EnglishApp - Documentación Técnica Completa

Arquitectura del Sistema

Patrón de Arquitectura

La aplicación sigue una arquitectura **Component-Based** con React, utilizando:

- **Single Page Application (SPA)** con React Router
- **Context API** para gestión de estado global
- **Custom Hooks** para lógica reutilizable
- **Service Workers** para funcionalidad offline

Flujo de Datos

```
Usuario → Componente → Hook/Custom Logic → Context/State → Supabase → UI Update
```

🛠️ Stack Tecnológico

Frontend Core

- **React 18.2.0**: Biblioteca principal para UI
- **Vite 4.4.5**: Build tool y dev server
- **React Router DOM 6.8.1**: Navegación SPA

UI/UX Libraries

- **Framer Motion 10.16.4**: Animaciones avanzadas
- **React Icons 5.5.0**: Iconografía moderna
- **CSS3**: Estilos con efectos modernos

Audio System

- **Howler.js 2.2.4**: Gestión de audio avanzada
- **Web Audio API**: Nativo del navegador

Data Management

- **Supabase 2.50.0**: Backend-as-a-Service
- **Recharts 3.0.0**: Gráficos y visualizaciones

Export & Utilities

- **jsPDF 3.0.1**: Generación de PDFs
- **html2canvas 1.4.1**: Captura de pantalla para PDFs
- **xlsx 0.18.5**: Exportación a Excel

Development Tools

- **ESLint**: Linting de código
- **Vitest**: Testing framework
- **Testing Library**: Testing de componentes

🗁 Estructura del Proyecto

```
englishapp-web-template/
├── public/                    # Archivos estáticos
│   ├── assets/               # Recursos multimedia
│   │   ├── images/           # Imágenes de juegos
│   │   └── sounds/           # Efectos de sonido
│   ├── _headers              # Headers de Netlify
│   ├── _redirects            # Redirecciones SPA
│   ├── manifest.json         # PWA manifest
│   └── sw.js                 # Service Worker
├── src/
│   ├── components/           # Componentes reutilizables
│   ├── games/                # Lógica de juegos
│   ├── pages/                # Páginas principales
│   ├── styles/               # Estilos globales
│   └── utils/                # Utilidades y hooks
├── package.json              # Dependencias y scripts
├── vite.config.js            # Configuración de Vite
└── netlify.toml              # Configuración de Netlify
```

🧩 Componentes Principales

Core Components

`AssetPreloader.jsx`

- **Propósito**: Precarga todos los assets antes de mostrar la app
- **Funcionalidad**:
  - Carga imágenes y sonidos
  - Muestra pantalla de carga
  - Maneja errores de carga

`BackgroundAudio.jsx`

- **Propósito**: Gestión de música de fondo
- **Características**:
  - Reproducción automática
  - Control de volumen
  - Persistencia de preferencias

`FloatingParticles.jsx`

- **Propósito**: Efectos visuales decorativos
- **Tecnología**: CSS animations + Framer Motion

`MusicToggle.jsx` & `VolumeControls.jsx`

- **Propósito**: Controles de audio
- **Integración**: Con Howler.js y Context API

`ThemeToggle.jsx`

- **Propósito**: Cambio entre tema claro/oscuro
- **Estado**: Persistido en localStorage

`OptimizedImage.jsx`

- **Propósito**: Carga optimizada de imágenes
- **Características**: Lazy loading, fallbacks

Game Components

Cada juego tiene su propia estructura:

```
games/[GameName]/
├── [GameName].jsx          # Componente principal
├── [GameName].css          # Estilos específicos
└── (archivos adicionales)
```

🎮 Sistema de Juegos

Arquitectura de Juegos

Cada juego sigue un patrón común:

1. **Theme Selector**: Selección de categoría
2. **Game Component**: Lógica principal del juego
3. **State Management**: Gestión de progreso y puntuación
4. **Audio Integration**: Efectos de sonido específicos

Juegos Implementados

1. Memory Game

- **Objetivo**: Encontrar pares de cartas
- **Temáticas**: Frutas, animales, colores, familia
- **Métricas**: Tiempo, intentos, puntuación

2. Typing Game

- **Objetivo**: Escribir palabras correctamente
- **Dificultades**: Fácil, Medio, Difícil
- **Métricas**: WPM, accuracy, errores

3. Math Game

- **Objetivo**: Resolver problemas matemáticos
- **Tipos**: Suma, resta, multiplicación
- **Temáticas**: Números y partes del cuerpo

4. Sorting Game

- **Objetivo**: Ordenar elementos secuencialmente
- **Temáticas**: Días, meses, números
- **Métricas**: Tiempo, precisión

5. Sound Matching Game

- **Objetivo**: Emparejar sonidos con imágenes
- **Tecnología**: Web Audio API + Howler.js
- **Temáticas**: Animales, objetos

6. Identification Game

- **Objetivo**: Identificar objetos por nombre
- **Métricas**: Tiempo de respuesta, precisión

🔄 Gestión de Estado

Context API Implementation

themeContext.jsx

```
// Gestión de tema global
const ThemeContext = createContext();
const ThemeProvider = ({ children }) => {
  const [isDark, setIsDark] = useState(false);
  // Lógica de persistencia y cambio de tema
};
```

musicState.js

```
// Estado global de audio
export const useMusicState = () => {
```

```
    // Gestión de música de fondo
    // Control de volumen
    // Persistencia de preferencias
};
```

Custom Hooks

`useGameStats.js`

- **Propósito**: Gestión de estadísticas de juegos
- **Funcionalidades**:
  - Guardar puntuaciones
  - Calcular promedios
  - Filtros por fecha/juego

`useServiceWorker.js`

- **Propósito**: Registro y gestión de SW
- **Características**:
  - Registro automático
  - Manejo de actualizaciones
  - Funcionalidad offline

🎵 Sistema de Audio

Arquitectura de Audio

```
AudioManager (Howler.js)
├── Background Music
├── Sound Effects
├── Volume Controls
└── Audio Context
```

Implementación

`soundManager.js`

```
// Gestión centralizada de audio
export class SoundManager {
  constructor() {
    this.sounds = new Map();
    this.backgroundMusic = null;
  }

  playSound(soundName) { /* ... */ }
  setVolume(volume) { /* ... */ }
```

```
  toggleMute() { /* ... */ }
}
```

Tipos de Audio

1. **Música de Fondo**: Reproducción continua
2. **Efectos de Sonido**: Interacciones del usuario
3. **Audio Educativo**: Pronunciación de palabras

🗄️ Base de Datos y Persistencia

Supabase Integration

Configuración (`supabase.js`)

```
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL;
const supabaseKey = import.meta.env.VITE_SUPABASE_ANON_KEY;

export const supabase = createClient(supabaseUrl, supabaseKey);
```

Tablas Principales

1. **game_statistics**
   - `id`: UUID (Primary Key)
   - `game_type`: String (memory, typing, etc.)
   - `theme`: String (fruits, animals, etc.)
   - `score`: Integer
   - `time_spent`: Integer (seconds)
   - `created_at`: Timestamp
2. **user_progress**
   - `id`: UUID (Primary Key)
   - `user_id`: String
   - `total_games`: Integer
   - `total_time`: Integer
   - `last_played`: Timestamp

Local Storage

- **Tema**: `theme-preference`
- **Audio**: `music-volume`, `sound-volume`, `music-enabled`
- **Progreso**: `game-progress`, `statistics`

PWA y Service Workers

Service Worker (`sw.js`)

```
// Cache strategy: Cache First, Network Fallback
const CACHE_NAME = 'englishapp-v1';
const urlsToCache = [
  '/',
  '/static/js/bundle.js',
  '/static/css/main.css',
  // ... otros recursos
];
```

Manifest (`manifest.json`)

```
{
  "name": "EnglishApp - Colegio Arauco",
  "short_name": "EnglishApp",
  "description": "Aplicación educativa para aprender inglés",
  "start_url": "/",
  "display": "standalone",
  "theme_color": "#4CAF50",
  "background_color": "#ffffff",
  "icons": [
    {
      "src": "android-chrome-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}
```

Características PWA

- ✅ **Instalable**: Como app nativa
- ✅ **Offline**: Funcionamiento sin conexión
- ✅ **Responsive**: Adaptable a todos los dispositivos
- ✅ **Fast Loading**: Caching inteligente

Sistema de Temas

Implementación

CSS Variables

```
:root {
  --primary-color: #4CAF50;
  --secondary-color: #2196F3;
  --background-color: #ffffff;
  --text-color: #333333;
  --accent-color: #FF9800;
}

[data-theme="dark"] {
  --background-color: #1a1a1a;
  --text-color: #ffffff;
  /* ... otras variables */
}
```

Context Provider

```
const ThemeProvider = ({ children }) => {
  const [isDark, setIsDark] = useState(() => {
    return localStorage.getItem('theme-preference') === 'dark';
  });

  useEffect(() => {
    document.documentElement.setAttribute('data-theme', isDark ? 'dark' :
'light');
    localStorage.setItem('theme-preference', isDark ? 'dark' : 'light');
  }, [isDark]);

  return (
    <ThemeContext.Provider value={{ isDark, setIsDark }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

📊 Exportación de Datos

PDF Generation

`jspdf` + `html2canvas`

```
import jsPDF from 'jspdf';
import html2canvas from 'html2canvas';

const generatePDF = async (element) => {
  const canvas = await html2canvas(element);
  const imgData = canvas.toDataURL('image/png');
```

```
  const pdf = new jsPDF();
  pdf.addImage(imgData, 'PNG', 0, 0);
  pdf.save('reporte.pdf');
};
```

Excel Export

`xlsx` Library

```
import * as XLSX from 'xlsx';

const exportToExcel = (data) => {
  const ws = XLSX.utils.json_to_sheet(data);
  const wb = XLSX.utils.book_new();
  XLSX.utils.book_append_sheet(wb, ws, 'Estadísticas');
  XLSX.writeFile(wb, 'estadisticas.xlsx');
};
```

⚙️ Configuración de Desarrollo

Variables de Entorno

```
# .env.local
VITE_SUPABASE_URL=your_supabase_url
VITE_SUPABASE_ANON_KEY=your_supabase_anon_key
VITE_APP_TITLE=EnglishApp
VITE_APP_VERSION=1.0.0
```

Scripts de Desarrollo

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext js,jsx",
    "test": "vitest",
    "test:ui": "vitest --ui",
    "test:coverage": "vitest run --coverage"
  }
}
```

Configuración de Vite

```
// vite.config.js
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000,
    open: true
  },
  build: {
    outDir: 'dist',
    sourcemap: true
  }
});
```

🚀 Deployment

Netlify Configuration

`netlify.toml`

```
[build]
  command = "npm run build"
  publish = "dist"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200

[build.environment]
  NODE_VERSION = "18"
```

Headers (`_headers`)

```
/*
  X-Frame-Options: DENY
  X-XSS-Protection: 1; mode=block
  X-Content-Type-Options: nosniff
  Referrer-Policy: strict-origin-when-cross-origin
```

Build Process

1. **Pre-build**: Instalación de dependencias

2. **Build**: Compilación con Vite
3. **Post-build**: Optimización de assets
4. **Deploy**: Subida a CDN de Netlify

🧪 Testing

Testing Stack

- **Vitest**: Test runner
- **Testing Library**: Testing de componentes
- **jsdom**: DOM simulation

Test Structure

```
tests/
├── components/          # Tests de componentes
├── games/               # Tests de juegos
├── utils/               # Tests de utilidades
└── integration/         # Tests de integración
```

Ejemplo de Test

```
import { render, screen } from '@testing-library/react';
import { describe, it, expect } from 'vitest';
import Home from '../src/pages/Home';

describe('Home Component', () => {
  it('renders all game cards', () => {
    render(<Home />);
    expect(screen.getByText('Memory Game')).toBeInTheDocument();
    expect(screen.getByText('Typing Game')).toBeInTheDocument();
  });
});
```

⚡ Optimizaciones

Performance

1. **Code Splitting**: Lazy loading de componentes
2. **Asset Optimization**: Compresión de imágenes y audio
3. **Caching**: Service Worker para recursos estáticos
4. **Bundle Optimization**: Tree shaking y minificación

SEO

1. **Meta Tags**: Optimizados para motores de búsqueda
2. **Structured Data**: Schema.org markup
3. **Sitemap**: Generación automática
4. **Performance Metrics**: Core Web Vitals

Accessibility

1. **ARIA Labels**: Navegación por teclado
2. **Color Contrast**: Cumplimiento WCAG 2.1
3. **Screen Reader**: Compatibilidad total
4. **Keyboard Navigation**: Navegación completa

🔧 Mantenimiento

Logs y Monitoreo

- **Error Tracking**: Captura de errores en producción
- **Performance Monitoring**: Métricas de rendimiento
- **User Analytics**: Comportamiento de usuarios

Actualizaciones

1. **Dependencias**: Actualización regular
2. **Security Patches**: Parches de seguridad
3. **Feature Updates**: Nuevas funcionalidades
4. **Bug Fixes**: Corrección de errores

📚 Recursos Adicionales

Documentación

- React Documentation
- Vite Documentation
- Supabase Documentation
- Framer Motion Documentation

Herramientas de Desarrollo

- React Developer Tools
- Vite Inspector
- Supabase Studio

Versión: 1.0.0
Última actualización: Junio 2025

🕹️ Cómo agregar un nuevo juego

1. **Crea una carpeta en `src/games/` con el nombre del juego**
   - Ejemplo: `src/games/MyNewGame/`
2. **Crea el componente principal**
   - Archivo: `MyNewGame.jsx`
   - Debe exportar un componente React funcional.
3. **Crea el archivo de estilos**
   - Archivo: `MyNewGame.css`
   - Importa el CSS en el componente principal.
4. **Agrega la lógica del juego**
   - Usa hooks para manejar estado y lógica.
   - Integra audio usando `soundManager.js` si es necesario.
   - Si el juego requiere estadísticas, usa `progressManager.js` y/o Supabase.
5. **Agrega la opción de navegación**
   - Modifica la página principal (`Home.jsx`) para incluir el nuevo juego en la grilla.
   - Si el juego tiene selector de temática, crea un componente en `src/pages/` siguiendo el patrón de los existentes.
6. **Actualiza la guía de usuario y técnica**
   - Documenta el nuevo juego en ambos archivos markdown.
7. **Ejemplo de estructura mínima:**

```
// src/games/MyNewGame/MyNewGame.jsx
import React from 'react';
import './MyNewGame.css';

export default function MyNewGame() {
  // Lógica y estado aquí
  return <div>My New Game</div>;
}
```

Cómo agregar una temática a un juego

1. **Identifica el juego y localiza la estructura de temáticas**
   - Por lo general, es un objeto o array en el archivo principal del juego (por ejemplo, `themes`, `wordSets`, `themeData`, etc.).
2. **Agrega la nueva temática como una nueva clave o elemento**
   - Ejemplo para Memory Game:

```
const themes = {
  ...
  newTheme: [
    { word: 'example1', image: 'example1.webp' },
    { word: 'example2', image: 'example2.webp' },
    // ...
  ],
};
```

3. **Asegúrate de que el selector de temática la incluya**
    - Modifica el componente selector correspondiente para mostrar la nueva opción.
4. **Agrega los assets necesarios**
    - Coloca las imágenes o sonidos en `public/assets/images/` o `public/assets/sounds/`.
5. **Prueba la integración**
    - Inicia la app y verifica que la temática aparece y funciona correctamente.
6. **Actualiza la documentación**
    - Añade la temática en la guía de usuario y técnica.
