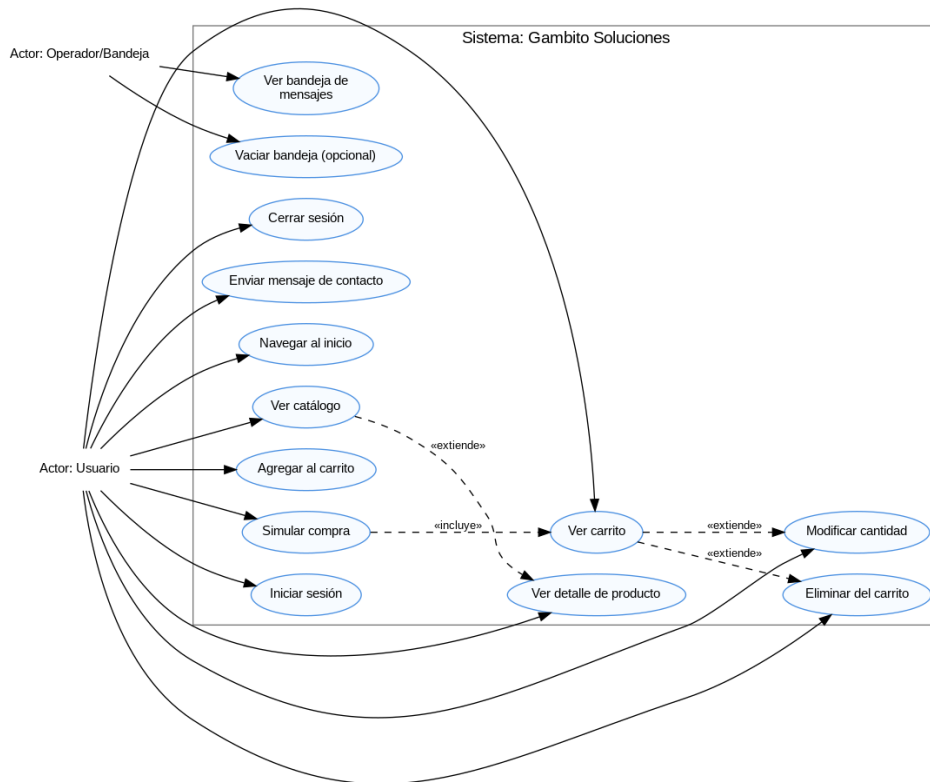
Módulo	Tipo	Funcionalidad principal
main.js	Control global	Carga dinámica y contador de carrito
auth.js	Autenticación	Control de sesión y cierre
login.js	Lógica de login	Validación de usuario
cart.js	E-commerce	Gestión del carrito
contact.js	Comunicación	Simulación de formulario
inbox.js	Administración	Visualización de mensajes

9. Diagramas UML del Sistema

Los diagramas UML permiten representar gráficamente la estructura, comportamiento y flujo de información dentro del sistema Entregable Final – Gambito Soluciones. A continuación, se describen los diagramas elaborados con base en la funcionalidad real del proyecto.

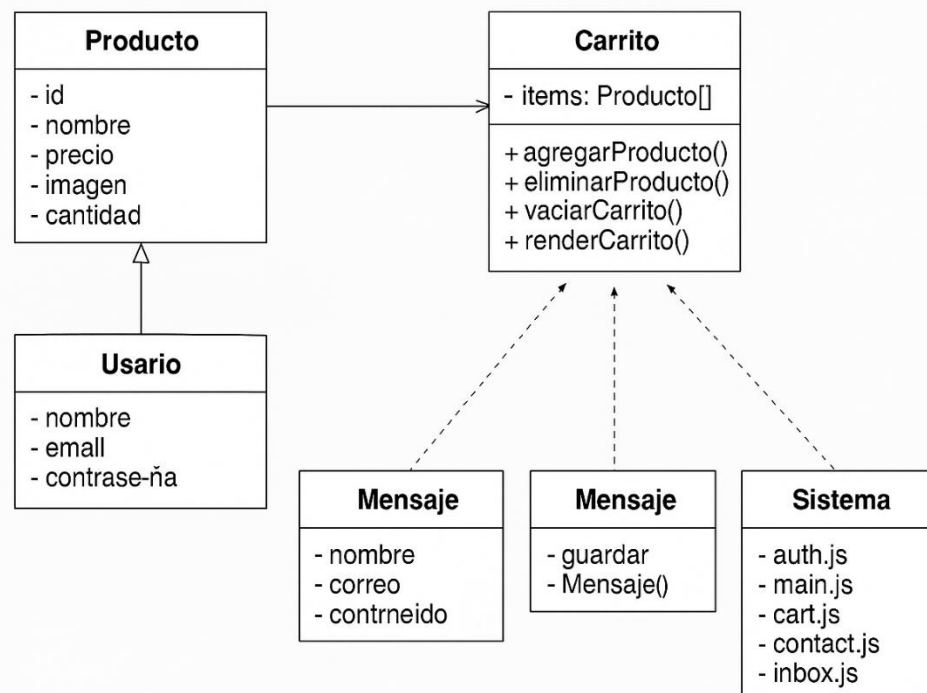
9.1. Diagrama de casos de uso

El diagrama de casos de uso muestra las interacciones entre los actores principales y las funcionalidades del sistema.



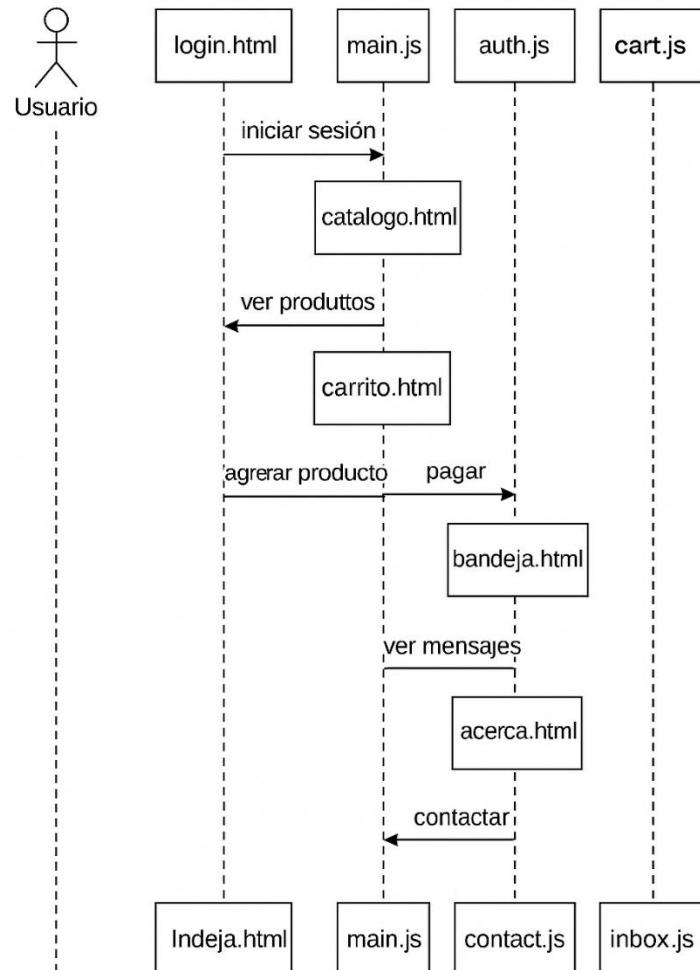
9.2. Diagrama de Clases

El diagrama de clases modela la estructura lógica de la aplicación, identificando los principales objetos JavaScript y sus relaciones.



9.3. Diagrama de Secuencia

Este diagrama muestra el flujo de interacción entre los actores, las vistas y los módulos del sistema durante una operación típica (por ejemplo, agregar un producto al carrito y simular una compra).

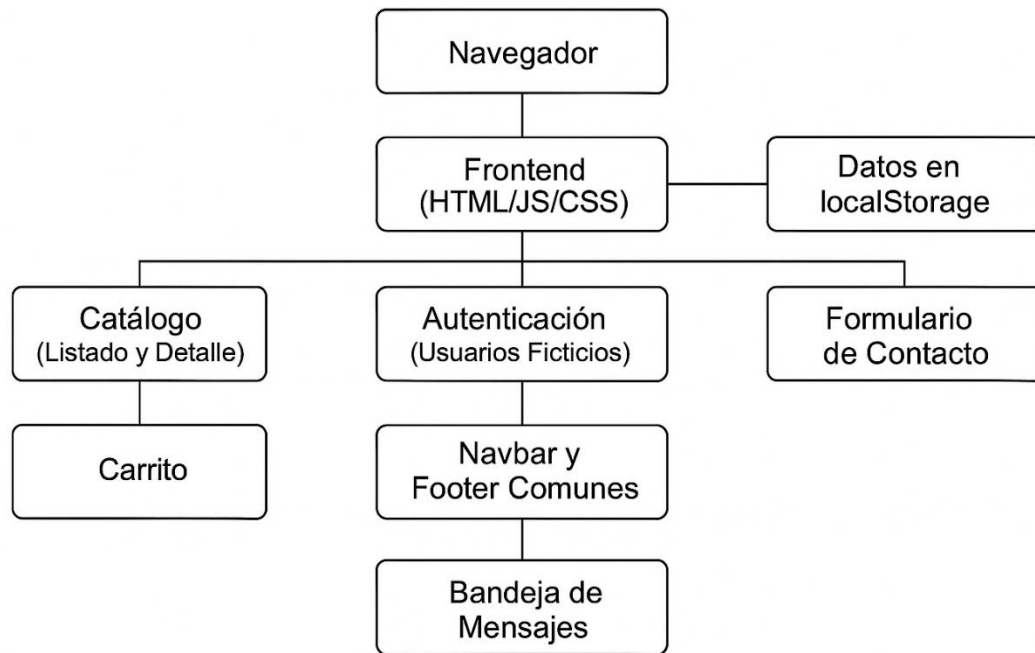


9.4. Diagrama de Arquitectura Funcional

Representa los componentes estructurales del sistema y su interacción.

El sistema está basado en una arquitectura frontend modular, donde cada archivo HTML actúa como una vista y los módulos JS manejan la lógica funcional.

Arquitectura Funcional del Sistema



9.5. Diagrama Complementario: Flujo de Navegación Funcional

Este diagrama muestra el recorrido de navegación del usuario dentro del sitio, desde el inicio hasta la simulación de compra y el envío de mensajes de contacto.

