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

Estándares de Programación

1. 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.

2. DECLARACIÓN DE VARIABLES

2.1 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.

2.2 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.

3. 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:

```python
@usuario_bp.route("/registro", methods=["GET", "POST"])
def registrar_usuario():
    ...
```

4. Definición de Servicios

- Archivo: usuario_service.py
- Clase: UsuarioService
- Métodos descriptivos (register_user, authenticate_user)
- Validación dentro del servicio (campos, errores, fechas)

EJEMPLO:

```python
class UsuarioService:
    def register_user(self, data: dict):
       ...
```

5. 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:

```csharp
public async Task<IActionResult> Create()
{
    ViewData["DocenteId"] = new SelectList(...);
    return View(new Aula { Estado = true });
}
```

6. 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.

7. 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.