**Plataforma Web Interactiva para el Aprendizaje Autónomo del Piano con Reconocimiento de Notas Musicales - PianoRise**

**Documento de Estándares de Programación**

**Versión 1.0**

**Historia de Revisión**

| **Historial de revisiones** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Ítem** | **Fecha** | **Versión** | **Descripción** | **Equipo** | |
| 1 | 02/06/2025 | 1.0 | Versión completa. | Grupo 6 | |

**Tabla de Contenidos**

[1. OBJETIVO 4](#_heading=h.5aydytkq1r0c)

[2. DECLARACIÓN DE VARIABLES 4](#_heading=h.mubsvx2td4ps)

[2.1 PYTHON (FASTAPI / FLASK) – Módulo Alumno 4](#_heading=h.j9v3mbfnfniq)

[2.2 ASP.NET CORE MVC – Módulo Docente y Administrador 4](#_heading=h.81ihktlupddc)

[3. Definición de Rutas 5](#_heading=h.sm6uizyn3bhu)

[4. Definición de Servicios 5](#_heading=h.pgjq1dw39b9y)

[5. Definición de Controladores 5](#_heading=h.8drr3tf0v3f1)

[6. Beneficios 6](#_heading=h.zciyqcvzzjbe)

[7. Conclusiones 6](#_heading=h.r9gig79oyxcl)

**Estándares de Programación**

# OBJETIVO

Reglamentar la forma en que se implementa el código fuente del sistema PianoRise, compuesto por:

* El módulo Alumno, desarrollado en Python utilizando FastAPI y Flask.
* El módulo Docente y Administrador, desarrollado en ASP.NET Core MVC (C#).

Este estándar tiene como propósito mejorar y uniformizar el estilo de programación de todo el equipo, facilitando la comprensión, escalabilidad y mantenibilidad del código.

# DECLARACIÓN DE VARIABLES

## PYTHON (FASTAPI / FLASK) – Módulo Alumno

Convenciones de nombres:

* Variables, funciones, atributos, archivos y carpetas: snake\_case en minúsculas.
* Booleanos: deben llamarse activo, activa o iniciar con es\_, esta\_.
* Fechas: deben tener el prefijo fecha\_ (ejemplo: fecha\_creacion).
* Funciones o métodos: verbo + objeto (registrar\_usuario, calcular\_promedio).
* Longitud máxima para nombres de variables: 30 caracteres.
* Evitar caracteres especiales, acentos o la letra ñ.

## ASP.NET CORE MVC – Módulo Docente y Administrador

Convenciones de nombres:

* Clases, métodos, propiedades, archivos: PascalCase.
* Variables locales y parámetros: camelCase.
* Booleanos: EsActivo, EstaHabilitado.
* Fechas: prefijo Fecha (ej: FechaCreacion, FechaRegistro).
* Métodos: verbo + objeto (RegistrarUsuario(), ObtenerReporte()).
* Nombres claros, descriptivos y sin abreviaturas innecesarias.

# **Definición de** Rutas

* Nombre del archivo: usuario\_routes.py
* Prefijo del router: "/usuarios" (en plural)
* Nombre del router: usuario\_routes = APIRouter(...)
* Funciones: verbo + objeto (registrar\_usuario, login\_usuario, vincular\_aula)

EJEMPLO:

@usuario\_bp.route("/registro", methods=["GET", "POST"])

def registrar\_usuario():

...



# Definición de Servicios

* Archivo: usuario\_service.py
* Clase: UsuarioService
* Métodos descriptivos (register\_user, authenticate\_user)
* Validación dentro del servicio (campos, errores, fechas)

EJEMPLO:



class UsuarioService:

def register\_user(self, data: dict):

...



# Definición de Controladores

* Decorado con [Authorize(Roles = "...")] según rol
* Métodos Index, Details, Create, Edit, Delete
* Uso de ViewData, ModelState, validaciones y RedirectToAction
* Valida duplicados con AnyAsync(...)
* Código limpio, mensajes de error específicos

EJEMPLO:

public async Task<IActionResult> Create()

{

ViewData["DocenteId"] = new SelectList(...);

return View(new Aula { Estado = true });

}



# **Beneficios**

* Mayor legibilidad y consistencia: El uso de convenciones uniformes en nombres de archivos, funciones, variables y rutas permite que cualquier miembro del equipo pueda comprender rápidamente el flujo del código sin depender del autor original.
* Facilita la incorporación de nuevos desarrolladores: Un estándar claro actúa como guía de referencia para nuevos integrantes, reduciendo el tiempo de adaptación y errores en el desarrollo colaborativo.
* Soporte para herramientas automáticas: La adopción de estilos compatibles con herramientas como black, isort y linters como flake8 o pylint mejora la calidad del código automáticamente, detectando errores comunes y garantizando uniformidad en cada commit.
* Reduce ambigüedades y errores: Al seguir patrones definidos, se evita la duplicidad de lógica, la mala elección de nombres y la dispersión en la estructura del proyecto, lo que fortalece la integridad del backend.
* Mejor mantenimiento y escalabilidad: La claridad en la estructura de carpetas y la lógica modular de servicios y rutas facilita el mantenimiento del sistema y la incorporación de nuevas funcionalidades sin comprometer lo existente.

# **Conclusiones**

* Un estándar sólido y compartido por todo el equipo es esencial para garantizar la calidad, mantenibilidad y evolución del código fuente de PianoRise.
* Este conjunto de normas no solo orienta el desarrollo técnico diario, sino que también permite mantener una cultura de código limpia y coherente, especialmente útil en un entorno académico o de colaboración entre múltiples desarrolladores.
* El estándar debe ser dinámico: es recomendable revisarlo y actualizarlo periódicamente según las necesidades del proyecto y las lecciones aprendidas durante el desarrollo.
* Finalmente, este estándar no es solo una guía técnica, sino un pilar organizativo que permite que PianoRise crezca con solidez, transparencia y escalabilidad, tanto en su módulo de estudiantes como en el de docentes y administradores.