UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

PWASP SCANNER – Sistema de Detección de Vulnerabilidades Web

Curso: *Patrones de Software*

Docente: Ing. Patrick Jose Cuadros Quiroga

Integrantes:

Ccalli Chata, Joel Robert ***(2017057528)***

Jarro Cachi, Jose Luis ***(2020067148)***

Tacna – Perú

2025

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	JCC	JPC	JCC	26/06/2025	Versión Original

PWASP SCANNER – Sistema de Detección de Vulnerabilidades Web
Documento – Estándar de Programación

Versión *1.0*

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	JCC	JCC	JCC	19/04/2025	Versión Original

ÍNDICE GENERAL

Introducción

Este documento define el **Estándar de Programación** adoptado para el desarrollo del sistema *PWASP SCANNER*, el cual tiene como objetivo mantener la calidad, consistencia, mantenibilidad y escalabilidad del código fuente a lo largo del ciclo de vida del proyecto.

Los estándares aquí definidos se aplican a todos los lenguajes, marcos y tecnologías utilizadas en el sistema, incluyendo el **frontend, backend, scripts de automatización y consultas SQL**.

Objetivos del Estándar

- Mejorar la **legibilidad** del código.
- Facilitar la **revisión entre pares** y auditoría técnica.
- Minimizar errores y **reducir deuda técnica**.
- Establecer **reglas uniformes** para todos los colaboradores.
- Garantizar la coherencia en el uso de **nombres, estructuras y estilos** de programación.

Tecnologías utilizadas

Módulo	Lenguaje / Framework
Backend	ASP.NET Core (C#)
Base de Datos	SQL Server
Frontend	HTML5, CSS3, JavaScript (con Bootstrap)
Chatbot	Integración con OpenAI API
Seguridad	OWASP, JWT Tokens

Estándares Generales

1. Nomenclatura

Elemento	Convención usada	Ejemplo
Clases	PascalCase	EscaneoController, Usuario
Métodos	PascalCase	RegistrarUsuario(), ValidarURL()
Variables	camelCase	idUsuario, urlEscaneada
Constantes	MAYÚSCULAS_CON_GUIONES	MAX_INTENTOS, TIMEOUT_API
Archivos	kebab-case o PascalCase	LoginService.cs, scan-result.component.ts

2. Estructura de Carpetas

swift
CopiarEditar

```
/PWASP-Scanner/
├── Controllers/
├── Models/
├── Views/
├── Services/
├── Repositories/
├── Utils/
├── Migrations/
└── Tests/
```

Cada carpeta representa una capa de responsabilidad clara. Se evita el uso de estructuras planas o mezclas de componentes de distintas capas.

3. Comentarios

Tipo de Comentario	Convención y Uso
De una línea	// Comentario descriptivo
De bloque	/* Comentario detallado */
XML/Docstring (C#)	/// <summary>Descripción</summary>
TODOs y FIXMEs	Usar etiquetas para seguimiento

Ejemplo:

```
csharp
CopiarEditar
/// <summary>
/// Registra un nuevo escaneo para el usuario.
/// </summary>
public void RegistrarEscaneo(string url) {
   // Validar que la URL no esté vacía
   ...
}
```

4. Buenas Prácticas

Buenas Prácticas	Descripción breve
Código limpio	Evitar funciones de más de 40 líneas. Dividir responsabilidades.
Control de versiones	Commits claros con mensajes del tipo: fix:, feat:, docs:, refactor:
Uso de excepciones	Capturar y manejar errores con try-catch, nunca suprimir excepciones.
Documentación	Todas las clases públicas deben tener comentarios XML o resumen.
Inyección de dependencias	Usar patrones como Dependency Injection para testabilidad.
Validaciones explícitas	Validar entradas del usuario y parámetros. No confiar en el frontend.
Principios SOLID	Aplicar siempre que sea posible.

Estándares por Lenguaje

A. C# (ASP.NET)

- Usar tipos explícitos: evitar var cuando el tipo no sea obvio.
- Aplicar patrón de repositorio y servicios.
- Nombrar interfaces con la letra I, ejemplo: IUsuarioService.
- No dejar métodos sin implementar sin razón válida.

csharp
CopiarEditar
```
public interface IScannerService {
    void EjecutarEscaneo(string url);
}
```

B. SQL Server

- Usar nombres claros y descriptivos.
- Comandos SQL en mayúsculas, objetos en minúscula.

sql
CopiarEditar
```
SELECT id_usuario, nombre_usuario
FROM usuarios
WHERE activo = 1;
```

- Evitar SELECT *.
- Aplicar INDEX en columnas de búsqueda frecuente.
- Usar transacciones para operaciones críticas.

C. JavaScript (Frontend)

- Usar let y const, evitar var.
- Evitar código inline en HTML (onclick, etc.).
- Todas las funciones deben tener nombres descriptivos.

javascript
CopiarEditar
```
function validarFormulario() {
  const url = document.getElementById("url").value;
  if (!url.startsWith("http")) {
    alert("URL no válida");
  }
}
```

D. HTML/CSS

- Usar etiquetas semánticas (<section>, <article>, <footer>).
- Estandarizar clases con BEM o similar.

html
CopiarEditar
```
<div class="scanner-result__item scanner-result__item--error">
  ...
</div>
```

- Separar lógica de estilos (no usar style="" inline).
- Usar Bootstrap con moderación, evitando sobrescribir clases directamente.

E. Manejo de Seguridad

Regla de Seguridad	Aplicación
Validación de entradas	En servidor y cliente
Prevención de XSS	Escapar salidas en HTML
SQL Injection	Usar parámetros en consultas
CSRF	Aplicar tokens de seguridad en formularios
Autenticación y autorización	Usar JWT, controlar sesiones, y proteger rutas sensibles

Herramientas de Soporte

Herramienta	Propósito
Visual Studio	Desarrollo backend
Git/GitHub	Control de versiones
Postman	Pruebas de APIs
SonarQube	Análisis estático de código
ESLint	Linter para JS
StyleCop	Validación de estilo C#

Política de Revisiones de Código

- Todo cambio debe pasar revisión por al menos **un desarrollador senior**.
- Se debe usar **Pull Requests** (PR) con comentarios.
- No se permiten commits directos a la rama main.

Conclusión

El cumplimiento de este estándar garantiza que el sistema *PWASP SCANNER* se desarrolle bajo buenas prácticas de ingeniería de software, asegurando un código mantenible, eficiente y seguro. Este documento debe actualizarse conforme evolucione el sistema o el stack tecnológico.