

Estándar de Programación

Lenguaje: C#

Framework: ASP.NET Core MVC

**Proyecto *“Sistema Web de Aprendizaje Autoguiado con IA para el Desarrollo de la Competencia “Gestiona proyectos de emprendimiento económico o social” del Área de Educación para el Trabajo en los Estudiantes del VI Ciclo de Educación Secundaria de la I.E. Marcelino Champagnat, Tacna***

***”***

Curso: *Construcción de Software I*

Docente: *Ing. ALBERTO JOHNATAN FLOR RODRIGUEZ*

Integrantes:

***Japura Quispe, herminia Aurelia (2018060912)***

***Concha Llaca, Gerardo Alejandro (2017057849)***

**

*El proyecto “Sistema Web para mejorar el aprendizaje de los estudiantes del VI ciclo de educación secundaria en el curso de Educación para el Trabajo” tiene como objetivo principal proporcionar una plataforma interactiva que facilite el acceso a recursos educativos, optimizando el proceso de enseñanza-aprendizaje. Este sistema está diseñado para ofrecer a los estudiantes herramientas digitales que potencien sus conocimientos y habilidades prácticas en el área de Educación para el Trabajo.*

*La implementación de este sistema web permite a los docentes gestionar materiales de aprendizaje, realizar evaluaciones en línea y dar seguimiento al progreso de los estudiantes. Por otro lado, los estudiantes podrán acceder a contenidos actualizados, realizar prácticas interactivas y recibir retroalimentación en tiempo real. El sistema se desarrolla utilizando tecnologías modernas como C# y ASP.NET Core, garantizando la robustez, escalabilidad y mantenibilidad.*

*Este documento refleja los estándares de codificación del lenguaje, no solo busca definir la nomenclatura de las variables, objetos, métodos y funciones, sino que también hace referencia al orden y legibilidad del código escrito.*

Estándar de Programación Lenguaje C#



# Nombres de los archivos

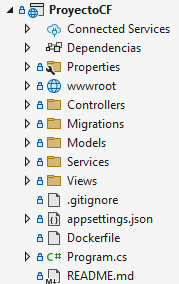
Los archivos de código fuente en C# deben seguir una convención clara y coherente para facilitar su identificación y mantenimiento.

# Extensiones de Archivos

| **Tipo de Archivo** | **Extensión** |
| --- | --- |
| Código fuente en C# | .cs |
| Archivos de proyecto | .csproj |
| Configuración de aplicación | .json, .xml |
| Archivos de vistas Razor, que combinan HTML con código C# para la generación dinámica de páginas web. | .cshtml |
| Hojas de estilo en cascada para la definición de la apariencia y el diseño de la interfaz de usuario. | .css |
| Archivos JavaScript para la lógica interactiva del lado del cliente. | .js |
| Archivos de configuración (appsettings.json, launchSettings.json) y para el almacenamiento de datos estructurados (ej. preguntas.json). | .json |
| Archivo de solución de Visual Studio que organiza uno o varios proyectos. | .sln |
| Archivos de documentos PDF, utilizados para materiales de curso y como fuente de conocimiento para el Chat IA. | .pdf |

# Organización de los archivos

Cada archivo debe contener una única clase pública o interfaz pública. Si existen clases auxiliares o privadas relacionadas, pueden incluirse en el mismo archivo, siempre que no excedan las 2000 líneas de código.



# Descripción de Carpetas

**Controllers:** Contiene los controladores que manejan las solicitudes HTTP y orquestan la lógica.

**Models:**

* **Entidades de Dominio:** Clases que representan las tablas de la base de datos y la lógica de negocio central (ej. Usuario.cs, Curso.cs).
* **ViewModels:** Clases para transferir datos entre controladores y vistas, adaptando el modelo de dominio a las necesidades de la UI (ej. CursoViewModel.cs, RendirEvaluacionViewModel.cs).

**Views:** Contiene las vistas Razor (.cshtml) que definen la interfaz de usuario.

**Services:** Contiene clases que encapsulan lógica de negocio específica o interactúan con servicios externos (ej. TelegramService.cs).

**Data (o similar):** Puede contener el DbContext (Connection.cs) y otras clases relacionadas con el acceso a datos.

**wwwroot:** Contiene archivos estáticos como CSS, JavaScript, imágenes y, en este proyecto, la carpeta uploads para archivos del Chat IA.

**json:** Contiene archivos JSON para datos estáticos como preguntas de la ruleta (preguntas.json).

# Estructura de un Archivo C#

El orden recomendado dentro de un archivo C# es:

1. Comentarios de inicio.
2. Directivas using.
3. Declaración del espacio de nombres (namespace).
4. Declaración de la clase o interfaz.
5. Campos estáticos.
6. Campos de instancia.
7. Constructores.
8. Propiedades.
9. Métodos (orden: públicos, protegidos, privados).

**Ejemplo Base de un Archivo C#**

/\*

\* Usuario.cs

\* Proyecto: Sistema Web Educativo

\* Autor: [Tu Nombre]

\* Fecha: [Fecha de Creación]

\* Descripción: Representa la entidad Usuario

\*/

using System.ComponentModel.DataAnnotations;

namespace Financiera.WebApp.Modelos

{

/// <summary>

/// Representa un usuario del sistema.

/// </summary>

public class Usuario

{

// Campos estáticos

// Campos de instancia

/// <summary>

/// Identificador único del usuario.

/// </summary>

[Key]

public int Id { get; set; }

/// <summary>

/// Nombre del usuario.

/// </summary>

[Required]

[MaxLength(100)]

public string Nombre { get; set; }

/// <summary>

/// Apellido del usuario.

/// </summary>

[Required]

[MaxLength(100)]

public string Apellido { get; set; }

/// <summary>

/// Correo electrónico del usuario.

/// </summary>

[Required]

[EmailAddress]

public string Email { get; set; }

/// <summary>

/// Contraseña del usuario (encriptada).

/// </summary>

[Required]

public string Contraseña { get; set; }

/// <summary>

/// Rol del usuario (Admin, Docente, Estudiante).

/// </summary>

[Required]

public string Rol { get; set; }

/// <summary>

/// Constructor por defecto.

/// </summary>

public Usuario() { }

/// <summary>

/// Constructor con parámetros.

/// </summary>

public Usuario(string nombre, string apellido, string email, string contraseña, string rol)

{

Nombre = nombre;

Apellido = apellido;

Email = email;

Contraseña = contraseña;

Rol = rol;

}

}

}

# Convenciones de Nomenclatura

**Clases, Interfaces, Métodos Públicos, Propiedades Públicas, Eventos:** PascalCase

* Ejemplos: Usuario, CursosController, IUserService, Login(), Nombre, CalcularPuntaje().

**Variables Locales y Parámetros de Métodos:** camelCase

* Ejemplos: usuarioId, email, requestBody.

**Campos Privados (Fields):** camelCase prefijado con un guion bajo (\_).

* Ejemplos: \_context, \_logger, \_telegramService.

**Constantes (const, static readonly):** PascalCase. Para constantes de configuración o de tipo "magic string" muy específicas, se podría usar UPPER\_CASE si la convención de equipo lo prefiere, pero PascalCase es lo estándar en C#.

* Ejemplos: MaxIntentos, DefaultPageSize.

**Archivos:** Deben seguir el nombre de la clase principal que contienen.

* Ejemplos: Usuario.cs, AuthController.cs.

**Espacios de Nombres (Namespaces):** PascalCase, reflejando la estructura de carpetas y la lógica del dominio.

* Ejemplos: ProyectoCF.Controllers, ProyectoCF.Models, ProyectoCF.Services.

# Validaciones y Anotaciones

Se utilizarán DataAnnotations para la validación de los campos en los Modelos de Dominio y ViewModels.

* [Required]: Para campos obligatorios.
  + Ejemplo: [Required(ErrorMessage = "El nombre es obligatorio")] public string Nombre { get; set; }
* [MaxLength(longitud)]: Para especificar la longitud máxima de una cadena.
  + Ejemplo: [MaxLength(100, ErrorMessage = "Máximo 100 caracteres")]
* [EmailAddress]: Para validar el formato de una dirección de correo electrónico.
* [Range(min, max)]: Para especificar un rango válido para valores numéricos.
  + Ejemplo: [Range(1, 3, ErrorMessage = "Seleccione una respuesta válida.")]
* [StringLength(max, min)]: Para validar la longitud mínima y máxima de una cadena.
* [DataType(DataType.Password)]: Para indicar que un campo contiene datos sensibles (ej. contraseñas) y no debe ser mostrado en logs, etc.
* **Validaciones Personalizadas:** Para lógicas de validación más complejas que no puedan cubrirse con Data Annotations estándar, se implementarán métodos de validación personalizados o interfaces como IValidatableObject.

# Comentarios Especiales

* **Comentarios de Bloque (// o /\* \*/):** Utilizar para explicar lógica compleja, decisiones de diseño no obvias o secciones de código específicas. Deben ser concisos y relevantes.
* **Comentarios XML (///):** Utilizar para documentar miembros públicos (clases, métodos, propiedades, interfaces) y parámetros. Esto genera documentación de IntelliSense y puede ser usado para generar documentación del API.

# Entidades de la Base de Datos

| **#** | **Nombre de la Tabla** | **Descripción de la Tabla** | **Nomenclatura** |
| --- | --- | --- | --- |
| 1 | Usuarios | Almacena la información de los usuarios | USR |
| 2 | Cursos | Información de los cursos disponibles | CRS |
| 3 | EstudiantesCursos | Relación entre estudiantes y cursos | EST\_CRS |
| 4 | Materiales | Almacena los materiales de los cursos | MAT |
| 5 | Evaluaciones | Contiene las evaluaciones de los cursos | EVA |
| 6 | Preguntas | Almacena las preguntas de las evaluaciones | PRG |
| 7 | Respuestas | Almacena las respuestas de las preguntas | RSP |
| 8 | Notas | Contiene las calificaciones de los estudiantes | NOT |
| 9 | Asistencias | Registra la asistencia de los estudiantes | AST |
| 10 | \_\_EFMigrationsHistory | Historial de migraciones de Entity Framework | — |

# Ejemplo de Modelo con Relaciones

public class Curso

{

[Key]

public int Id { get; set; }

[Required]

public string Nombre { get; set; }

[Required]

public string Descripcion { get; set; }

public int DocenteId { get; set; }

[ForeignKey("DocenteId")]

public Usuario Docente { get; set; }

public List<Material> Materiales { get; set; }

}