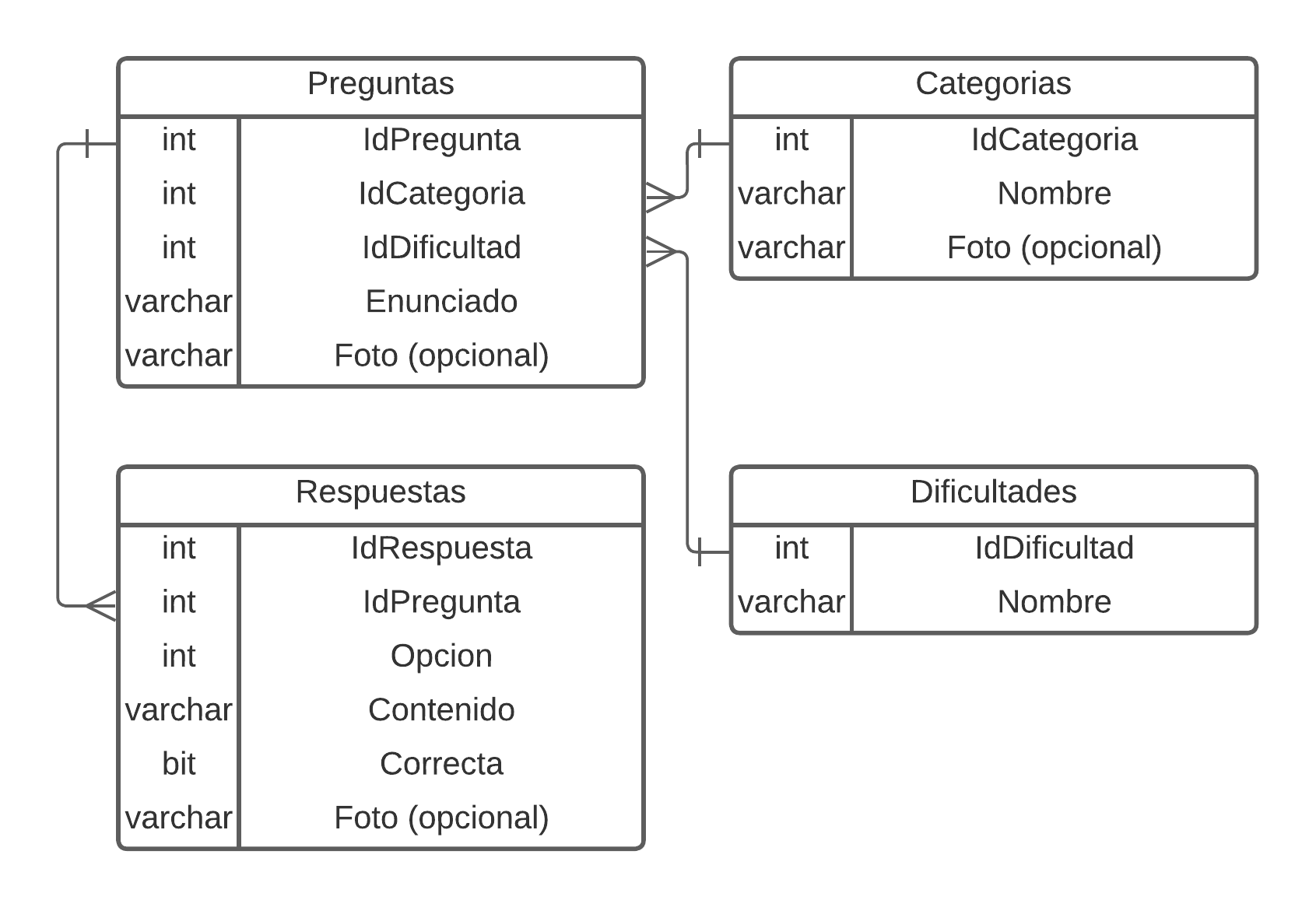
## TP - PreguntadOrt

*Desarrollar un sitio web utilizando MVC Net Core.*

*Obtener y registrar información en una base de datos SQL Server utilizando Dapper ORM.*

Crear la base de datos **PreguntadOrt** con las siguientes tablas (respetar nombres y tipos):



Crear un proyecto .NET Core MVC **TP\_PreguntadORT** que contenga la siguiente estructura:

**Models**

1. Crear y definir las clases **Categoria**, **Dificultad**, **Pregunta** y **Respuesta**, definiendo correctamente sus propiedades de lectura/escritura.
2. Crear y definir la clase estática ***BD*** con los siguientes métodos estáticos y públicos
   1. **ObtenerCategorias()**: Devuelve una lista con todas las categorías
   2. **ObtenerDificultades()**: Devuelve una lista con todas las dificultades
   3. **ObtenerPreguntas(int dificultad, int categoria)**: Recibe un id de dificultad y un id de categoría. Devuelve una lista con las preguntas que se van a utilizar en el juego.

Aclaración:

Si dificultad = -1, trae las preguntas de todas las dificultades.

Si categoria = -1, trae las preguntas de todas las categorías.

* 1. **ObtenerRespuestas(int idPregunta)**: Recibe un id de Pregunta y devuelve una lista con todas las respuestas para dichas preguntas.

(Crear además todos los atributos y métodos privados que se requieran vistos en clase, para poder conectar el proyecto con la base de datos)

1. Crear y definir la clase **estática *Juego*** con los siguientes elementos **estáticos**:

Atributos privados recomendados:

1. string username
2. int puntajeActual
3. int contadorPreguntaActual
4. int cantidadPreguntasCorrectas
5. List<Pregunta> ListaPreguntas
6. List<Respuesta> ListaRespuestas

Métodos:

1. **InicializarJuego():** Debe inicializar todos los atributos privados de la clase que dan inicio el juego. Username a vacío, puntaje actual y cantidad preguntas correctas a 0.
2. **ObtenerCategorias():** Retorna la lista de categorías.
3. **ObtenerDificultades():** Retorna la lista de dificultades.
4. **CargarPartida(string username, int dificultad, int categoria):** Recibe la dificultad y categoría elegidas, invoca a los métodos ObtenerPreguntas y ObtenerRespuestas (en ese orden) y guarda los resultados en los atributos \_preguntas y \_respuestas.
5. **ObtenerProximaPregunta():** Retorna, de ser posible, una pregunta al azar de la lista de preguntas.
6. **ObtenerProximasRespuestas(int idPregunta):** Retorna una lista con todas las respuestas relacionadas a la pregunta enviada por parámetro
7. **VerificarRespuesta(int idPregunta, int idRespuesta)**: Recibe un id de pregunta y un id de respuesta, y retorna un booleano indicando si la respuesta fue correcta o incorrecta. Previo a devolver el booleano realiza dos acciones:
8. Si la respuesta del usuario fue correcta, suma una cantidad específica de puntos a \_puntajeActual (la definen ustedes) y suma 1 respuesta correcta en \_cantidadPreguntasCorrectas.
9. Elimina la pregunta enviada por parámetro de la lista de preguntas

**Controller**

1. **public IActionResult Index:** Debe retornar la view Index
2. **public IActionResult ConfigurarJuego()**: Debe llamar al método InicializarJuego() de la clase Juego y retornar la View ConfigurarJuego. Por ViewBag deben viajar las categorías y dificultades!
3. **public IActionResult Comenzar(string username, int dificultad, int categoria)**: Recibe el username, dificultad y categoría elegidas por el usuario, invoca al método CargarPartida de la clase Juego y, siempre y cuando lleguen preguntas de la base de datos, redirige el sitio al ActionResult Jugar. En caso de elegir una configuración sin preguntas en base de datos, debe redirigir nuevamente al ActionResult ConfigurarJuego.
4. **public IActionResult** **Jugar():** Carga en ViewBag todo lo necesario para mostrar la pregunta actual con sus respectivas respuestas (que proviene del método ObtenerProximaPregunta. Si ya no hay más preguntas disponibles, retorna la view Fin. Si el método retorna una pregunta, invoca a ObtenerProximasRespuestas de la clase Juego guardando estos datos en ViewBag y retorna la view Juego.
5. **[HttpPost] public IActionResult** **VerificarRespuesta(int idPregunta, int idRespuesta):**  Recibe el id de la respuesta elegida, invoca al método VerificarRespuesta de la clase Juego y retorna la view Respuesta, enviando por ViewBag si fue correcta o no. (Como opcional, podés enviar también cuál era la respuesta correcta).

**Views**

1. **\_Layout.cshtml**  
   En la navbar debe estar el link a Jugar (invoca a ConfigurarJuego) y el logo del juego (lo crean ustedes)
2. **Index.cshtml**  
   Debe mostrar el logo del juego y un enlace en formato de botón que invoca al método ConfigurarJuego del controller.
3. **ConfigurarJuego.cshtml**

Permite configurar mediante un formulario (GET) usando *select* o *input type radio*:

1. Dificultad (Eligiendo una en particular, o todas → value = -1 )
2. Categoría (Eligiendo una en particular o todas → value = -1)
3. Username

Todos los campos deben ser required!

Debe existir un botón Jugar que invoque al método Comenzar del controller.

1. **Juego.cshtml**

Muestra:

1. Nombre de usuario y puntaje actual
2. Número de pregunta actual y enunciado (Si hay foto la muestra)
3. Un formulario (POST) con 4 botones para elegir la respuesta correcta (Si las respuestas tienen fotos, las muestra también como parte de la opción).
4. Un botón para responder y enviar el formulario (llama al método VerificarRespuesta del controller).

El diseño de esta pantalla se la dejamos a criterio de cada grupo!

1. **Respuesta.cshtml**

Muestra:

1. un cartel grande (ponerle onda!!) indicando si la respuesta fue correcta o incorrecta. En caso de haber sido incorrecta, mostrar cuál era la respuesta correcta.
2. Un enlace para jugar la siguiente pregunta (llama al método Jugar del controller).
3. **Fin.cshtml**

Muestra:

1. Nombre de usuario
2. Puntaje final
3. Un enlace para volver a jugar (Reinicia el juego invocando al método ConfigurarJuego del controller)

**Opcionales!**

1. **Tabla de Puntajes**. Al final del juego se graba en una nueva tabla de la base de datos un registro con: Fecha-Hora, Username y Puntaje. Además, agregar una nueva view llamada HighScores, donde en una tabla se visualicen los mejores puntajes (por favor, al pedir los datos a la base mediante un método en la clase BD, traer los registros ordenados de mayor a menor puntaje!!)
2. **Creador de Preguntas**. Crear el **BackOfficeController**, que permitirá mediante algunos ActionResult crear nuevas preguntas para el juego. Este controller deberá invocar a un método CrearPregunta de la clase BD. Por último, definir además la/s view/s necesarias para poder realizar esta acción.
3. **Super desafío:** implementar un timer en la view Juego (seteado en la cantidad de tiempo que deseen), y que en base a cuánto antes responde el usuario, sumar más o menos puntos a su puntaje general (siempre y cuando la respuesta sea correcta). Aclaración: Este opcional es 100% autodidacta e independiente.

**Formato de entrega**

Subir al servicio del Campus Virtual el link a GitHub. Guardar en la carpeta wwwroot el archivo .sql de la base de datos. El archivo a subir debe tener como nombre el apellido de los/las integrantes y el número de Trabajo Práctico, respetando la siguiente nomenclatura que muestra el ejemplo: TP\_PreguntadORT\_Medina\_Kristal.zip