Informe de Estado del Proyecto "Adivina la Canción" para Retomar

Fecha del Informe: 28 de Julio de 2025

Objetivo de este Informe: Recordar rápidamente el estado del proyecto, las decisiones clave, los problemas resueltos y los pendientes, así como los comandos esenciales de Git/despliegue para retomar el desarrollo de forma eficiente.

---

1. Visión General del Proyecto y Objetivos

- **Nombre:** Adivina la Canción.
- **Concepto:** Juego web interactivo de trivial musical. Adivinar canciones/BSO tras escuchar un fragmento.
- **Estado General Actual:** Funcional a nivel de juego principal y gestión de usuarios/puntuaciones. Hay un problema pendiente con la visibilidad de todas las décadas en el menú.
- **Próximos Pasos Pendientes (Prioridad):**
    - **Resolver el problema del menú de décadas:** Hacer que aparezcan los botones de todas las décadas (80s, 90s, 00s, 10s, Actual) además de "Todas las Décadas". Esto implica asegurarse de que los archivos de datos de canciones estén correctos y completos para cada década/categoría.
    - Generar los archivos .js de muestra para las categorías de los 80s (excepto español) con 10 "audios" inventados, para pruebas.
- **Objetivos a Largo Plazo (después de lo anterior):**
    - Implementar **confirmación de email** (Nodemailer + servicio SMTP externo).
    - Implementar **recuperación de contraseña** (Nodemailer + servicio SMTP, tokens seguros).
    - Pantalla de Estadísticas Avanzada (Tablas de clasificación, duelos, etc.).
    - Multijugador Online (WebSockets).
    - Nuevas Modalidades de Juego, mejoras UX, SFX avanzados, etc.

---

2. Arquitectura y Tecnologías Clave

- **Frontend:** HTML5, CSS3 (estilo retro/neón), **JavaScript (Vanilla JS)**.
    - **Importante:** ¡El frontend ahora tiene una estructura modular con carga dinámica de datos!
- **Backend: Node.js + Express.js**.
    - **Base de Datos: MongoDB Atlas** (desplegada en Railway).
    - **Autenticación:** bcryptjs (hashing de contraseñas).

    - **Comunicación:** cors (manejo de Cross-Origin Resource Sharing).
    - **Variables de Entorno:** dotenv.
- **Despliegue:**
    - **Frontend:** Netlify (adivinala.netlify.app).
        - **Publish directory en Netlify:** frontend/ (¡CRÍTICO! Asegurarse siempre de que apunte a frontend/ en la configuración de Netlify).
    - **Backend:** Railway (accomplished-balance-production.up.railway.app).
        - **Root Directory en Railway:** backend/.
        - **Start Command:** npm start.

---

3. Estructura de Archivos (Frontend - ¡Fundamental!)

La estructura es modular. Es **CRÍTICO** que cada archivo esté en su lugar y con el contenido correcto.

Mi juego/ (Raíz del repositorio de Git. TODOS los comandos 'git' se ejecutan desde aquí)

├— frontend/

| ├— index.html <-- SOLO la estructura HTML + enlaces a CSS/JS externos

| ├— css/

| | └— style.css <-- TODO el CSS puro. NO etiquetas <style> ni comentarios HTML.

| ├— js/

| | ├— main.js <-- Lógica principal del juego, UI, eventos.

| | └— songs-loader.js <-- Carga dinámica de los datos de canciones.

| ├— data/ <-- ¡NUEVA CARPETA para los datos de canciones!

| | └— songs/

| | ├— 80s/ <-- Carpetas por DÉCADA.

| | | ├— espanol.js <-- Archivos JS con array de canciones para esa CATEGORÍA y DÉCADA.

| | | ├— ingles.js <-- (ej. 10 canciones inventadas para tests)

| | | ├— peliculas.js <-- (ej. 10 canciones inventadas para tests)

| | | ├— series.js <-- (ej. 10 canciones inventadas para tests)

| | | ├— tv.js <-- (ej. 10 canciones inventadas para tests)

| | | └— anuncios.js <-- (ej. 10 canciones inventadas para tests)

| | ├— 90s/ <-- ¡Asegurarse de que existan y estén pobladas las que se usen!

```
|   |   |   └── ...          (ej. espanol.js, ingles.js, etc., con al menos 4 canciones CADA UNO)
|   |   ├── 00s/
|   |   |   └── ...
|   |   ├── 10s/
|   |   |   └── ...
|   |   └── Actual/
|   |       └── ...
|   ├── img/                 (adivina.png, etc.)
|   └── audio/               <-- Carpetas de audios MP3. ¡Debe reflejar EXACTAMENTE la estructura de data/songs/!
|       ├── 80s/
|       |   ├── espanol/     <-- Aquí van los .mp3 de español de los 80s
|       |   ├── ingles/      <-- Aquí van los .mp3 de inglés de los 80s
|       |   └── ...          (y así para todas las categorías/décadas)
|       └── ...
└── backend/
    ├── server.js            <-- Lógica del servidor, API REST.
    ├── package.json         (dependencias de Node.js)
    └── ...
```

---

4. Estado de Archivos Clave y Puntos de Atención (Frontend)

- **frontend/index.html:**
    - **Propósito:** Es el esqueleto de la aplicación.
    - **Contenido:** NO debe tener etiquetas <style> con CSS incrustado ni etiquetas <script> con JS incrustado. SOLO debe contener los enlaces a css/style.css, js/songs-loader.js y js/main.js en el <body> al final, en ese orden.
    - **Verificación:** Asegurarse de que la y estén presentes.
- **frontend/css/style.css:**
    - **Propósito:** Todos los estilos CSS.
    - **Contenido:** Solo CSS puro. NO debe tener etiquetas HTML como `` o <style>.
- **frontend/js/songs-loader.js:**

    - **Propósito:** Gestiona la carga asíncrona de los archivos de datos de canciones.
    - **CRÍTICO:**
        - window.allSongsByDecadeAndCategory: Objeto global que almacena todas las canciones cargadas. configuracionCanciones en main.js hace proxy a este objeto.
        - allDecadesDefined (constante): **Esta lista DEBE contener SOLO las décadas para las cuales has creado y poblado archivos .js de canciones** (ej. ['80s', '90s', '00s', '10s', 'Actual']). Si hay más décadas aquí de las que tienes archivos, se generarán errores 404 en la consola.
        - loadSongsForDecadeAndCategory(decade, category): Carga un archivo .js específico de data/songs/.
        - loadAllSongs(): Consolida las canciones de todas las allDecadesDefined en window.allSongsByDecadeAndCategory['Todas']['consolidated'].
        - **Verificación:** Cada objeto de canción en los archivos .js de datos **DEBE** tener originalDecade y originalCategory para que la reproducción de audio funcione correctamente, especialmente en el modo "Todas las Décadas". songs-loader.js intenta añadirlas al vuelo si faltan, pero es mejor que vengan del origen.
- **frontend/js/main.js:**
    - **Propósito:** Contiene la lógica central del juego, UI y eventos.
    - **CRÍTICO:**
        - decadeNames (constante): Mapea los IDs de década a nombres legibles (ej. '80s': 'Década de los 80'). Asegurarse de que solo contenga las décadas que vas a usar.
        - categoryNames (constante): Mapea los IDs de categoría a nombres legibles.
        - window.onload o loginUser/setPlayerName: Deben llamar a generateDecadeButtons() para que el menú de décadas se construya al entrar en la pantalla decade-selection-screen.
        - generateDecadeButtons():
            - decadesOrder (constante): **Esta lista DEBE contener SOLO las décadas para las cuales quieres que se muestre un botón** (ej. ['80s', '90s', '00s', '10s', 'Actual']).
            - Solo crea un botón de década si configuracionCanciones[decadeId] tiene **al menos una categoría con 4 o más canciones válidas**.
        - selectDecade(): Maneja la selección de década (incluyendo "Todas las Décadas") y llama a loadSongsForDecadeAndCategory().

        - playAudioSnippet(): **CRÍTICO:** Utiliza currentQuestion.originalDecade y currentQuestion.originalCategory para construir la ruta del audio. Si faltan, el audio no se reproduce.

- **Archivos de datos de canciones (frontend/data/songs/[decade]/[category].js):**
    - **Propósito:** Almacenar arrays de objetos de canciones para cada combinación de década y categoría.
    - **CRÍTICO:**
        - **Contenido:** Cada archivo .js solo debe contener la asignación al objeto global window.allSongsByDecadeAndCategory (ej., window.allSongsByDecadeAndCategory['80s'].espanol = [...]). NO debe tener const configuracionCanciones = { ... } envolvente.
        - **Suficiencia de Canciones:** Cada array de canciones debe contener **al menos 4 objetos de canción** para que su categoría/década sea considerada "jugable" y su botón se muestre.
        - **Metadatos de Canción:** CADA objeto de canción en CADA archivo debe tener estas propiedades **correctamente definidas**:

JavaScript

```
{
   file: 'nombre-archivo.mp3',
   display: 'Artista - Título',
   listenUrl: 'URL_a_Spotify_o_YouTube',
   platform: 'spotify' o 'youtube',
   originalDecade: '80s',      // <--- ¡VALOR DE LA DÉCADA CORRECTO!
   originalCategory: 'espanol' // <--- ¡VALOR DE LA CATEGORÍA CORRECTO!
},
```

¡Los errores 404 de audio y la falta de botones de década provienen PRINCIPALMENTE de la ausencia o incorrección de originalDecade/originalCategory en los objetos de canción y de la falta de al menos 4 canciones por categoría!

---

5. Estado de Archivos Clave y Puntos de Atención (Backend)

- **backend/server.js:**
    - **Propósito:** Servidor de API REST, manejo de usuarios, puntuaciones, historial.
    - **CRÍTICO:** Los esquemas de Mongoose scoreSchema y gameHistorySchema **DEBEN** incluir el campo decade: { type: String, required: true }.
    - Las rutas /api/scores y /api/gamehistory ya están actualizadas para recibir y guardar la decade correctamente.

- **Verificación:** Si cambiaste los esquemas después de crear datos, es **recomendable vaciar las colecciones scores y gamehistories en MongoDB Atlas/Railway** antes de probar nuevas partidas para evitar errores de validación de datos antiguos.

---

6. Comandos Esenciales de Git y Despliegue

¡CRÍTICO! Todos los comandos git se ejecutan desde la RAÍZ de tu proyecto (E:\- Mi juego).

1. **Navegar a la raíz del proyecto (en tu terminal/VSC integrado):**

Bash

cd "E:\- Mi juego"

2. **Verificar el estado de los cambios:**

Bash

git status

(Te mostrará archivos modificados, nuevos, etc., en frontend/, backend/, data/songs/, etc.)

3. **Añadir todos los cambios al área de preparación:**

Bash

git add .

4. **Realizar el commit (guardar la instantánea de cambios):**

Bash

git commit -m "Descripción clara de los cambios realizados (ej., 'fix: Corregido contenido de canciones y errores de despliegue')"

5. **Subir los cambios al repositorio de GitHub (para Netlify y Railway):**

Bash

git push origin main

(Asumiendo que tu rama principal es main).

6. **Redesplegar en Netlify (si los cambios no se reflejan automáticamente):**
    - Inicia sesión en Netlify.
    - Ve a tu sitio (adivinala.netlify.app).
    - Haz clic en "Deploys".
    - Haz clic en el despliegue más reciente (o en "Trigger deploy" -> "Deploy site").
    - **Si los problemas persisten:** Haz clic en "Clear cache & deploy site" para forzar una reconstrucción completa.
7. **Redesplegar en Railway (si hiciste cambios en backend/server.js y no se reflejan):**

- Railway suele detectar los pushes a la rama conectada automáticamente.
- Si no se redepliega, puedes ir al panel de Railway de tu proyecto, seleccionar el servicio backend, y buscar la opción "Deploy" o "Restart" para forzar la actualización.

Conclusión para el Retomar:

El problema principal ahora mismo es la **calidad y exactitud de los datos en frontend/data/songs/**. Debes asegurarte de que:

1. La estructura de carpetas data/songs/[decade]/[category].js sea perfecta.
2. Cada archivo .js contenga un array con al menos 4 objetos de canción.
3. CADA objeto de canción tenga las propiedades originalDecade y originalCategory correctas.
4. La asignación window.allSongsByDecadeAndCategory['DECADA'].CATEGORIA = [...] dentro de cada archivo .js sea precisa.

Una vez que esa base de datos de canciones esté impecable, el resto del código (que parece estar bien) debería funcionar y los botones de décadas aparecerán.