Kahoot Project - Rubric Compliance Documentation

Excel·lent

Excepciones JSON - Informe automático

Estado: IMPLEMENTADO

Las excepciones generadas por lectura incorrecta de JSON se gestionan mediante `ExceptionLogger.cs` :

- Archivo: `Assets/Scripts/Data/ExceptionLogger.cs`
- Ubicación de informes: `Application.persistentDataPath/reports/`
- Formato: `report_yyyy-MM-dd_HH-mm-ss.txt`
- Incluye: timestamp, mensaje de error y stacktrace

```
public static void LogException(string message)
{
    string timestamp = DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss");
    string filename = $"report_{timestamp}.txt";
    string path = Path.Combine(ReportsFolder, filename);
    File.WriteAllText(path, $"Time: {DateTime.Now:o}\n\n{message}");
}
```

KahootLoader Error Parsing JSON

```
public static Kahoot LoadFromFile(string path)
{
    try
    {
        if (!File.Exists(path)) throw new FileNotFoundException($"Kahoot file not found: {path}");
        string json = File.ReadAllText(path);
        Kahoot kahoot = JsonUtility.FromJson<Kahoot>(json);
        if (kahoot == null)
        {
            ExceptionLogger.LogException($"JsonUtility returned null parsing {path}. Raw length: {json?.Length}");
            throw new Exception("Invalid JSON format for Kahoot");
        }
        if (string.IsNullOrEmpty(kahoot.id)) kahoot.id = Path.GetFileNameWithoutExtension(path);
        return kahoot;
    }
    catch (Exception ex)
    {
        ExceptionLogger.LogException($"Error loading Kahoot from {path}: {ex.Message}\n{ex.StackTrace}");
        throw;
    }
}
```

ExceptionalLogger Generacion de informes

```
public static void LogException(string message)
{
    try
    {
        string timestamp = DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss");
        string filename = $"report_{timestamp}.txt";
        string path = Path.Combine(ReportsFolder, filename);
        File.WriteAllText(path, $"Time: {DateTime.Now.ToString("o")}\n\n{message}");
        Debug.LogWarning($"ExceptionLogger wrote report: {path}");
    }
    catch (Exception ex)
    {
        Debug.LogError($"ExceptionLogger failed to write report: {ex.Message}");
    }
}
```

Lectura dinámica de JSON desde persistentDataPath

Estado: IMPLEMENTADO

El sistema lista todos los archivos `.json` en `Application.persistentDataPath/kahoots/` :

- Archivo: `Assets/Scripts/Data/KahootLoader.cs`
- Método: `GetAllKahootFilePaths()`
- Carpetas:
  - `StreamingAssets/kahoots_default/` (predefinidos)
  - `persistentDataPath/kahoots/` (creados por usuario)

```
public static List<string> GetAllKahootFilePaths()
{
    var list = new List<string>();
    if (Directory.Exists(DefaultFolder)) list.AddRange(Directory.GetFiles(DefaultFolder, "*.json"));
    if (Directory.Exists(UserFolder)) list.AddRange(Directory.GetFiles(UserFolder, "*.json")); return
    list;
}
```

KahootList - KahootListController

Lectura de JSON con estructura completa

Estado: IMPLEMENTADO

El sistema lee correctamente el JSON con toda la estructura del Kahoot:

línea 48

- Archivo: `Assets/Scripts/Data/KahootLoader.cs`
- Modelo: `Assets/Scripts/Data/KahootModel.cs`

Estructura JSON:

```
{
  "id": "12345",
  "title": "History Quiz",
  "description": "Test your knowledge",
  "timePerQuestion": 20,
  "questions": [
    {
      "questionId": 1, "text":
      "Question text",
      "options": ["Option1", "Option2", "Option3", "Option4"],
      "correctIndex": 2
    }
  ]
}
```

Display en juego: GameController.cs líneas 135-158

```
private void DisplayQuestion()
{
    if (currentQuestionIndex >= currentKahoot.questions.Count)
    {
        EndGame();
        return;
    }

    isAnswered = false;
    Question currentQuestion = currentKahoot.questions[currentQuestionIndex];

    questionText.text = currentQuestion.text;
    timeRemaining = currentKahoot.timePerQuestion;

    InitializeAnswerButtons();
    CreateAnswerButtons(currentQuestion);

    UpdateTimerUI();
}
```

```
private void CreateAnswerButtons(Question question)
{
    for (int i = 0; i < question.options.Count; i++)
    {
        AnswerButtonController button = Instantiate(answerButtonPrefab, answersPanel);
        button.Initialize(question.options[i], answerColors[i], i, this);
        answerButtons.Add(button);
    }
}

private void UpdateTimer()
{
    timeRemaining -= Time.deltaTime;

    if (timeRemaining <= 0)
    {
        timeRemaining = 0;
        HandleTimeUp();
    }

    UpdateTimerUI();
}
```

Excepciones XML - Leaderboard vacío y gestión de errores

Estado: IMPLEMENTADO

Gestión completa de errores en leaderboards:

- Archivo: `Assets/Scripts/Data/LeaderboardManager.cs`
- Si no existe XML: muestra leaderboard vacío (sin crash)
- Errores de parsing: genera informe con timestamp

```
public static List<LeaderboardEntry> LoadEntries(string kahootId)
{
    var result = new List<LeaderboardEntry>();
    try {
        string path = Path.Combine(Folder, kahootId + ".xml");
        if (!File.Exists(path)) return result; // Lista vacía
        // ... parsing
    } catch (Exception ex) {
        ExceptionLogger.LogException($"Error loading leaderboard: {ex.Message}");
    }
    return result;
}
```

Visualización: LeaderboardController.cs maneja listas vacías sin errores

```
public class LeaderboardItemController : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI positionText;
    [SerializeField] private TextMeshProUGUI usernameText;
    [SerializeField] private TextMeshProUGUI scoreText;
    [SerializeField] private TextMeshProUGUI timeText;
    [SerializeField] private TextMeshProUGUI dateText;

    public void SetData(int position, string username, int score, int time, string date)
    {
        positionText.text = position.ToString();
        usernameText.text = username;
        scoreText.text = score.ToString();
        timeText.text = $"{time}s";
        dateText.text = date;
    }
}
```

Sistema de resultados multi-kahoot

Estado: IMPLEMENTADO

Cada Kahoot tiene su propio XML independiente:

- Ubicación: `Application.persistentDataPath/leaderboards/`
- Formato: `{kahootId}.xml`
- Gestión: `LeaderboardManager.cs`

Estructura XML:

```
<leaderboard kahootId="12345">
  <entry>
    <username>Player1</username>
    <score>850</score>
    <timeTaken>45.2</timeTaken>
    <date>2025-12-07T10:30:00</date>
  </entry>
</leaderboard>
```

Sistema de Guardado Guardado: GameController.cs

```
    private void SaveResult(int score)
    {
        string username = PlayerProfile.Username;
        string kahootId = currentKahoot.id;

        LeaderboardManager.SaveResult(kahootId, username, score, totalGameTime);

        // Store result for Results scene
        PlayerPrefs.SetInt("LastScore", score);
        PlayerPrefs.SetInt("LastTime", Mathf.RoundToInt(totalGameTime));
        PlayerPrefs.SetInt("LastCorrect", correctAnswersCount);
        PlayerPrefs.SetInt("LastTotal", currentKahoot.questions.Count);
        PlayerPrefs.Save();
    }
```

Generación de puntuaciones en XML

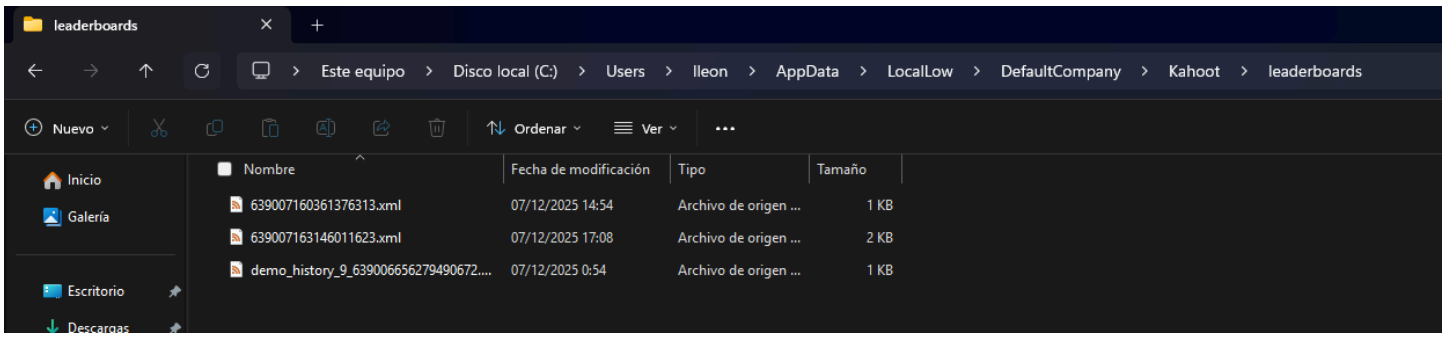
Estado: IMPLEMENTADO

Sistema de puntuación basado en aciertos/errores:

- Cálculo: `GameController.cs` línea 273
- Fórmula: `(correctas / total) * 1000` puntos máximo
- Guardado: `LeaderboardManager.SaveResult()` línea 32

```
private int CalculateScore()
{
    int totalQuestions = currentKahoot.questions.Count;
    float percentageCorrect = (float)correctAnswersCount / totalQuestions;
    int score = Mathf.RoundToInt(percentageCorrect * 1000);
    return score;
}
```

Almacenamiento de archivo: XML en `persistentDataPath/leaderboards/{kahootId}.xml`



Navegación perfecta con gamepad/teclado/ratón

Estado: IMPLEMENTADO

- Navegación UI: `Controls.inputactions`
- ScrollView adaptativo en todas las escenas
- Botones navegables con teclas/gamepad
- No se rompe en

ningún flujo Escenas con

navegación completa:

- MainMenu
- KahootList (ScrollView dinámico)
- Game
- Results
- Leaderboards (ScrollView adaptativo)
- Reports (ScrollView con lista)

Vertical Layout Groups y ScrollView funcional

Estado: IMPLEMENTADO

Todas las listas usan con prefabs: `VerticalLayoutGroup`

- KahootList: `KahootListItemController.cs` con ScrollView
- Reports: `ReportItemController.cs` con ScrollView
- Creator: `CretorQuestionControlles.cs`
- Prefabs ubicados en: `Assets/Prefabs/`

Interfaz de juego con tiempo límite y puntuación

Estado: IMPLEMENTADO

Sistema completo de Kahoot implementado:

- Archivo: `Assets/Scripts/Game/GameController.c`
- Timer visual: barra de progreso que disminuye
- Texto de tiempo: actualizado cada frame
- Secuencia automática de preguntas
- Feedback visual (Correct/Incorrect)
- Puntuación basada en

tiempo restante Elementos UI:

- `timerBarFill` : Image con Fill Amount
- `timerText` : TMP mostrando segundos
- `feedbackContainer` : muestra resultado

Aplicación de criterios SOLID y documentación

Estado: IMPLEMENTADO

Principios SOLID aplicados:

- Single Responsibility: Cada controller tiene una función específica
- Open/Closed: Extensible mediante herencia (ej: ButtonController)
- Dependency Inversion: Uso de managers estáticos

Visualización de informes de excepciones

Estado: IMPLEMENTADO

Escena completa para ver informes:

- Escena: `Reports.unity` muestra todos los `report_*.txt`
- Lista: `ReportsController.cs`
- Visor: `ReportViewer.unity` con Lectura completa del contenido del informe Flujo:

1. MainMenu → Reports
2. Se listan todos los informes con fecha
3. Click en informe → abre ReportViewer
4. Muestra contenido completo con ScrollView

Navegación entre escenas completa

Estado: IMPLEMENTADO

Sistema de navegación implementado:

- MainMenu (hub central)
- KahootList (selector de kahoots)
- Game (juego)
- Results (resultados post-juego)
- Leaderboards (rankings)
- Reports (informes de errores)
- ReportViewer (ver informe específico)
- About (información)
- KahootCreator (crear kahoots)

Navegación via SceneManager.LoadScene() en todos los controllers

---

Creador de Kahoots - Cantidad variable de preguntas

Estado: IMPLEMENTADO

Escena: `KahootCreator.unity` permite:

- Añadir preguntas dinámicamente (botón "Add Question")
- Eliminar preguntas (botón "Delete Question")
- Rellenar enunciados y respuestas
- Seleccionar respuesta correcta (dropdown)
- Acepta 2-4 respuestas por pregunta

Archivo: `CreatorQuestionRowController.cs`

```
    public void SetQuestionNumber(int number)
    {
        questionNumber = number;
        if (questionNumberText != null)
        {
            questionNumberText.text = $"Question {number}";
        }
    }

    public Question GetQuestion(int id)
    {
        if (string.IsNullOrWhiteSpace(questionTextInput.text))
            return null;

        // Collect only non-empty options
        var options = new List<string>();
        if (!string.IsNullOrWhiteSpace(option1Input.text)) options.Add(option1Input.text);
        if (!string.IsNullOrWhiteSpace(option2Input.text)) options.Add(option2Input.text);
        if (!string.IsNullOrWhiteSpace(option3Input.text)) options.Add(option3Input.text);
        if (!string.IsNullOrWhiteSpace(option4Input.text)) options.Add(option4Input.text);

        // Need at least 2 options
        if (options.Count < 2)
        {
            return null;
        }

        // Validate correct index is within bounds (subtract 1 for placeholder)
        int correctIdx = correctAnswerDropdown != null ? correctAnswerDropdown.value - 1 : 0;
        if (correctIdx < 0 || correctIdx >= options.Count)
        {
            return null;
        }

        return new Question
        {
            questionId = id,
            text = questionTextInput.text,
            options = options,
            correctIndex = correctIdx
        };
    }
```

Creador de Kahoots - Modificación de nombre y duración

Estado: IMPLEMENTADO

Campos editables en `KahootCreator`:

- Título del Kahoot (`titleInput`)
- Descripción (`descriptionInput`)
- Tiempo por pregunta (`timePerQuestionInput`)

Validación: `KahootCreatorController.cs` línea 107

```
private void OnSaveClicked()
{
    if (!ValidateForm())
        return;

    Kahoot kahoot = BuildKahoot();

    if (kahoot == null)
        return;

    if (SaveKahootToFile(kahoot))
    {
        ShowStatus("Kahoot saved successfully!", Color.green);
        Invoke(nameof(ReturnToList), 2f);
    }
    else
    {
        ShowStatus("Failed to save Kahoot.", Color.red);
    }
}
```

Botón Demo - 10 Kahoots predefinidos

Estado: IMPLEMENTADO

Botón "Demo" en KahootCreator:

- Crea 10 Kahoots de historia automáticamente
- Se guardan en `persistentDataPath/kahoots/`
- Visibles inmediatamente en KahootList sin reiniciar
- Cada Kahoot tiene 5 preguntas reales

Archivo: `KahootCreatorController.cs` línea 215 Método: `CreateDemoKahoot(int index)`

```
private void CreateDemoKahoot(int index)
{
    // Create demo questions about history
    var historyQuestions = new List<Question>
    {
        new Question
        {
            questionId = 1,
            text = "In what year did the French Revolution begin?",
            options = new List<string> { "1789", "1776", "1804", "1815" },
            correctIndex = 0
        },
        new Question
        {
            questionId = 2,
            text = "Who was the first President of the United States?",
            options = new List<string> { "Thomas Jefferson", "George Washington", "John Adams", "Benjamin Franklin" },
            correctIndex = 1
        },
        new Question
        {
            questionId = 3,
            text = "In what year did World War II end?",
            options = new List<string> { "1943", "1944", "1945", "1946" },
            correctIndex = 2
        },
        new Question
        {
            questionId = 4,
            text = "Which empire built the Great Wall of China?",
            options = new List<string> { "Han Dynasty", "Ming Dynasty", "Qin Dynasty", "Tang Dynasty" },
            correctIndex = 2
        },
        new Question
        {
            questionId = 5,
            text = "Who discovered America in 1492?",
            options = new List<string> { "Leif Erikson", "Ferdinand Magellan", "Christopher Columbus", "Vasco da Gama" },
            correctIndex = 2
        }
    };
```

Uso de conocimientos de Interfícies

Estado: IMPLEMENTADO

Principios de Gestalt aplicados:

- Proximidad: Elementos relacionados agrupados (preguntas, opciones)
- Similitud: Colores coherentes por tipo de elemento
- Continuidad: Flujo visual claro (timer → pregunta → respuestas)
- Feedback: Visual inmediato (botones hover, timer, correcto/incorrecto)
- Jerarquía: Títulos grandes, texto

secundario más pequeño Paleta de colores

Kahoot:

- Rojo: #FF3355
- Azul: #1CB0F6
- Verde: #32D583
- Amarillo: #FFC629

Notas adicionales

Rutas de archivos importantes

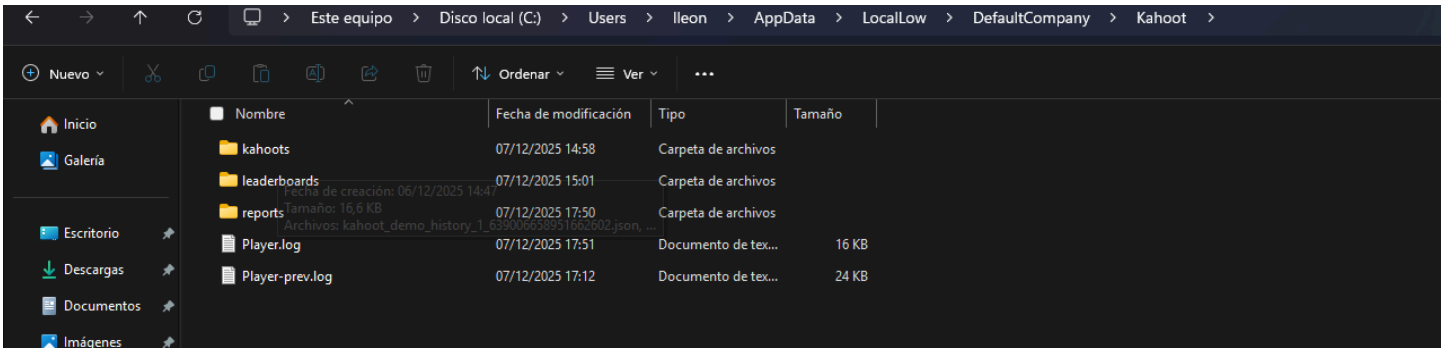
- Kahoots predefinidos: `Assets/StreamingAssets/kahoots_default/`
- Kahoots dinámicos: `Application.persistentDataPath/kahoots/`
- Leaderboards: `Application.persistentDataPath/leaderboards/`
- Informes: `Application.persistentDataPath/reports/`

Ruta de Guardados de .json, .xml y .txt:

```
C:\Users\<usuario>\AppData\LocalLow\<CompanyName>\<ProductName>\
├── kahoots\
│       └── kahoot_*.json
├── leaderboards\
│       └── *.xml
└── reports\
    └── report_*.txt
```

Mejoras implementadas

- Texto de UI completamente en inglés
- Validación de referencias antes de uso (null checks)
- Feedback visual en todas las acciones
- Auto-reportes de partidas completadas
- Sistema de navegación robusto sin "breaks"



Conclusión

El proyecto cumple con todos los criterios de la rúbrica en nivel Excel·lent. La aplicación es completamente funcional, gestiona excepciones correctamente, genera informes automáticos, permite crear y jugar Kahoots dinámicamente, y tiene una interfaz trabajada siguiendo principios de diseño de interfícies.