Documentación del Proyecto QuixoWeb

BONILLA VASQUEZ BRANDON RAFAEL
NELSON RODRÍGUEZ LÓPEZ

9 de diciembre de 2025

Integrantes y datos de Git

Nombre	Carné	Usuario de Git	Correo de Git
BONILLA VASQUEZ BRANDON RAFAEL	FI17008598	`brandonbonillavasquez`	`bbonilla40084@ufide.ac.cr`
NELSON RODRÍGUEZ LÓPEZ	FI20016869	`nelson2587`	`nrodriguez70450@ufide.ac.cr`

Stack, frameworks y herramientas

- **Framework web**: ASP.NET Core MVC (Multi-Page Application, MPA).
- **ORM**: Entity Framework Core (EF Core).
- **Motor de base de datos**: Microsoft SQL Server (puede usarse LocalDB durante desarrollo).
- **Lenguaje**: C# (controladores, servicios, modelos).
- **Cliente**: Vistas Razor + JavaScript (AJAX).
- **Herramientas**: .NET SDK, Visual Studio/VS Code, Git, LaTeX (PGF/TikZ) para este documento.

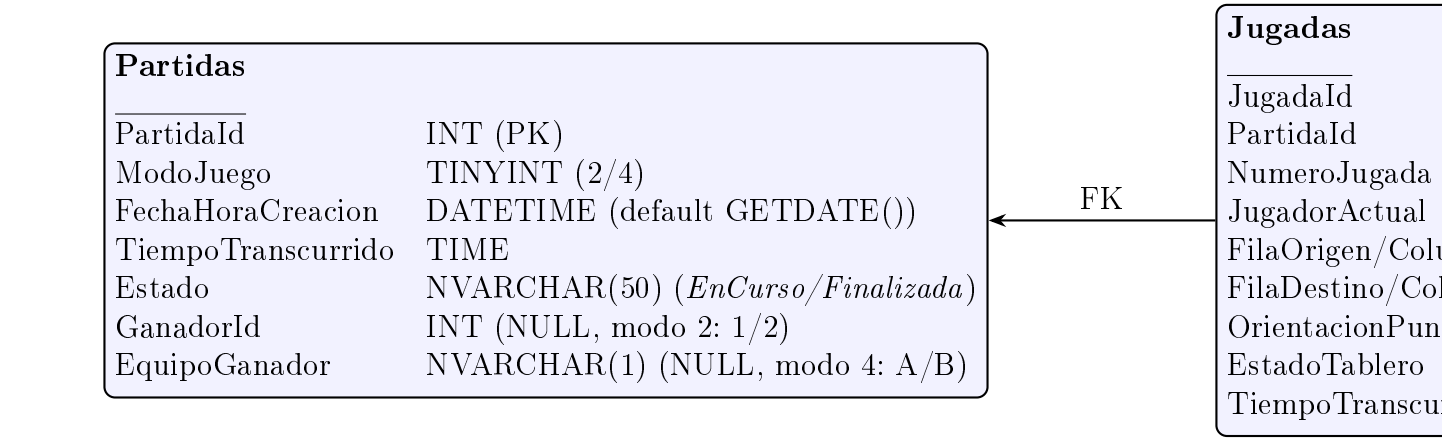
Tipo de aplicación

- **MPA**: Se renderizan múltiples vistas Razor en servidor; además, el proyecto expone acciones que retornan JSON para actualizar la UI vía AJAX.

Arquitectura utilizada

- **MVC**: Controladores (Home, Game, Stats) se utilizan para realizar peticiones y delegan en un servicio de dominio (`IGameService/GameService`). EF Core (`QuixoDbContext`) persiste entidades (`Partida`, `Jugada`, `Estadistica`). ViewModels (`GameViewModel`, `HistoryViewModel`, `StatsViewModel`) preparan datos para vistas/JSON.

Definición de la base de datos (PGF/TikZ)



Estadisticas

EstadisticaId	INT (PK)
ModoJuego	TINYINT (2/4)
JugadorEquipo	NVARCHAR(20) (*Jugador1/Jugador2/EquipoA/EquipoB*)
PartidasGanadas	INT
PartidasJugadas	INT

Referencias y recursos consultados

- ASP.NET Core MVC (Controllers y Views): learn.microsoft.com/aspnet/core/mvc
- Entity Framework Core (Modelos, relaciones e índices): learn.microsoft.com/ef/core
- System.Text.Json (serialización a JSON): learn.microsoft.com/dotnet/standard/serialization/system-text-json-overview
- ASP.NET Core Session: learn.microsoft.com/aspnet/core/fundamentals/app-state
- SQL Server GETDATE(): learn.microsoft.com/sql/t-sql/functions/getdate-transact-sql
- LaTeX PGF/TikZ: ctan.org/pkg/pgf

Prompts y agentes de IA usados

- Interacción con M365 Copilot para: explicación de código, diseño de documentación, y generación de diagramas y secciones del README.
- Incluya aquí los *prompts* exactos de entrada y los fragmentos de salida que se utilizaron durante el desarrollo y documentación.
- Interacción con ChatGPT (GPT-5.1 Thinking) para la creación del código, resolución de errores y documentación técnica del proyecto.

- Repositorio y desarrollo general del proyecto Quixo Web.
- Sesiones de apoyo para la documentación y tareas complementarias.
- Generación del README en LaTeX y explicación detallada del código.
- Discusión de funcionalidades específicas y pruebas.
- Ajustes finales y preparación para la exposición.

https://github.com/BrandonBonillaVasquez/FI17008598.git

Instructivo

Prerrequisitos

- .NET SDK 7/8 instalado (dotnet.microsoft.com/download).
- SQL Server (LocalDB durante desarrollo) y una cadena de conexión `QuixoConnection` en `appsettings.json`.

Instalación

1. Clonar el repositorio: `git clone https://github.com/BrandonBonillaVasquez/FI17008598.git`
2. Configurar la cadena de conexión en `appsettings.json`, por ejemplo:

```
{
  "ConnectionStrings": {
    "QuixoConnection": "Server=(localdb)\\MSSQLLocalDB;Database=QuixoDb;Trusted_Connec
  }
}
```

3. Restaurar paquetes: `dotnet restore`
4. Aplicar migraciones y crear/actualizar la base: `dotnet ef database update`

Compilación / Creación

```
dotnet build
```

Ejecución

```
dotnet run
```

La aplicación expondrá rutas como `/Home/Index`, `/Game/SelectMode`, `/Stats/Index`.

Compilar este README en PDF (LaTeX)

1. Compilar localmente: `pdflatex README.tex` (ejecutar 2 veces si es necesario).
2. Alternativa: Importar `README.tex` en Overleaf y exportar `README.pdf`.

Notas finales

- El repositorio **debe incluir todo el código fuente** y **excluir artefactos compilados** (carpetas `bin/` y `obj/`).
- Incluya `README.tex` y el `README.pdf` generado con LaTeX en la raíz del proyecto.