Desarrollo web del juego Quixxo en ASP.NET Core MVC

Programación Avanzada Web

Diciembre 2025

Integrantes

- Kimberly Michelle León Ramírez – Carné FI22026954 – `kleon80701@ufide.ac.cr` – Usuario Git: `theleonkim`
- Keylor Daniel Chacón Salas – Carné FH18005886 – `kchacon80925@ufide.ac.cr` – Usuario Git: `Keylor-CR`
- Juan Pablo Solís Benamburg – Carné FH21009671 – `jsolis30293@ufide.ac.cr` – Usuario Git: `juanpablosb04`
- Marvin Alberto Camacho Salas – Carné FH19006859 – `mcamacho40200@ufide.ac.cr` – Usuario Git: `marvcs`

1. Repositorio en Git

El código fuente completo del proyecto se encuentra en el siguiente repositorio:

- **Repositorio GitHub:** `https://github.com/theleonkim/ProyectoPAW`

Solo uno de los integrantes sube el enlace al Campus Virtual, pero todos los miembros del grupo figuran en este documento y en el repositorio.

2. Descripción General de la Aplicación

El proyecto consiste en el desarrollo web del juego **Quixxo** utilizando **ASP.NET Core MVC**.

La aplicación permite:

- Crear nuevas partidas de Quixxo en distintos modos de juego (por ejemplo, 2 y 4 jugadores).
- Registrar y persistir los movimientos realizados durante una partida.
- Visualizar el estado del tablero en cada movimiento.
- Detectar el ganador en función de las reglas del juego.
- Consultar estadísticas de jugadores y equipos.

3. Frameworks y Herramientas Utilizadas

- **Framework principal:** ASP.NET Core MVC.
- **Lenguaje de programación:** C#.
- **ORM:** Entity Framework Core.
- **Motor de base de datos:** SQLite.
- **Front-end:** HTML5, CSS3, Bootstrap, JavaScript.
- **Control de versiones:** Git y GitHub.
- **Entorno de desarrollo:** Visual Studio / Visual Studio Code.
- **Otros:** .NET SDK (versión utilizada en el proyecto), herramientas de línea de comandos de `dotnet` y `dotnet-ef`.

4. Tipo de Aplicación

La aplicación desarrollada es de tipo:

- **MPA (Multi Page Application)** basada en el patrón **MVC** de ASP.NET Core.

Cada sección principal del sistema (creación de partidas, tablero de juego, historial, estadísticas, etc.) se maneja mediante controladores y vistas independientes. El servidor genera las vistas en el lado servidor y las entrega al cliente en cada petición HTTP, manteniendo una navegación clásica por páginas.

5. Arquitectura Utilizada

La arquitectura utilizada en el proyecto es:

- **Arquitectura MVC (Model-View-Controller)** con capas lógicas separadas.

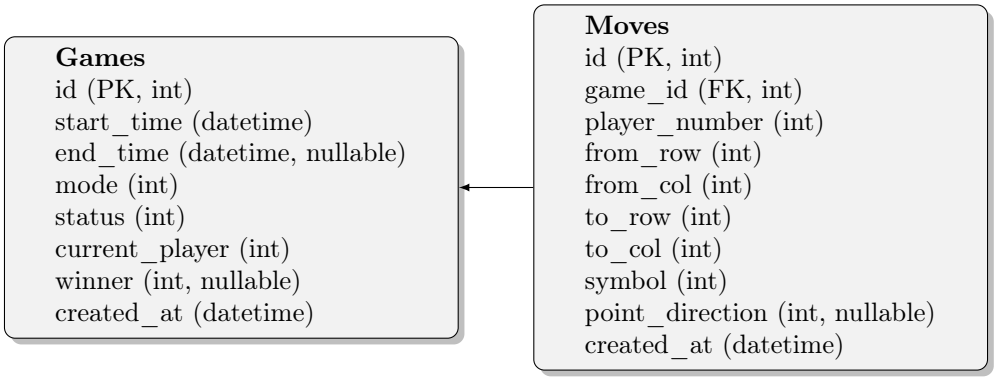
De forma resumida:

- **Capa de Modelos:** Contiene las clases de dominio (`Game`, `Move`, etc.) y las entidades que se mapean a la base de datos mediante Entity Framework Core.
- **Capa de Servicios:** Implementa la lógica de negocio y la lógica interna del juego (`GameService`, `GameLogicService`, `ExportService`, etc.), desacopladas de los controladores.
- **Capa de Controladores:** Recibe las peticiones HTTP, coordina la lógica de negocio a través de los servicios y retorna las vistas o respuestas JSON correspondientes.
- **Capa de Vistas:** Implementada con Razor Views, donde se renderiza la interfaz de usuario y se muestran los datos del modelo.

Esta separación facilita el mantenimiento, las pruebas y la extensión futura del proyecto.

6. Diagrama de Base de Datos

A continuación se muestra un diagrama simplificado de la base de datos del juego Quixxo. Para el proyecto se utiliza principalmente la relación entre partidas (`Games`) y movimientos (`Moves`) en una base de datos relacional SQL.



Este modelo refleja la relación uno-a-muchos: una partida (`Game`) tiene muchos movimientos (`Move`). La clave foránea `Moves.game_id` referencia a `Games.id` y los movimientos se eliminan en cascada cuando se elimina la partida.

7. Referencias y Recursos Web Utilizados

A continuación se listan algunas de las referencias consultadas para la implementación del proyecto, ejemplos de código y documentación técnica:

- Documentación oficial de ASP.NET Core MVC: `https://learn.microsoft.com/aspnet/core`
- Documentación oficial de Entity Framework Core: `https://learn.microsoft.com/ef/core`
- Documentación de Bootstrap: `https://getbootstrap.com/docs/`
- Foros y ejemplos de uso en *Stack Overflow*: `https://stackoverflow.com/`
- Ejemplos de uso de LINQ y patrones de repositorio en C# publicados en blogs técnicos.

8. Prompts de IA Utilizados

En esta sección se documentan los prompts utilizados con agentes de IA (por ejemplo, ChatGPT) tanto de entrada como de salida. Se incluyen los enlaces a las respuestas generadas cuando aplica.

Prompt (entrada)	**Respuesta (salida)**
"Generame la acción HTTP POST que inicia una nueva partida usando un servicio GameService y redirige a la vista de juego."	**Respuesta generada:** `https://chatgpt.com/s/t_6930e42bb4188191a3f4e19abbc2e6e3`

“Escríbeme una acción POST en ASP.NET Core MVC llamada MakeMove que reciba un movimiento del jugador vía JSON, que ejecute la jugada usando un servicio y devuelva un JSON con éxito o error.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e5f7d4548191916dd4a066fbfc8a
“Generá una acción llamada Reset"que reinicie la partida, ten en cuenta el ID y que luego redirija nuevamente a la pantalla del juego.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e63b4c608191818bd11b4a0a338b
“Creá una acción GET que reciba el ID de una de las partida, obtenga la información, prepare la lista de movimientos que hubo y retorne la vista correspondiente.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e6e35fc08191821986be958e308d
“Generame un endpoint GET que devuelva el estado del tablero en un movimiento dado.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e73fa0e48191994961b7d6858363
“Generá una acción que exporte una partida a XML y permita descargar el archivo con un nombre dinámico.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e77238048191b2829c22a172d0ba
“Creame un controlador que use GameService y el illoger. Debe tener una acción GET Index que obtenga las estadísticas de los jugadores y de los equipos desde GameService, que las coloque en ViewBag y retorne la vista.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e849dea48191b9da359b88cf9c3d
“Quiero que generes un DbContext completo para una aplicación ASP.NET Core que usa Entity Framework Core. El DbContext debe llamarse QuixoDbContext, debe recibir en el constructor DbContextOptions<QuixoDbContext>y debe incluir los DbSet necesarios para manejar partidas y movimientos del juego Quixo mencionado anteriormente. Las clases del modelo son Move y Game. Requisitos: Un Game tiene muchos Moves (relación 1 a muchos) con borrado en cascada. Las propiedades enum de Game (Mode y Status) deben almacenarse como int. Las propiedades enum de Move (Symbol y PointDirection, esta última nullable) deben almacenarse como int.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e8f33b9c8191b0ab2781fc6e98f9

“Necesito un servicio ExportService en ASP.NET Core que, dado un gameId, cargue la partida desde la base de datos (incluye los movimientos) y genere un XML de toda la información usando XElement / XDocument. Debe devolver el XML como string en un método ExportGameToXml. Entrega el código completo.”	**Respuesta generada:** https://chatgpt.com/s/t_6930e9a753ac8191b8943959e14251a4

“Necesito que generes un servicio llamado GameLogicService para un proyecto ASP.NET Core MVC del juego Quixo. Este servicio debe encargarse de toda la lógica interna del juego, sin acceso a base de datos, solo lógica pura. El servicio debe incluir: Inicialización del tablero Método que cree un tablero 5x5 de cubos neutrales. Serialización y deserialización Serializar el tablero (con símbolo y dirección del punto de cada cubo) a JSON. Deserializarlo nuevamente en un arreglo 2D de Cube. Reglas del juego Método IsPeripheral para validar si una posición está en la periferia. Método CanPickCube que determine si un jugador puede seleccionar un cubo según: Si está en la periferia. Si es primera ronda (solo neutros). Si es 2 jugadores o 4 jugadores. Reglas de propiedad del cubo según símbolo y orientación del punto. Movimiento de cubos Método GetValidPlacementPositions que dado un cubo seleccionado devuelva todas las posiciones válidas de inserción siguiendo las reglas reales de Quixo. Método MakeMove que: Copie el tablero. Extraiga un cubo de la periferia. Desplace los demás cubos según si el movimiento es horizontal o vertical. Inserte el cubo en la posición final con el símbolo del jugador. Detección de ganador Método CheckWinner que verifique líneas horizontales, verticales y diagonales para detectar si el juego terminó. Debe soportar modo de 2 jugadores y de 4 jugadores.”	**Respuesta generada:** https://chatgpt.com/s/t_6930eadaf5488191a6a49ef98957f7d7

“Quiero que generes un servicio llamado GameService para un proyecto ASP.NET Core MVC del juego Quixo, que dependa de un DbContext y de un servicio de lógica interna llamado GameLogicService. El servicio debe incluir: Crear nueva partida Un método CreateNewGame que: Inicialice el tablero usando GameLogicService. Serialice el tablero. Cree y guarde un registro Game en la base de datos. Configure modo, estado, jugador actual y banderas necesarias. Obtener partida por ID Un método GetGameById que retorne un Game con sus movimientos ordenados. Obtener GameViewModel Un método GetGameViewModel que: Deserialice el tablero. Calcule tiempo transcurrido. Determine ganador si aplica. Devuelva un GameViewModel completo. Hacer movimiento Un método MakeMove que: Valide periferia. Valide que el jugador pueda retirar el cubo con CanPickCube. Valide posiciones de destino con GetValidPlacementPositions. Determine símbolo según modo (2/4 jugadores). Ejecute MakeMove del GameLogicService. Serialice nuevo tablero. Registre el movimiento en base de datos. Cambie de turno o determine ganador. Devuelva un GameResponseDto. Chequear línea ganadora Un método privado CheckLine que revise líneas horizontales, verticales y diagonales. Juegos finalizados Un método GetFinishedGames. Estadísticas GetPlayerStatistics para modo 2 jugadores. GetTeamStatistics para 4 jugadores. Resetear partida Un método ResetGame que elimine movimientos y reconstruya estado inicial. Conversión a DTO Un método privado ConvertToDto.”	**Respuesta generada:** https://chatgpt.com/s/t_6930ebdc7dcc8191a5ada27b821f43df

9. Instructivo de Instalación, Compilación y Ejecución

A continuación se detalla un instructivo breve para poner en marcha la aplicación en un entorno local.

9.1. Requisitos Previos

- **.NET SDK** (versión utilizada en el proyecto, por ejemplo .NET 8.0).
- **SQLite** (no requiere servidor, solo acceso al archivo de base de datos configurado en la cadena de conexión).
- **Git** para clonar el repositorio.
- (Opcional) **Visual Studio** o **Visual Studio Code** con extensiones para desarrollo en .NET.

9.2. Instalación

```
# 1. Clonar el repositorio
git clone https://github.com/theleonkim/ProyectoPAW.git

# 2. Ingresar a la carpeta del proyecto (ajustar si hay solución / proyectos internos)
cd ProyectoPAW/QuixoGame

# 3. Restaurar paquetes NuGet
dotnet restore
```

9.3. Compilación / Creación

```
# Compilar la solución
dotnet build
```

En Visual Studio, también se puede compilar utilizando la opción *Build Solution* desde el menú.

9.4. Ejecución

```
# Ejecutar la aplicación desde la línea de comandos
dotnet run
```

O bien, desde Visual Studio, seleccionar el proyecto web correspondiente (por ejemplo, `ProyectoPAW`) y presionar **F5** o el botón de *Run*.

Por defecto, la aplicación quedará disponible en una URL similar a:

```
https://localhost:5001
http://localhost:5000
```

(La URL exacta puede variar según la configuración del `launchSettings.json`.)