UPN
UNIVERSIDAD PRIVADA DEL NORTE

FACULTAD DE INGENIERÍA

CARRERA PROFESIONAL DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

Título del trabajo:

Trabajo Grupal: Semana 2

Curso:

Soluciones web y aplicaciones distribuidas

Integrantes:

- ✓ Goicochea Saldaña, Joaquín Alejandro
- ✓ Pflücker Benites, Jesús Rodrigo
- ✓ Quipuzcoa Lopez, Yuleisy Anais
- ✓ Sánchez Boy, Ana Belén Del Pilar
- ✓ Zavaleta Perez, Flavio Alberto

Docente:

Edgar Vílchez Valdez

19 DE ABRIL, 2025

TRUJILLO - PERÚ

INDICE

I. INTRODUCCIÓN

En el desarrollo de soluciones web modernas, la combinación de buenas prácticas de programación, frameworks eficientes y medidas de seguridad resulta fundamental para garantizar aplicaciones escalables, funcionales y confiables. La creciente complejidad de los sistemas distribuidos exige que los desarrolladores dominen herramientas que faciliten tanto la construcción como el mantenimiento de los proyectos.

En este trabajo se abordan aspectos clave para el diseño y desarrollo de aplicaciones web. Primero, se presentan los patrones de diseño, los cuales proporcionan soluciones reutilizables a problemas comunes en la ingeniería de software, favoreciendo la estandarización y la calidad del código. Luego, se analiza el uso de Bootstrap, Material Design y SASS, tecnologías que optimizan la experiencia del usuario mediante interfaces consistentes, adaptables y fáciles de mantener. Asimismo, se estudian los métodos GET y POST, esenciales para la comunicación cliente-servidor bajo el protocolo HTTP. Finalmente, se examinan las principales vulnerabilidades web (SQL Injection, Cross-Site Scripting y CSRF), junto con recomendaciones de seguridad que permiten mitigar riesgos y proteger la integridad de los sistemas.

El propósito de este informe es ofrecer una visión integral sobre la importancia de aplicar patrones de diseño, utilizar frameworks de desarrollo y adoptar prácticas de seguridad, con el fin de fortalecer las competencias en la creación de aplicaciones web modernas y seguras.

II. PATRONES DE DISEÑO

2.1. ¿Qué son los patrones de diseño?

En el desarrollo de software son soluciones generalizadas y reutilizables para problemas recurrentes que se presentan en la construcción de sistemas. Un patrón no es un fragmento de código listo para copiar y pegar, sino una guía o esquema conceptual que puede adaptarse a distintos lenguajes de programación

Estos patrones surgieron como parte de la ingeniería de software orientada a objetos y buscan estandarizar la forma en que los desarrolladores enfrentan problemas comunes, promoviendo la reutilización, legibilidad y mantenibilidad del código

2.2. Importancia de los patrones de diseño

- Facilitar la comunicación entre desarrolladores, ya que brindan vocabulario común
- Favorecer el principio de reutilización al aplicar soluciones probadas

2.3. Fundamentos de Patrones de diseño

Los patrones de diseño se dividen en tres categorías principales

- Creacionales: Gestionan la creación de objetos de forma flexible.
- Estructurales: Organizan las clases y objetos para formar sistemas escalables.
- De comportamiento: Definen la comunicación y distribución de responsabilidades entre objetos.

Estos patrones son esenciales en programación moderna, ya que permiten reutilizar soluciones probadas, reducir errores y mejorar la comunicación entre desarrolladores al emplear un lenguaje común (Freeman&Freeman,2004).

2.4. Patrón Singleton

El patrón singleton asegura que una clase tenga una única instancia en todo el sistema y proporciona un punto de acceso global a ella. Es útil en contextos donde se requiere una solo fuente de verdad, como gestores de configuración, logs o conexiones a base de datos.

- Un constructor privado es necesario para evitar instanciación directa.
- Una variable estática que almacena la instancia única.
- Un método público y estático que devuelve la instancia.

VENTAJAS

- Control de acceso a una única instancia.
- Uso eficiente de recursos en entornos distribuidos.

DESVENTAJAS

- Puede dificultar pruebas unitarias por su naturaleza global.
- Riesgo de convertirse en un antipatrón si se abusa de él.

2.5. Patrón Factory

El patrón Factory define una interfaz para crear objetos, pero delega la decisión de instanciación a las subclases. Es útil cuando se necesita desacoplar la creación de objetos de su uso, como en sistemas de mensajería, servicios web o manejo de múltiples formatos de datos.

- Una clase abstracta o interfaz con un método de creación.
- Subclases concretas que implementan la creación de objetos.

VENTAJAS

- Desacopla la creación de objetos del código cliente.
- Facilita la extensión sin modificar el sistema existente.

DESVENTAJAS

- Puede incrementar la complejidad del diseño.

2.6. Patrón Decorator

El patrón Decorator permite añadir dinámicamente funcionalidades adicionales a un objeto sin modificar su estructura original. Es ideal en sistemas web para implementar características como seguridad, logging o compresión de datos.

- Una interfaz o clase de base común
- Un objeto “Componente” que se envuelve
- Decoradores que añaden funcionalidades extendiendo el comportamiento

VENTAJAS

- Flexibilidad para añadir funciones sin modificar código existente.
- Promueve el principio abierto/cerrado.

DESVENTAJAS

- Puede incrementar la complejidad si se encadenan demasiados decoradores.

III. USO DEL FRAMEWORK BOOTSTRAP

3.1. ¿Qué es Bootstrap?

Bootstrap es un framework de código abierto para desarrollo web front-end, compuesto por HTML, CSS y JavaScript, que facilita la creación de interfaces de usuario adaptables a distintos dispositivos mediante el uso de componentes predefinidos como menús, botones, cuadros, formularios y ventanas modales. Esta característica, conocida como Responsive Design, permite que los elementos de la página se ajusten automáticamente a las proporciones y resoluciones de la pantalla del usuario, mejorando la experiencia de navegación.

El origen de Bootstrap se remonta a 2011, cuando Mark Otto y Jacob Thornton, de Twitter, desarrollaron un conjunto de herramientas llamado Blueprint como solución interna para estandarizar el diseño de sus

proyectos web. Con el tiempo, Blueprint se liberó como un proyecto de código abierto en GitHub y evolucionó hasta convertirse en el framework que hoy conocemos como Bootstrap (Quisaguano Collaguazo et al., 2024).

En el contexto del desarrollo web, existen diversos frameworks CSS que facilitan el proceso de maquetación y estilización de sitios web. Entre los más populares se encuentran **Tailwind CSS, Bulma, Foundation, Pure CSS** y **Bootstrap**, que destaca por su robustez e intuición para crear interfaces web amigables con el usuario y de rápida implementación (Hernández, 2020).

Bootstrap se ha consolidado como una herramienta ampliamente utilizada, especialmente por su capacidad de mejorar la consistencia y eficiencia en el desarrollo de aplicaciones web, permitiendo interfaces uniformes y adaptables desde cualquier dispositivo.

3.1.1. Características

Bootstrap se destaca por que facilitan el desarrollo web moderno según Hernández (2020) y Quisaguano Collaguazo (2024) presenta varias características que lo diferencian de otros frameworks CSS:

- **Adaptabilidad y compatibilidad:** Se ajusta automáticamente a diferentes navegadores y dispositivos, garantizando interfaces responsivas.
- **Componentes predefinidos:** Incluye botones, menús, formularios, cuadros, ventanas modales y otros elementos listos para usar.
- **Integración con JavaScript:** Compatible con librerías como jQuery y permite el uso de efectos dinámicos.
- **Uso de preprocesadores CSS:** Permite trabajar con Sass y Less para crear estilos más personalizados y mantener estándares de CSS.
- **Flexibilidad en el diseño:** Ofrece layouts fijos y fluidos, basados en un sistema de 12 columnas, adaptables según el tamaño de la pantalla.

- **Ligero y modular:** Su estructura es adaptable a distintos tipos de proyectos, desde sitios sencillos hasta aplicaciones web complejas.

3.1.2. Ventajas y Desventajas de Bootstrap

VENTAJAS	DESVENTAJAS
Gran soporte de comunidad y documentación abundante.	Aprendizaje inicial: es necesario familiarizarse con la estructura y nomenclatura de clases.
Rapidez y comodidad para construir sitios web e interfaces.	Mantenimiento: cambiar de versión puede ser complicado.
Amplia variedad de componentes, temas y plantillas listos para usar.	Limitaciones al ampliar o personalizar ciertos componentes sin afectar el diseño.
Permite crear aplicaciones responsivas de forma intuitiva.	Peso del framework puede afectar el tiempo de carga en algunas aplicaciones.
Compatibilidad con HTML5, CSS3 y JavaScript moderno.	Diseño basado en un grid de 12 columnas que debe adaptarse cuidadosamente según el dispositivo.

Fuente: (Hernández, 2020. https://n9.cl/sixd5w)

3.2. Instalación y configuración básica

Bootstrap ofrece diversas formas de instalación que permiten adaptarse a diferentes tipos de proyectos web, desde prototipos rápidos hasta aplicaciones profesionales. La documentación oficial de Bootstrap describe principalmente tres métodos de instalación:

a. Uso de CDN (Content Delivery Network):

Es el método más sencillo y rápido, ya que consiste en incluir un enlace externo al archivo CSS y JS de Bootstrap directamente en el documento HTML. Este enfoque evita la descarga de archivos locales y asegura que siempre se use una versión optimizada.

b. **Descarga de archivos locales:**

Permite descargar los archivos de Bootstrap (CSS y JS) desde su página oficial para utilizarlos en un proyecto sin depender de internet. Es útil en entornos restringidos o cuando se desea mayor control de las versiones.

c. **Instalación con gestores de paquetes:**

Esta alternativa está dirigida a proyectos de desarrollo más avanzados, donde se requiere integrar Bootstrap con frameworks modernos como Angular, React o Vue. Ofrece un control más preciso de las dependencias y facilita las actualizaciones.

Ejemplo de Configuración Básica

A continuación, se muestra la estructura mínima de un documento HTML que incluye Bootstrap mediante **CDN**

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.7/dist/css/bootstrap.min.css" rel="
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.7/dist/js/bootstrap.bundle.min.js"
  </body>
</html>
```

3.3. Aplicación de estilos con Bootstrap

La implementación de Bootstrap mejora la apariencia de un sitio web al proporcionar componentes y estilos predefinidos que garantizan un diseño limpio, moderno y consistente. Elementos como botones, formularios, tarjetas y menús de navegación adquieren una estética uniforme sin necesidad de escribir CSS adicional, lo que facilita que el sitio luzca profesional y atractivo para los usuarios.

Además, Bootstrap optimiza la experiencia de usuario (UX) al ofrecer un diseño responsivo que se adapta automáticamente a distintos dispositivos y tamaños de pantalla. Esto permite que el contenido sea accesible y legible, mejorando la navegación y la interacción. Para los desarrolladores, representa una reducción del tiempo de implementación y errores de diseño, mientras que los clientes se benefician de un producto visualmente consistente y funcional (Bootstrap, 2023).

Categoría	Componente / Clase	Función / Aplicación	Notas de personalización
Botones	.btn, .btn-primary	Crear botones con estilo predefinido	Se puede cambiar color (.btn-success, .btn-danger), tamaño (.btn-lg, .btn-sm), forma (.rounded)
Formularios	.form-control, .form-group, .form-check	Dar estilo a campos de entrada, agrupaciones y casillas de verificación	Se pueden aplicar validaciones y feedback visual
Tarjetas (Cards)	.card, .card-body	Mostrar contenido en bloques organizados	Se puede agregar imágenes, listas, botones, enlaces
Barras de navegación	.navbar, .navbar-expand-lg, .navbar-light	Crear menús responsivos adaptables a diferentes pantallas	Se pueden cambiar colores, agregar dropdowns
Alertas	.alert, .alert-success	Mostrar mensajes de notificación o advertencias	Colores (.alert-danger, .alert-warning), dismissible

Acordeones	.accordion, .accordion-item	Mostrar contenido expandible y colapsable	Se pueden combinar con cards y estilos personalizados
Tablas	.table, .table-striped, .table-hover	Dar estilo a tablas de datos	Se pueden añadir sombreado, bordes, y responsive
Grid / Layout	.row, .col-*	Organizar contenido en filas y columnas adaptables	Combinable con breakpoints (.col-md-6, .col-lg-4)
Imágenes	.img-fluid, .rounded, .rounded-circle	Hacer imágenes responsivas y darles bordes o formas circulares	Combinable con bordes, sombras, efectos de hover
Tipografía	.text-center, .fw-bold, .fst-italic	Aplicar alineación, peso de fuente, cursiva	Combinable entre sí para estilos más complejos
Espaciado	.m-*, .p-*	Controlar márgenes y padding de elementos	Se puede usar con breakpoints (.mt-lg-3)
Modales	.modal, .modal-dialog	Crear ventanas emergentes o pop-ups	Se pueden animar, dimensionar y personalizar el contenido

Fuente: Adaptado de Bootstrap Official Documentation. (2023)

3.4. ¿Qué es Material Design?

Material Design es un lenguaje de diseño desarrollado por Google en 2014 bajo la dirección de Matías Duarte. Su nombre en clave inicial fue “Quantum Paper”, y nació con el propósito de crear un lenguaje visual que sintetizara los principios clásicos del buen diseño con las posibilidades de la innovación tecnológica y científica. Este sistema surgió como una respuesta a las limitaciones de las dos tendencias dominantes de la época: el esqueumorfismo, que al imitar de forma detallada objetos físicos resultaba recargado y poco funcional, y el diseño plano, que en su afán

de simplicidad eliminaba la profundidad y la jerarquía visual necesarias para orientar al usuario. Como señala Moran (2017), “cuando los diseñadores aplanan la interfaz de usuario, tienden a eliminar muchos indicadores que normalmente indican a los usuarios dónde hacer clic” (párr. 4). Frente a estos extremos, Material Design se presenta como un punto de equilibrio, conservando la claridad y simplicidad del diseño plano, pero incorporando sutiles elementos de profundidad, sombras y materialidad que mejoran la comprensión y la experiencia de navegación.

3.4.1. **Características**

Material Design se distingue por un conjunto de características que buscan generar interfaces claras, intuitivas y coherentes en distintas plataformas digitales. Entre las más importantes se encuentran:

- Minimalismo y colores planos: predomina el uso de colores vivos y tipografía clara (Roboto), manteniendo un orden jerárquico en los elementos. La premisa es “menos es más”, priorizando la limpieza visual y la facilidad de acceso a la información.
- Sombreado simulando niveles: los elementos usan sombras naturales para transmitir profundidad y jerarquías en la interfaz, simulando un efecto tridimensional.
- Iconos vectoriales: los íconos simples y escalables son esenciales para evitar pixelación y mantener claridad en diferentes dispositivos (móviles, tablets o computadoras).
- Animación de elementos: el movimiento es fundamental para guiar al usuario y hacer más dinámica la navegación, evitando depender únicamente de textos aclaratorios.
- Concepto espacial 3D: se incorporan los tres ejes (X, Y y Z), simulando la interacción con objetos reales como si fueran hojas de papel.
- Homogeneidad en todas las plataformas: Material Design asegura coherencia visual en distintas versiones de aplicaciones y sitios web, lo que facilita la experiencia del usuario y el desarrollo.

Como señala Perdomo (2017), “material design se caracteriza por seguir un estilo visual que simula un espacio real, para que el usuario se sienta cómodo y pueda interactuar con las diferentes plataformas de forma más intuitiva” (párr. 4).

3.4.2. Principios de Material Design

- El material es la metáfora

  Material Design se fundamenta en “la inspiración del mundo físico y sus texturas, incluyendo cómo los materiales reflejan la luz y proyectan sombras. Las superficies digitales reinventan los soportes tradicionales del papel y la tinta” (Material Design – Version 2, s.f., párr. 2) lo que genera una experiencia visualmente familiar, pero a la vez innovadora. En esta misma línea, Duarte, citado por la Interaction Design Foundation (2016), señala que “a diferencia del papel real, nuestro material digital puede expandirse y reformarse inteligentemente. El material tiene superficies y bordes físicos. Las costuras y las sombras dan significado a lo que se puede tocar” (párr. 3). Esta metáfora del “material digital” permite que los usuarios comprendan intuitivamente cómo interactuar con los elementos de la interfaz, trasladando aprendizajes de los objetos físicos al entorno virtual.

- Atrevido, gráfico, intencional

  Este principio supuso un cambio significativo en el diseño digital de la época. Con Material Design, ya no se trataba de seguir tendencias o preferencias estéticas, sino de diseñar con un propósito claro. El sistema desglosó pautas para elementos como tipografía, color, iconografía, navegación y movimiento, de modo que cada uno aportara valor real y no fuera decorativo.

  En este sentido, la coherencia y consistencia fueron claves, pues como señala Solier (2024), “las interfaces de usuario

deben seguir un conjunto de reglas y pautas predefinidas para asegurar que los usuarios puedan interactuar de manera intuitiva […]. La coherencia es esencial para proporcionar una experiencia de usuario satisfactoria y familiar" (párr. 4). Esto refuerza la idea de que cada elemento debía tener un propósito definido, evitando ambigüedades y mejorando la comunicación visual.

Lo "intencional" alude a diseñar con un motivo concreto: orientar, comunicar y facilitar la interacción. Lo "atrevido" y "gráfico" se expresan en colores intensos, tipografías claras y jerarquías visuales evidentes, que junto a imágenes y animaciones aportan contraste y profundidad (Davidov, 2021). En conjunto, este principio buscaba reforzar el mensaje y generar una experiencia clara, coherente y atractiva para el usuario.

- El movimiento proporciona significado

  En Material Design, el movimiento no se entiende como animación decorativa, sino como un recurso intencional para guiar y mejorar la experiencia del usuario. Este principio establece que las transiciones deben ser coherentes y sutiles, de modo que ayuden a mantener la continuidad espacial y temporal dentro de la interfaz. Así, cuando un elemento aparece, desaparece o se transforma, lo hace de manera que el usuario comprenda dónde estaba, hacia dónde se dirige y cómo se reorganiza el entorno digital.

  Existen tres razones fundamentales por las que el movimiento es clave en Material Design, en palabras de Davidov (2021):

  - ✓ Para ser informativo y permitir que los usuarios sepan dónde y cuándo estuvo disponible la acción.
  - ✓ Para ayudar a enfocar y guiar a los usuarios hacia las partes más importantes de la página.

- ✓ Para ser expresivo y añadir un poco de personalidad y alegría a la interfaz.

De esta manera, incluso pequeños detalles —como el icono de un botón que se expande suavemente al pulsarlo— se convierten en señales visuales que transmiten retroalimentación inmediata, reducen la incertidumbre y refuerzan la lógica de navegación. El movimiento, por tanto, añade significado y continuidad, creando interfaces más intuitivas y dinámicas.

3.5. ¿Qué es SASS?

SASS significa “Syntactically Awesome Style Sheets”, es un preprocesador CSS que extiende el lenguaje CSS añadiendo características que son propias de los lenguajes de programación, como pueden ser variables, funciones, selectores anidados, herencia, etc. (Pérez, 2019). Su objetivo principal es proporcionar herramientas que permitan escribir hojas de estilo más mantenibles y eficientes, reduciendo la repetición de código y mejorando la organización de este.

Como explica Nguyen (2020), "usando SASS se podría ayudar a los programadores a escribir CSS mantenible y no repetitivo" (párr. 1). Este preprocesador debe ser compilado a CSS estándar antes de ser utilizado en páginas web, pero ofrece capacidades avanzadas que el CSS tradicional no posee.

3.5.1. Características de SASS

- ✓ Variables

Las variables en SASS permiten almacenar valores que pueden reutilizarse a lo largo de la hoja de estilos. Esto facilita el mantenimiento del código, ya que cambiar un color, tamaño o cualquier valor CSS solo requiere modificar la variable correspondiente en lugar de buscar y reemplazar múltiples instancias. Sass usa el $símbolo para convertir algo en una variable.

Ejemplo comparativo entre SCSS y CSS:

SCCS	CSS

`$font-stack: Helvetica, sans-serif;`
`$primary-color: #333;`
 	
`body {`
`  font: 100% $font-stack;`
`  color: $primary-color;`
`}`
	`body {`
	`  font: 100% Helvetica, sans-serif;`
	`  color: #333;`
	`}`

Fuente: SASS (s.f.)

- ✓ Anidación (Nesting)

La anidación es una característica distintiva que permite escribir selectores CSS de manera jerárquica, reflejando la estructura HTML. Como indica la documentación oficial de SASS, "la anidación de selectores ul, li y a dentro del selector nav es una excelente manera de organizar el CSS y hacerlo más legible".

Ejemplo comparativo entre SCSS y CSS:

SCCS	CSS

`nav {`
`  ul {`
`  margin: 0;`
`  padding: 0;`
`  list-style: none;`
`  }`
 	
`  li { display: inline-block; }`
 	
`  a {`
`  display: block;`
`  padding: 6px 12px;`
`  text-decoration: none;`
`  }`
`}`
	`nav ul {`
	`  margin: 0;`
	`  padding: 0;`
	`  list-style: none;`
	`}`
	`nav li {`
	`  display: inline-block;`
	`}`
	`nav a {`
	`  display: block;`
	`  padding: 6px 12px;`
	`  text-decoration: none;`
	`}`

Fuente: SASS (s.f.)

- ✓ Mixins

Los mixins en Sass son bloques de código reutilizable que permiten agrupar declaraciones CSS y usarlas en distintas partes del proyecto sin repetir código. Se definen con @mixin y

se aplican con @include, lo que facilita la escritura de propiedades complejas como las de CSS3 con múltiples prefijos de proveedores. Una de sus mayores ventajas es que pueden recibir variables con $ para generar estilos dinámicos y flexibles. Como explica Dhivyaa (2023), “los mixins son para Sass lo que las funciones son para JavaScript. Nos permiten generalizar las clases aún más al recibir argumentos” (párr. 7).

Ejemplo comparativo entre SCSS y CSS:

SCCS

```scss
@mixin theme($theme: DarkGray) {
  background: $theme;
  box-shadow: 0 0 1px rgba($theme, .25);
  color: #fff;
}

.info {
  @include theme;
}
.alert {
  @include theme($theme: DarkRed);
}
.success {
  @include theme($theme: DarkGreen);
}
```

CSS

```css
.info {
  background: DarkGray;
  box-shadow: 0 0 1px rgba(169, 169, 169, 0.25);
  color: #fff;
}

.alert {
  background: DarkRed;
  box-shadow: 0 0 1px rgba(139, 0, 0, 0.25);
  color: #fff;
}

.success {
  background: DarkGreen;
  box-shadow: 0 0 1px rgba(0, 100, 0, 0.25);
  color: #fff;
}
```

Fuente: SASS (s.f.)

- ✓ Partials y módulos

En Sass, la modularidad del código se logra a través de los partials y los módulos. Los parciales son archivos Sass cuyo nombre comienza con un guion bajo (por ejemplo, _botones.scss), lo que indica que contienen fragmentos de estilos reutilizables y no generan directamente un archivo CSS; su función es dividir y organizar el código para facilitar el mantenimiento. Esto permite estructurar los estilos en fragmentos específicos —como tipografía, botones o formularios— que luego se integran en un archivo principal, lo cual resulta especialmente útil en proyectos grandes (MarcosNew, 2024). A su vez, estos parciales se gestionan

mediante la regla @use, que los convierte en módulos, asignándoles un espacio de nombres para acceder a sus variables, mixins y funciones, garantizando un código más limpio y estructurado (Sass, s.f.).

Ejemplo comparativo entre SCSS y CSS:

SCCS

```
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

CSS

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}

.inverse {
  background-color: #333;
  color: white;
}
```

Fuente: SASS (s.f.)

- ✓ Extends/Herencia

La directiva @extend en Sass permite que un selector herede el conjunto de propiedades de otro, lo que evita la repetición de código y facilita el mantenimiento de los estilos. Para ello, Sass utiliza las clases de marcador de posición (definidas con %), las cuales no generan CSS propio, sino que se imprimen únicamente cuando son extendidas. Por ejemplo, al definir %message-shared con estilos comunes y extenderlo en .message, .success, .error y .warning, todas estas clases comparten las mismas propiedades sin necesidad de reescribirlas, manteniendo el CSS limpio y ordenado (Sass, s.f.).

Ejemplo comparativo entre SCSS y CSS:

SCCS

```scss
/* This CSS will print because %message-shared is extended. */
%message-shared
  border: 1px solid #ccc
  padding: 10px
  color: #333

// This CSS won't print because %equal-heights is never extended.
%equal-heights
  display: flex
  flex-wrap: wrap

.message
  @extend %message-shared

.success
  @extend %message-shared
  border-color: green

.error
  @extend %message-shared
  border-color: red

.warning
  @extend %message-shared
  border-color: yellow
```

CSS

```css
/* This CSS will print because %message-shared is extended. */
.warning, .error, .success, .message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}
```

Fuente: SASS (s.f.)

- ✓ Operadores

Sass proporciona una variedad de operadores que permiten realizar cálculos y comparaciones dentro de las hojas de estilo, lo que facilita la creación de reglas dinámicas y reutilizables (GeeksforGeeks, 2023). Estos operadores incluyen tanto los matemáticos tradicionales como otros aplicables a valores lógicos, cadenas y colores.

Orden de operaciones:

1. Los operadores unarios: NOT, +, - y /.
2. Los operadores /, * y %.
3. Los operadores + y -.
4. Los operadores <, <=, > y >=.
5. Los operadores == y !=.
6. El operador AND.

7. El operador OR.
8. El operador = cuando esté disponible.

Ejemplo comparativo entre SCSS y CSS:

SCCS

```
@use "sass:math";

.container {
  display: flex;
}

article[role="main"] {
  width: math.div(600px, 960px) * 100%;
}

aside[role="complementary"] {
  width: math.div(300px, 960px) * 100%;
  margin-left: auto;
}
```

CSS

```
.container {
  display: flex;
}

article[role=main] {
  width: 62.5%;
}

aside[role=complementary] {
  width: 31.25%;
  margin-left: auto;
}
```

Fuente: SASS (s.f.)

Nota: El ejemplo en Sass usa el módulo math para calcular los anchos de los elementos en función de sus proporciones (600px/960px y 300px/960px). Esto evita escribir los valores porcentuales manualmente, logrando un código más dinámico, preciso y fácil de mantener en comparación con CSS puro.

IV. MÉTODOS GET Y POST

4.1. GET

El método GET realiza una solicitud de datos a un recurso específico. Devuelve tanto la cabecera como el contenido, que puede estar en formatos como HTML, JSON, XML, imágenes o JavaScript. Semánticamente equivale a una consulta SELECT en SQL, aunque en ocasiones se usa de forma incorrecta para insertar, actualizar o eliminar datos. (Morales, s.f)

Ejemplo de un método GET en Java:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
```

En Java, el método doGet() es el encargado de procesar este tipo de solicitudes. NetBeans suele generar un método común processRequest(...) para manejar la lógica compartida. En este método, el servlet puede leer parámetros enviados en la URL mediante request.getParameter() y generar la respuesta con response.getWriter(), que se envía al navegador en el formato correspondiente. De esta manera, GET se utiliza principalmente para consultar y mostrar información dinámica.

4.2. POST

El método POST se utiliza para enviar datos al servidor a través del cuerpo (body) de la solicitud, a diferencia de GET, que los envía en la URL. El formato de los datos se define en la cabecera Content-Type, pudiendo ser texto, JSON, XML o incluso archivos. Semánticamente, POST se emplea para registrar información, de manera análoga a un INSERT en SQL, aunque también puede usarse en otras operaciones como actualizaciones, eliminaciones o carga de archivos. (Morales, s.f)

Ejemplo de un método POST en Java:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
```

En Java, el método doPost() generalmente recibe los datos de un formulario web mediante request.getParameter() y los envía a una lógica de negocio o base de datos. A diferencia de GET, POST no es idempotente, lo que significa que cada petición puede producir un efecto distinto en el servidor. Por esta razón, es el método más usado para

operaciones que modifican datos o que requieren mayor seguridad en la transmisión de información.

4.3. DIFERENCIAS ENTRE GET Y POST

Característica	GET	POST
Propósito	Recuperar datos (sin modificar)	Enviar datos para crear o modificar recursos
Transmisión de datos	En URL (cadena de consulta))	En el cuerpo de la solicitud
Visibilidad de datos	En la URL → visible y registrada	Oculta de la URL → más segura
Limitaciones de longitud	Limitada por la longitud máxima de URL (~2048 caracteres en algunos navegadores)	Sin restricciones prácticas de longitud
Caché y marcadores	Cacheable, puede guardarse en historial y marcarse como favorito	No cacheable ni marcable
Seguridad	Menos seguro (datos en URL y expuestos en historial/logs)	Más seguro (datos en cuerpo, no en URL)
Idempotencia	Sí — repetir la solicitud no debe cambiar el estado del servidor	No — repetir puede generar efectos (ej. múltiples registros)
Tipología de datos	Solo datos simples (ASCII), longitud limitada	Datos complejos o binarios, formatos variados

V. VULNERABILIDADES Y RECOMENDACIONES DE SEGURIDAD

5.1. SQL Injection (SQLi)

SQL Injection es una vulnerabilidad que permite a un atacante interferir con las consultas que una aplicación realiza a su base de datos (Open Web Application Security Project, s.f.). Generalmente, esto se logra insertando una porción de código SQL malicioso a través de un campo de formulario no sanitizado, lo que permite al atacante manipular la consulta original. Esta vulnerabilidad sigue siendo una de las más críticas y se encuentra listada en el OWASP Top 10 bajo la categoría de Inyección (OWASP Foundation, 2021).

5.1.1. Vector de Ataque y Mecanismo

El ataque explota la concatenación insegura de entradas del usuario para construir sentencias SQL. Por ejemplo, una consulta de autenticación vulnerable podría ser:

```
SELECT id FROM usuarios WHERE username = ' ' + userInput + ' ' AND password = ' ' + userPass + '
';
```

Si un atacante introduce ' OR '1'='1 en el campo del nombre de usuario, la consulta resultante se convierte en:

```
SELECT id FROM usuarios WHERE username = ' ' OR '1'='1' ' AND password = '...';
```

La condición '1' = '1' siempre es verdadera, lo que podría permitir eludir el proceso de autenticación (Open Web Application Security Project, s.f.).

5.1.2. Impacto Potencial

- **Exposición de datos:** Lectura de información sensible, como credenciales de usuario, datos personales y registros financieros.
- **Manipulación de datos:** Modificación o eliminación de datos existentes en la base de datos.

- **Pérdida de control:** En casos graves, un atacante puede obtener control administrativo sobre el servidor de la base de datos, comprometiendo todo el sistema.

5.1.3. Estrategias de Prevención

- **Consultas Parametrizadas (Prepared Statements):** Es el método de defensa más eficaz. Consiste en definir la estructura de la consulta SQL primero y luego pasar las entradas del usuario como parámetros. De este modo, el motor de la base de datos trata la entrada como datos literales y no como código ejecutable, neutralizando el ataque (OWASP Foundation, 2021).
- **Validación de Entradas (Whitelist):** Se debe validar rigurosamente que los datos recibidos coincidan con el tipo, formato y longitud esperados. Por ejemplo, si se espera un ID numérico, se debe rechazar cualquier entrada que contenga caracteres no numéricos.
- **Principio de Mínimo Privilegio:** La cuenta de base de datos utilizada por la aplicación web debe tener únicamente los permisos indispensables para su funcionamiento, limitando el daño potencial si una vulnerabilidad es explotada.

5.2. Cross-Site Scripting (XSS)

El Cross-Site Scripting es un tipo de inyección en la que se introducen scripts maliciosos (comúnmente JavaScript) en páginas web que luego son vistas por otros usuarios. A diferencia del SQLi, que ataca al servidor, el XSS se dirige a los usuarios finales de la aplicación (PortSwigger, s.f.).

5.2.1. Vector de Ataque y Mecanismo

Un ataque XSS ocurre cuando una aplicación web toma datos no confiables de un formulario y los incluye en su respuesta HTML sin validarlos ni codificarlos correctamente (Open Web Application Security Project, s.f.).

- **XSS Reflejado:** El script malicioso se inyecta a través de una solicitud (por ejemplo, en un parámetro de URL) y se "refleja" de

vuelta en la respuesta del servidor, ejecutándose en el navegador de la víctima.

- **XSS Almacenado:** El script se guarda de forma permanente en el servidor (por ejemplo, en un comentario de un blog o un perfil de usuario) y se ejecuta cada vez que otro usuario carga la página afectada (PortSwigger, s.f.).

5.2.2. Impacto Potencial

- **Robo de Sesión:** El atacante puede robar las cookies de sesión de un usuario para suplantar su identidad.
- **Acciones no Autorizadas:** Ejecutar acciones en nombre del usuario, como cambiar su contraseña o realizar compras.
- **Phishing y Redirección:** Redirigir a los usuarios a sitios web fraudulentos para robar sus credenciales.

5.2.3. Estrategias de Prevención

- **Codificación de Salida (Output Encoding):** Es la defensa principal. Antes de renderizar cualquier dato proveniente del usuario en una página HTML, los caracteres especiales deben ser codificados a sus respectivas entidades HTML (p. ej., < se convierte en <). Esto asegura que el navegador interprete los datos como texto y no como código (Open Web Application Security Project, s.f.).

- **Política de Seguridad de Contenidos (CSP):** Es una cabecera HTTP que permite definir desde qué fuentes el navegador tiene permitido cargar recursos (como scripts). Una CSP estricta puede bloquear la ejecución de scripts inyectados (PortSwigger, s.f.).

- **Uso de Frameworks Modernos:** Frameworks como React, Angular o Vue.js aplican automáticamente la codificación de salida, reduciendo significativamente el riesgo de XSS si se utilizan correctamente.

5.3. Cross-Site Request Forgery (CSRF)

La Falsificación de Solicitudes entre Sitios es un ataque que obliga a un usuario autenticado a ejecutar acciones no deseadas en una aplicación web. El atacante aprovecha la confianza que el sitio tiene en el navegador del usuario (PortSwigger, s.f.).

5.3.1. Vector de Ataque y Mecanismo

El ataque se produce cuando un usuario está autenticado en un sitio vulnerable (ej., banco.com) y, al mismo tiempo, visita una página maliciosa. Esta página puede contener código (como un formulario oculto que se envía automáticamente) que envía una solicitud a banco.com. Como el navegador del usuario envía automáticamente las cookies de sesión con la solicitud, el sitio vulnerable la procesa como si fuera una acción legítima del usuario (Open Web Application Security Project, s.f.).

5.3.2. Impacto Potencial

- **Cambios de Estado no Autorizados:** Modificar la contraseña o el correo electrónico de la cuenta del usuario.
- **Transacciones Fraudulentas:** Realizar transferencias de fondos, compras u otras acciones críticas.
- **Compromiso de Datos:** Forzar al usuario a realizar acciones que comprometan la integridad de sus datos (PortSwigger, s.f.).

5.3.3. Estrategias de Prevención

- **Tokens Anti-CSRF (Patrón de Token Sincronizador):** Es la defensa más robusta. La aplicación genera un token único y secreto para cada sesión de usuario. Este token se incluye como un campo oculto en cada formulario que realiza acciones sensibles. Al recibir la solicitud, el servidor verifica que el token enviado coincida con el almacenado para esa sesión, invalidando las solicitudes que no lo contengan (Open Web Application Security Project, s.f.).

- **Atributo SameSite en Cookies:** Configurar las cookies de sesión con el atributo SameSite=Strict o SameSite=Lax instruye al navegador para que no envíe la cookie en solicitudes iniciadas desde sitios de terceros, mitigando eficazmente la mayoría de los ataques CSRF (PortSwigger, s.f.).
- **Verificación de Encabezados Origin y Referer:** Comprobar que estas cabeceras HTTP coinciden con el dominio de la aplicación puede ofrecer una capa de protección adicional, aunque no es infalible.

VI. CONCLUSIONES

- El uso de patrones de diseño en el desarrollo de software permite estructurar soluciones más claras, eficientes y reutilizables frente a problemas comunes. Estos patrones no solo favorecen la escalabilidad y mantenibilidad de los sistemas, sino que también mejoran la comunicación entre los desarrolladores al establecer un lenguaje común de diseño, contribuyendo así a proyectos más sólidos y organizados.
- Frameworks y herramientas como Bootstrap, Material Design y SASS han revolucionado la manera en que se construyen las interfaces web al ofrecer componentes reutilizables, estilos consistentes y opciones de personalización. Sin embargo, aunque aumentan la productividad y aceleran el desarrollo, también plantean retos como la posible falta de originalidad en los diseños o la sobrecarga de código, lo que exige un uso balanceado y consciente de estas tecnologías.
- Los métodos HTTP GET y POST, junto con la gestión adecuada de vulnerabilidades como SQL Injection, XSS y CSRF, son pilares fundamentales en la seguridad y el correcto funcionamiento de las aplicaciones web. Comprender sus diferencias y aplicar medidas de protección basadas en buenas prácticas, como el uso de sentencias preparadas, codificación de salida y tokens de verificación, resulta indispensable para garantizar la integridad, confidencialidad y confianza en los sistemas desarrollados.

VII. REFERENCIAS

Bootstrap Official Documentation. (2023). Getting Started with Bootstrap. Recuperado de https://getbootstrap.com/docs

Davidov, S. (2021, 2 febrero). *What Is Material Design and How Should It Be Used?* Blog de Elementor. https://elementor.com/blog/what-is-material-design/

DhivyaaM. (2023, 6 mayo). *SCSS: Defining Variables and Using Mixins — With Examples*. Medium. https://medium.com/@dhivyaa.nm/scss-defining-variables-and-using-mixins-with-examples-ba407cc30c4a

Freeman, E., & Freeman, E. (2004). *Head first design patterns*. O'Reilly Media. https://www.oreilly.com/library/view/head-first-design/9781492077992/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley. https://www.oreilly.com/library/view/design-patterns-elements/0201633612/

GeeksforGeeks. (2023, 10 abril). *SASS | Operators*. GeeksforGeeks. https://www.geeksforgeeks.org/css/sass-operators/

Hernández, E. (2020). Desarrollo de una aplicación web con el framework Bootstrap y el precompilador SASS para la gestión de pedidos de productos agrícolas de la Empresa El Chagra. Repositorio Escuela Superior Politécnica De Chimborazo. https://n9.cl/sixd5w

Interaction Design Foundation - IxDF. (2016, May 25). *What is Material Design?*. Interaction Design Foundation - IxDF. https://www.interaction-design.org/literature/topics/material-design

Material Design - Version 2. (s. f.). Material Design. https://m2.material.io/design/introduction#principles

Morales. (s,f) Métodos HTTP – POST, GET, PUT, DELETE. Estilo web. https://estilow3b.com/metodos-http-post-get-put-delete/#Referencias

Moran, K. (2017, 12 de marzo). *Flat-Design Best Practices*. Nielsen Norman Group. https://www.nngroup.com/articles/flat-design-best-practices/

Nguyen, L. (2020, 22 febrero). *Understand SASS/SCSS with Variables, Nesting, and Mixin principles.* Medium. https://medium.com/@lutnguyn_32958/understand-sass-scss-with-variables-nesting-and-mixin-principles-a52de07a6df0

Open Web Application Security Project (OWASP). (s.f.). Cross-site request forgery (CSRF). https://owasp.org/www-community/attacks/csrf

Open Web Application Security Project (OWASP). (s.f.). Cross-site scripting (XSS). https://owasp.org/www-community/attacks/xss/

Open Web Application Security Project (OWASP). (s.f.). SQL injection. https://owasp.org/www-community/attacks/SQL_Injection

Perdomo, S. (2017, 26 de abril). *¿Qué es el Material Design?* Deusto. https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/que-es-material-design

Pérez, J. (2019, 20 noviembre). Qué es Sass: ventajas, desventajas y ejemplos de desarrollo. *OpenWebinars.net*. https://openwebinars.net/blog/que-es-sass-ventajas-desventajas-y-ejemplos-de-desarrollo/

PortSwigger. (s.f.). Cross-site request forgery (CSRF). Web Security Academy. https://portswigger.net/web-security/csrf

Quisaguano Collaguazo, L. R., Lasluisa Alajo, A. J., Oto-UsuñoL, M., & Esquivel PaulaG, G. (2024). Optimización del desarrollo Front-end aplicando frameworks de diseño web: Bootstrap, Foundation y Bulma. Revista UTCiencia, 11(1), 17-31. https://investigacion.utc.edu.ec/index.php/utciencia/article/view/650

Sass. (s. f.). *Sass Basics*. Sass. https://sass-lang.com/guide/