

Plan de Tareas Detallado - OPSCANNER

Generado: 11/2/2026

1. Código de carta no desaparece

Lista: BUGS

Prioridad:  Alta

Estimación: 2-4 horas

Problema

Después de escanear una carta, el código detectado permanece visible en la UI cuando debería desaparecer tras el feedback de éxito.

Análisis Técnico

El estado `(detectionState.lastSavedCode)` no se está limpiando correctamente después del timeout de animación.

Solución Propuesta

```
typescript

// En useCardScanner.ts
setTimeout(() => {
  setDetectionState(prev => ({
    ...prev,
    lastSavedCode: null,
    isDetecting: false
  }));
}, 2000); // Limpiar después de 2 segundos
```

Archivos Afectados

- hooks/useCardScanner.ts
- components/DetectionFeedback.tsx

Tests

1. Escanear una carta
2. Verificar que el feedback aparece
3. Verificar que desaparece completamente tras 2 segundos
4. Escanear otra carta y verificar que no hay interferencias

Checklist

- Identificar el timeout actual en DetectionFeedback
 - Añadir cleanup de lastSavedCode en useCardScanner
 - Verificar que FadeOutUp funciona correctamente
 - Testing en dispositivo físico
-

2. Revisar textos de los sets en cardDetail

Lista: BUGS

Prioridad:  Media

Estimación: 1-2 horas

Problema

Los nombres de sets en CardDetailScreen no se muestran correctamente o están incompletos.

Análisis Técnico

El parser `cardCodeParser.getSetName()` puede no tener todos los sets mapeados.

Solución Propuesta

1. Crear mapeo completo de sets en `utils/constants.ts`:

typescript

```
export const SET_NAMES = {  
    'OP01': 'Romance Dawn',  
    'OP02': 'Paramount War',  
    'OP03': 'Pillars of Strength',  
    'OP04': 'Kingdoms of Intrigue',  
    'OP05': 'Awakening of the New Era',  
    'OP06': 'Wings of the Captain',  
    'OP07': '500 Years in the Future',  
    'OP08': 'Two Legends',  
    'EB01': 'Memorial Collection',  
    'ST01': 'Straw Hat Crew',  
    'ST02': 'Worst Generation',  
    // ... completar todos  
};
```

2. Actualizar `cardCodeParser` para usar este mapeo

Archivos Afectados

- utils/constants.ts
- utils/cardCodeParser.ts
- screens/CardDetailScreen.tsx

Checklist

- Investigar lista completa de sets del juego
 - Crear constante SET_NAMES con todos los sets
 - Actualizar getSetName() para usar la constante
 - Verificar en CardDetailScreen que se muestran correctamente
-

3. Imágenes Alters

Lista: BUGS

Prioridad:  Alta

Estimación: 8-12 horas

Problema

Las URLs de imágenes de cartas alternate art no se generan correctamente. Necesitamos un sistema robusto para identificar y generar las URLs correctas según el tipo de variante.

Análisis Técnico

Actualmente en `scripts/import_cards.js` se usa:

```
javascript
const generateUrlFromPythonLogic = (card) => {
  const BASE_IMG_URL = "https://en.onepiece-cardgame.com/images/cardlist/card/";
  const code = card.id_normal;
  let suffix = "";

  if (card.name.includes('V2')) suffix = "_p1";
  else if (card.name.includes('V3')) suffix = "_p2";

  return `${BASE_IMG_URL}${code}${suffix}.png`;
};
```

Esto es insuficiente para todas las variantes.

Tipos de Variantes

1. **Normal:** OP01-001.png
2. **Parallel (V2):** OP01-001_p1.png

3. **Manga/Special (V3):** OP01-001_p2.png
4. **SP (Special):** Investigar patrón
5. **Winner/Judge:** Investigar patrón
6. **Promo:** Código diferente (P-XXX)

Solución Propuesta

Script de Verificación

```
javascript

// scripts/verify_card_images.js
const axios = require('axios');
const { supabase } = require('../lib/supabase');

async function verifyImages() {
  const { data: cards } = await supabase
    .from('cards')
    .select('id, code, variant, image_url');

  for (const card of cards) {
    try {
      const response = await axios.head(card.image_url);
      if (response.status !== 200) {
        console.log(`❌ Imagen no encontrada: ${card.code} (${card.variant})`);
        // Intentar patrones alternativos
        await tryAlternativePatterns(card);
      }
    } catch (error) {
      console.log(`❌ Error: ${card.code}`);
    }
  }
}

async function tryAlternativePatterns(card) {
  const patterns = [
    `${card.code}.png`,
    `${card.code}_p1.png`,
    `${card.code}_p2.png`,
    `${card.code}_parallel.png`,
    // etc...
  ];

  // Probar cada patrón
}
```

Actualizar Generador de URLs

javascript

```
function generateImageUrl(card) {
  const BASE_URL = "https://en.onepiece-cardgame.com/images/cardlist/card/";
  const code = card.code;

  // Mapeo de variantes a sufijos
  const variantSuffixes = {
    'Normal': '',
    'Parallel': '_p1',
    'Parallel (V2)': '_p1',
    'Manga / Special (V3)': '_p2',
    'SP': '_sp', // A confirmar
    'Winner': '_winner', // A confirmar
    'Judge': '_judge', // A confirmar
  };

  const suffix = variantSuffixes[card.variant] || '';
  return `${BASE_URL}${code}${suffix}.png`;
}
```

Plan de Acción

1. Investigación manual de patrones (revisar 10-20 cartas de cada tipo)
2. Crear script de verificación
3. Ejecutar script sobre toda la BD
4. Documentar patrones encontrados
5. Actualizar import_cards.js
6. Re-importar todas las cartas

Archivos Afectados

- scripts/import_cards.js
- scripts/verify_card_images.js (nuevo)
- Documentación de URLs (nuevo)

Checklist

- Investigar patrones de URLs para cada variante
- Crear script verify_card_images.js
- Ejecutar verificación sobre BD actual

- Documentar todos los patrones encontrados
 - Actualizar generateImageUrl() con todos los patrones
 - Re-ejecutar import_cards.js con nueva lógica
 - Verificar en app que todas las imágenes cargan
-

4. Perfil de usuario

Lista: USERS

Prioridad:  Baja

Estimación: 6-8 horas

Objetivo

Crear pantalla de perfil de usuario con información personal, estadísticas y configuración.

Funcionalidades

1. Información Personal

- Avatar (editable con image picker)
- Nombre de usuario
- Email (no editable)
- Fecha de registro

2. Estadísticas

- Cartas totales en colección
- Cartas únicas
- Sets completados
- Valor total de colección
- Carta más valiosa

3. Configuración

- Notificaciones push
- Vibración al escanear
- Sonidos
- Tema (claro/oscuro)

Implementación Técnica

Nueva Tabla en BD

```
sql
```

```
ALTER TABLE profiles ADD COLUMN IF NOT EXISTS avatar_url text;
ALTER TABLE profiles ADD COLUMN IF NOT EXISTS display_name text;
ALTER TABLE profiles ADD COLUMN IF NOT EXISTS bio text;
ALTER TABLE profiles ADD COLUMN IF NOT EXISTS preferences jsonb DEFAULT '{}';
```

Estructura de Archivos

```
screens/ProfileScreen.tsx
components/profile/
    ├── ProfileHeader.tsx
    ├── StatsCard.tsx
    ├── SettingsSection.tsx
    └── AvatarPicker.tsx
```

Screen Principal

```
typescript
```

```

// screens/ProfileScreen.tsx
export const ProfileScreen = () => {
  const { user } = useAuth();
  const { stats } = useCollection();
  const [profile, setProfile] = useState(null);

  useEffect(() => {
    loadProfile();
  }, []);

  const loadProfile = async () => {
    const { data } = await supabase
      .from('profiles')
      .select('*')
      .eq('id', user.id)
      .single();
    setProfile(data);
  };

  const updateAvatar = async (uri) => {
    // Upload a Supabase Storage
    // Actualizar avatar_url en profiles
  };

  return (
    <ScrollView>
      <ProfileHeader profile={profile} onAvatarChange={updateAvatar} />
      <StatsCard stats={stats} />
      <SettingsSection />
    < ScrollView>
  );
};

```

Navegación

Añadir botón de perfil en header de CollectionScreen

Checklist

- Crear nueva pantalla ProfileScreen
- Implementar ProfileHeader con avatar editable
- Crear componente StatsCard
- Implementar SettingsSection
- Añadir columnas a tabla profiles
- Configurar Supabase Storage para avatares

- Añadir navegación desde CollectionScreen
 - Testing completo
-

5. Amigos

Lista: USERS

Prioridad:  Media

Estimación: 12-16 horas

Objetivo

Sistema de amigos para compartir colecciones y facilitar intercambios.

Funcionalidades

1. Búsqueda de Usuarios

- Por username
- Por código de amigo (generado)

2. Gestión de Amigos

- Enviar solicitud
- Aceptar/rechazar solicitudes
- Lista de amigos
- Eliminar amigo

3. Visualización

- Ver colección de amigos
- Ver mazos de amigos (si implementado)
- Comparar colecciones

Modelo de Datos

Nueva Tabla: friend_requests

sql

```
CREATE TABLE friend_requests (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    sender_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    receiver_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    status text CHECK (status IN ('pending', 'accepted', 'rejected')),
    created_at timestamp DEFAULT now(),
    updated_at timestamp DEFAULT now(),
    UNIQUE(sender_id, receiver_id)
);
```

Nueva Tabla: friendships

sql

```
CREATE TABLE friendships (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user1_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    user2_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    created_at timestamp DEFAULT now(),
    CHECK (user1_id < user2_id), -- Evitar duplicados
    UNIQUE(user1_id, user2_id)
);
```

Actualizar profiles

sql

```
ALTER TABLE profiles ADD COLUMN friend_code text UNIQUE;
-- Generar automáticamente: SUBSTR(MD5(RANDOM())::text, 1, 8)
```

Implementación

Servicio de Amigos

typescript

```

// services/friendsService.ts

export const friendsService = {

  async sendFriendRequest(receiverId: string) {
    const senderId = await supabaseService.getCurrentUserId();
    const { error } = await supabase
      .from('friend_requests')
      .insert({ sender_id: senderId, receiver_id: receiverId, status: 'pending' });
    return !error;
  },

  async acceptRequest(requestId: string) {
    // 1. Actualizar request a 'accepted'
    // 2. Crear entrada en friendships
    // 3. Notificar al sender (si hay notificaciones)
  },
}

async getFriends() {
  const userId = await supabaseService.getCurrentUserId();
  const { data } = await supabase
    .from('friendships')
    .select(
      '*',
      user1:profiles!user1_id(id, username, avatar_url),
      user2:profiles!user2_id(id, username, avatar_url)
    )
    .or(`user1_id.eq.${userId},user2_id.eq.${userId}`);
}

// Formatear para devolver solo el amigo (no el usuario actual)
return data.map(f=>
  f.user1_id === userId ? f.user2 : f.user1
);

async getFriendCollection(friendId: string) {
  const { data } = await supabase
    .from('user_collection')
    .select('*', card:cards('*'))
    .eq('user_id', friendId);
  return data;
}
};

```

Pantallas

screens/

```
└── FriendsScreen.tsx (lista de amigos + solicitudes)
    ├── AddFriendScreen.tsx (búsqueda)
    └── FriendProfileScreen.tsx (colección del amigo)
```

Navegación

Añadir nueva stack de Friends en navegación principal

Checklist

- Crear tablas friend_requests y friendships
 - Generar friend_code para usuarios existentes
 - Implementar friendsService con todos los métodos
 - Crear FriendsScreen con tabs (amigos/solicitudes)
 - Implementar AddFriendScreen con búsqueda
 - Crear FriendProfileScreen para ver colección
 - Añadir navegación a stack principal
 - Testing de todos los flujos
-

6. Préstamo de cartas

Lista: USERS

Prioridad:  Baja

Estimación: 10-14 horas

Objetivo

Sistema para registrar préstamos de cartas entre amigos.

Funcionalidades

1. Crear Préstamo

- Seleccionar amigo
- Seleccionar cartas de tu colección
- Fecha de préstamo
- Fecha de devolución esperada
- Notas opcionales

2. Gestión

- Ver préstamos activos (dadas/recibidas)
- Marcar como devuelto
- Recordatorios de devolución

- Historial de préstamos

3. Validaciones

- Solo prestar a amigos
- No prestar cartas ya prestadas
- Actualizar quantity en colección

Modelo de Datos

Nueva Tabla: card_loans

sql

```
CREATE TABLE card_loans (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    lender_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    borrower_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    card_id uuid REFERENCES cards(id),
    quantity integer DEFAULT 1,
    loaned_at timestamp DEFAULT now(),
    expected_return_date date,
    returned_at timestamp,
    status text CHECK (status IN ('active', 'returned', 'overdue')),
    notes text,
    created_at timestamp DEFAULT now()
);

CREATE INDEX idx_loans_lender ON card_loans(lender_id) WHERE status = 'active';
CREATE INDEX idx_loans_borrower ON card_loans(borrower_id) WHERE status = 'active';
```

Implementación

Servicio

typescript

```
// services/loanService.ts
export const loanService = {
  async createLoan(borrowerId, cardId, quantity, expectedReturn, notes) {
    const lenderId = await supabaseService.getCurrentUserId();

    // 1. Verificar que son amigos
    const areFriends = await friendsService.checkFriendship(lenderId, borrowerId);
    if (!areFriends) throw new Error("Solo puedes prestar a amigos");

    // 2. Verificar que tienes suficientes cartas
    const { data: collection } = await supabase
      .from('user_collection')
      .select('quantity')
      .eq('user_id', lenderId)
      .eq('card_id', cardId)
      .single();

    if (!collection || collection.quantity < quantity) {
      throw new Error("No tienes suficientes copias");
    }

    // 3. Crear préstamo
    const { data: loan, error } = await supabase
      .from('card_loans')
      .insert({
        lender_id: lenderId,
        borrower_id: borrowerId,
        card_id: cardId,
        quantity,
        expected_return_date: expectedReturn,
        status: 'active',
        notes
      })
      .select()
      .single();

    // 4. Actualizar quantity en colección (restar)
    await supabase
      .from('user_collection')
      .update({ quantity: collection.quantity - quantity })
      .eq('user_id', lenderId)
      .eq('card_id', cardId);

    return loan;
  },
}
```

```

async returnLoan(loanId) {
  // 1. Marcar como returned
  // 2. Sumar quantity de vuelta al lender
  // 3. Notificar al lender
  ,
}

async getActiveLoans() {
  const userId = await supabaseService.getCurrentUserId();
  const { data } = await supabase
    .from('card_loans')
    .select(
      *,
      card:cards(*),
      lender:profiles!lender_id(username),
      borrower:profiles!borrower_id(username)
    )
    .eq('status', 'active')
    .or(`lender_id.eq.${userId},borrower_id.eq.${userId}`);
}

return {
  lent: data.filter(l => l.lender_id === userId),
  borrowed: data.filter(l => l.borrower_id === userId)
};
};

async checkOverdueLoans() {
  // Cron job diario
  // Actualizar status a 'overdue' si passed expected_return_date
}
};

```

Pantallas

```

screens/
  └── LoansScreen.tsx (tabs: prestadas/recibidas)
  └── CreateLoanScreen.tsx (formulario)

```

Checklist

- Crear tabla card_loans con índices
- Implementar loanService completo
- Crear LoansScreen con tabs
- Implementar CreateLoanScreen con validaciones
- Actualizar lógica de quantity en colección
- Añadir función de recordatorios (opcional)

- Testing de edge cases
 - Documentar flujo completo
-

7. Cambiar parámetros de análisis en función de la luz disponible

Lista: SCANNER

Prioridad:  Media

Estimación: 6-8 horas

Objetivo

Ajustar automáticamente los parámetros de OCR según las condiciones de iluminación para mejorar la precisión.

Análisis Técnico

React Native Vision Camera proporciona información sobre exposición y luz ambiental que podemos usar para optimizar el OCR.

Parámetros a Ajustar

1. **Contraste de imagen** antes del OCR
2. **Frecuencia de captura** (más rápido en buena luz)
3. **Sugerencia visual** al usuario (activar flash)

Implementación

Detector de Luz

```
typescript

// hooks/useLightDetection.ts

import { useCameraDevice } from 'react-native-vision-camera';

export const useLightDetection = () => {
  const [lightLevel, setLightLevel] = useState<'low' | 'medium' | 'high'>('medium');

  const device = useCameraDevice('back');

  useEffect(() => {
    // Analizar metadata de frames
    // Vision Camera puede proporcionar ISO, exposición, etc.
  }, []);

  return lightLevel;
};
```

Ajuste Dinámico en Scanner

typescript

```
// screens/ScannerScreen.tsx
const lightLevel = useLightDetection();

const scanConfig = useMemo(() => {
  switch(lightLevel) {
    case 'low':
      return {
        throttle: 250,    // Más lento
        contrast: 1.5,   // Mayor contraste
        suggestFlash: true
      };
    case 'medium':
      return {
        throttle: 150,
        contrast: 1.2,
        suggestFlash: false
      };
    case 'high':
      return {
        throttle: 100,   // Más rápido
        contrast: 1.0,
        suggestFlash: false
      };
  }
}, [lightLevel]);

// Usar scanConfig.throttle en lugar de SCANNER_CONFIG.THROTTLE_MS
```

Procesamiento de Imagen

typescript

```
// utils/imageProcessing.ts
import { manipulateAsync } from 'expo-image-manipulator';

export async function preprocessImage(uri: string, contrast: number) {
  const result = await manipulateAsync(
    uri,
    [
      { resize: { width: 1000 } }, // Reducir tamaño
      // Ajustar brillo/contraste (requiere library adicional)
    ],
    { compress: 0.8, format: 'png' }
  );

  return result.uri;
}
```

Indicador Visual

typescript

```
// En ScanOverlay.tsx
{lightLevel === 'low' && (
  <View style={styles.lowLightWarning}>
    <Text>💡 Poca luz detectada</Text>
    <Text>Activa el flash para mejores resultados</Text>
  </View>
)}
```

Librerías Adicionales

bash

```
npm install expo-image-manipulator
```

Checklist

- Instalar expo-image-manipulator
- Crear hook useLightDetection
- Implementar preprocessImage con ajuste de contraste
- Añadir configuración dinámica en ScannerScreen
- Crear indicador visual en ScanOverlay
- Testing en diferentes condiciones de luz
- Optimizar rendimiento
- Documentar parámetros óptimos

8. Identificación mangas, sp y cartas especiales más allá de los alters

Lista: SCANNER

Prioridad:  Media

Estimación: 8-10 horas

Objetivo

Mejorar el sistema de detección para identificar correctamente cartas Manga, SP, Winner, Judge y otras variantes especiales.

Análisis del Problema

Actualmente solo detectamos "Normal" vs "Parallel" basándonos en el toggle AA. Necesitamos un sistema más sofisticado.

Tipos de Cartas Especiales

1. **Manga Rare:** Ilustración estilo manga
2. **SP (Special):** Versiones especiales con ilustración única
3. **Winner:** Cartas de torneo
4. **Judge:** Cartas de juez
5. **Promo:** Promocionales con código P-XXX
6. **Comic:** Variante especial

Estrategia de Detección

Opción A: Visual (Machine Learning)

Entrenar modelo ML para identificar estilo visual:

- Detectar estilo manga por características visuales
- Identificar bordes dorados (Winner/Judge)
- Reconocer watermarks especiales

Pros: Más preciso **Contras:** Complejo, requiere training data

Opción B: Código + Contexto

Analizar el código y buscar en BD variantes:

```
typescript
```

```
// Cuando se detecta OP01-001
// 1. Buscar todas las variantes en BD
// 2. Si existen múltiples, mostrar selector
// 3. Usuario elige cuál escaneó
```

Pros: Simple, preciso **Contras:** Requiere interacción manual

Opción C: Híbrido (RECOMENDADO)

1. Detectar código automáticamente
2. Si existen múltiples variantes en BD, mostrar preview rápido
3. Usuario confirma con tap (o auto-detectar si solo hay una)

Implementación Opción C

Actualizar useCardScanner

```
typescript

const processDetectedText = useCallback((text: string) => {
  // ... código actual ...

  if (currentCount >= SCANNER_CONFIG.REQUIRED_CONFIRMATIONS) {
    // Buscar TODAS las variantes de este código
    const variants = await supabaseService.getCardVariants(codeString);

    if (variants.length === 1) {
      // Solo una variante, guardar directamente
      await supabaseService.addCardToCollection(codeString, variants[0].is_foil);
    } else {
      // Múltiples variantes, mostrar selector
      setShowVariantSelector({
        code: codeString,
        variants: variants
      });
    }
  }
}, []);
```

Nuevo Componente: VariantSelector

```
typescript
```

```
// components/VariantSelector.tsx
export const VariantSelector = ({ code, variants, onSelect }) => {
  return (
    <Modal visible={true} animationType="slide">
      <View style={styles.container}>
        <Text style={styles.title}>Detectado: {code}</Text>
        <Text style={styles.subtitle}>¿Qué versión tienes?</Text>

        <ScrollView horizontal>
          {variants.map(variant => (
            <Pressable
              key={variant.id}
              style={styles.variantCard}
              onPress={() => onSelect(variant)}
            >
              <Image source={{ uri: variant.image_url }} />
              <Text>{variant.variant}</Text>
              {variant.rarity && <Text>{variant.rarity}</Text>}
            </Pressable>
          ))}
        </ScrollView>
      </View>
      <Modal>
    );
  );
}
```

Nuevo método en supabaseService

typescript

```
async getCardVariants(code: string) {
  const { data, error } = await supabase
    .from('cards')
    .select('*')
    .eq('code', code)
    .order('variant');

  return data || [];
}
```

Mejora Futura: Detección por Color

Analizar color dominante de la imagen:

- Normal: Colores estándar
- Parallel: Efecto holográfico (difícil de detectar)

- Manga: Blanco y negro predominante
- SP: Colores únicos

typescript

```
// utils/colorDetection.ts
export function analyzeCardColor(imageUri: string) {
  // Usar react-native-image-colors o similar
  // Determinar si es manga (B&W) o normal (color)
}
```

Checklist

- Implementar getCardVariants() en supabaseService
 - Crear componente VariantSelector
 - Integrar selector en flujo de escaneo
 - Añadir lógica para auto-selección si hay una sola variante
 - Diseñar UI del selector (imagenes + labels)
 - Testing con cartas de múltiples variantes
 - Investigar detección por color (opcional)
 - Documentar tipos de variantes
-

9. Añadir cartas promo, judge y winner

Lista: SCANNER

Prioridad:  Media

Estimación: 4-6 horas

Objetivo

Importar y gestionar cartas promocionales, de juez y de torneos en la base de datos.

Análisis

Estas cartas tienen códigos especiales:

- **Promo:** P-001, P-002, etc.
- **Judge:** J-001, J-002, etc.
- **Winner:** W-001, W-002, etc.
- **Staff:** S-001, etc.

Fuentes de Datos

1. **Official Website:** <https://en.onepiece-cardgame.com/cardlist/>
2. **Community Databases:** Limitless TCG, One Piece Top Decks
3. **Manual Entry:** Para cartas muy raras

Actualización del Parser

```
typescript

// utils/cardCodeParser.ts
// Actualizar PATTERN para incluir prefijos especiales
PATTERN: /((?:OP|EB|ST|PRB|P|J|W|S)[0-9O]{2,3})\s?[-—]?\s?([0-9O]{3})/i

// O más específicamente:
PATTERN: /((?:OP|EB|ST|PRB)[0-9O]{2}|(?:P|J|W|S)-[0-9O]{3})\s?[-—]?\s?([0-9O]{3})?/i
```

Script de Importación

```
javascript

// scripts/import_special_cards.js
const specialCards = [
  {
    code: 'P-001',
    name: 'Monkey D. Luffy',
    variant: 'Promo',
    rarity: 'P',
    set_code: 'PROMO',
    // ...
  },
  // ...
];

async function importSpecialCards() {
  for (const card of specialCards) {
    await supabase.from('cards').upsert(card, {
      onConflict: 'code, variant'
    });
  }
}
```

Actualización de Sets

```
sql
```

```
INSERT INTO sets (code, name, release_date) VALUES
    ('PROMO', 'Promotional Cards', '2024-01-01'),
    ('JUDGE', 'Judge Cards', '2024-01-01'),
    ('WINNER', 'Winner Cards', '2024-01-01')
ON CONFLICT (code) DO NOTHING;
```

UI Updates

En CollectionScreen, añadir filtro para sets especiales:

typescript

```
const SPECIAL_SETS = ['PROMO', 'JUDGE', 'WINNER', 'STAFF'];
```

✓ Checklist

- Investigar lista completa de cartas especiales
- Actualizar cardCodeParser para códigos P/J/W/S
- Crear sets especiales en BD
- Crear archivo special_cards.json con datos
- Crear script import_special_cards.js
- Ejecutar importación
- Actualizar filtros en CollectionScreen
- Testing de escaneo de cartas especiales
- Documentar códigos especiales

10. Crear deck

Lista: DECKS

Prioridad: ● Alta

Estimación: 16-20 horas

Objetivo

Sistema completo de creación y gestión de mazos con validación de reglas del juego.

Reglas de One Piece TCG

1. **50 cartas exactas** (Leader + 49 cartas del deck)
2. **1 Leader obligatorio**
3. **Máximo 4 copias** de cada carta (por nombre, excepto Leader)
4. **DON!! cards:** 10 por defecto (no cuentan en el deck)
5. **Colores:** Las cartas deben coincidir con los colores del Leader (o ser multi-color)

Modelo de Datos

Nueva Tabla: decks

sql

```
CREATE TABLE decks (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    name text NOT NULL,
    description text,
    leader_card_id uuid REFERENCES cards(id),
    is_public boolean DEFAULT false,
    is_valid boolean DEFAULT false,
    created_at timestamp DEFAULT now(),
    updated_at timestamp DEFAULT now()
);

CREATE INDEX idx_decks_user ON decks(user_id);
CREATE INDEX idx_decks_public ON decks(is_public) WHERE is_public = true;
```

Nueva Tabla: deck_cards

sql

```
CREATE TABLE deck_cards (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    deck_id uuid REFERENCES decks(id) ON DELETE CASCADE,
    card_id uuid REFERENCES cards(id),
    quantity integer CHECK (quantity >= 1 AND quantity <= 4),
    UNIQUE(deck_id, card_id)
);

CREATE INDEX idx_deck_cards_deck ON deck_cards(deck_id);
```

Servicio de Decks

typescript

```
// services/deckService.ts
export const deckService = {
  async createDeck(name: string, description?: string) {
    const userId = await supabaseService.getCurrentUserId();
    const { data, error } = await supabase
      .from('decks')
      .insert({ user_id: userId, name, description })
      .select()
      .single();
    return data;
  },
  async addCardToDeck(deckId: string, cardId: string, quantity: number = 1) {
    // 1. Verificar que el deck pertenece al usuario
    // 2. Verificar que no excede 4 copias
    // 3. Verificar que no excede 50 cartas
    // 4. Upsert en deck_cards
    const { data: existing } = await supabase
      .from('deck_cards')
      .select('quantity')
      .eq('deck_id', deckId)
      .eq('card_id', cardId)
      .single();

    if (existing) {
      if (existing.quantity + quantity > 4) {
        throw new Error('Máximo 4 copias por carta');
      }
      await supabase
        .from('deck_cards')
        .update({ quantity: existing.quantity + quantity })
        .eq('deck_id', deckId)
        .eq('card_id', cardId);
    } else {
      await supabase
        .from('deck_cards')
        .insert({ deck_id: deckId, card_id: cardId, quantity });
    }

    await this.validateDeck(deckId);
  },
  async removeCardFromDeck(deckId: string, cardId: string, quantity: number = 1) {
    const { data: existing } = await supabase
      .from('deck_cards')
      .select('quantity')
```

```
.eq('deck_id', deckId)
.eq('card_id', cardId)
.single();

if (!existing) return;

if (existing.quantity <= quantity) {
    await supabase
        .from('deck_cards')
        .delete()
        .eq('deck_id', deckId)
        .eq('card_id', cardId);
} else {
    await supabase
        .from('deck_cards')
        .update({ quantity: existing.quantity - quantity })
        .eq('deck_id', deckId)
        .eq('card_id', cardId);
}

await this.validateDeck(deckId);
} ,

async validateDeck(deckId: string) {
    const { data: deck } = await supabase
        .from('decks')
        .select(
            *,
            leader:cards!leader_card_id(*),
            cards:deck_cards(quantity, card:cards(*))
        )
        .eq('id', deckId)
        .single();

    if (!deck) return false;

    const errors = [];

    // 1. Verificar Leader
    if (!deck.leader_card_id) {
        errors.push('Debe tener un Leader');
    }

    // 2. Contar cartas
    const totalCards = deck.cards.reduce((sum, dc) => sum + dc.quantity, 0);
    if (totalCards !== 49) {
        errors.push(`Debe tener 49 cartas (tiene ${totalCards})`);
    }
}
```

```
}

// 3. Verificar colores (si hay leader)
if (deck.leader) {
  const leaderColors = deck.leader.color.split('/');
  const invalidCards = deck.cards.filter(dc => {
    const cardColors = dc.card.color.split('/');
    return !cardColors.some(c => leaderColors.includes(c) || c === 'Multi');
  });
  if (invalidCards.length > 0) {
    errors.push('Hay cartas con colores incompatibles con el Leader');
  }
}

// 4. Actualizar is_valid
const isValid = errors.length === 0;
await supabase
  .from('decks')
  .update({ is_valid: isValid })
  .eq('id', deckId);

return { isValid, errors },
};

async getDeckById(deckId: string) {
  const { data } = await supabase
    .from('decks')
    .select(
      *,
      leader:cards!leader_card_id(*),
      cards:deck_cards(
        quantity,
        card:cards(*)
      )
    )
    .eq('id', deckId)
    .single();

  return data;
},
};

async getUserDecks() {
  const userId = await supabaseService.getCurrentUserId();
  const { data } = await supabase
    .from('decks')
    .select('id, name, leader:cards!leader_card_id(name, image_url), is_valid')
    .eq('user_id', userId)
```

```
.order('updated_at', { ascending: false });

return data;
}

};
```

Pantallas

DecksListScreen

Lista de mazos del usuario con opciones:

- Ver/Editar
- Duplicar
- Eliminar
- Crear nuevo

DeckBuilderScreen

Pantalla principal de construcción:

- Header con nombre del deck y estadísticas (49/49 cartas)
- Selector de Leader (búsqueda de Leaders)
- Lista de cartas del deck (agrupadas por tipo/cost)
- Botón "Añadir Cartas"
- Validación en tiempo real

CardSelectorScreen

Modal para añadir cartas:

- Búsqueda y filtros
- Muestra colección del usuario
- Indica cuántas copias hay en colección vs deck
- Botón + para añadir

Componentes

typescript

```
// components/deck/
  ├── DeckCard.tsx      // Card para lista de decks
  ├── DeckStats.tsx     // Estadísticas del deck
  ├── LeaderSelector.tsx // Selector de leader
  ├── DeckCardList.tsx   // Lista de cartas en el deck
  └── ValidationErrors.tsx // Mostrar errores de validación
```

Checklist

- Crear tablas decks y deck_cards
 - Implementar deckService completo
 - Crear DecksListScreen
 - Implementar DeckBuilderScreen
 - Crear CardSelectorScreen con filtros
 - Implementar validación de reglas
 - Añadir indicadores visuales de validez
 - Testing exhaustivo de edge cases
 - Optimizar rendimiento de queries
 - Documentar reglas del juego
-

11. Importar deck

Lista: DECKS

Prioridad:  Media

Estimación: 6-8 horas

Objetivo

Permitir importar mazos desde texto (copiar del portapapeles o de URLs).

Formatos Soportados

Formato 1: One Piece Top Decks

```
// Leader
1x OP01-001 Monkey D. Luffy

// Deck
4x OP01-025 Roronoa Zoro
4x OP01-026 Nami
3x OP01-027 Usopp
...
```

Formato 2: Limitless TCG

```
1 OP01-001 Monkey D. Luffy [Leader]  
4 OP01-025 Roronoa Zoro  
4 OP01-026 Nami  
...
```

Formato 3: Simple (nuestra propia app)

json

```
{  
  "name": "Mi Deck",  
  "leader": "OP01-001",  
  "cards": [  
    { "code": "OP01-025", "quantity": 4 },  
    { "code": "OP01-026", "quantity": 4 }  
  ]  
}
```

Implementación

Parser de Decklist

typescript

```
// utils/decklistParser.ts
export interface ParsedDecklist {
  leader?: string;
  cards: Array<{ code: string; quantity: number; name?: string }>;
  format: 'optd' | 'limitless' | 'json' | 'unknown';
}

export function parseDecklistText(text: string): ParsedDecklist {
  // Detectar formato
  if (text.trim().startsWith('{')) {
    return parseJSON(text);
  }

  // Intentar formato OPTD/Limitless
  const lines = text.split('\n').filter(l => l.trim());
  const result: ParsedDecklist = { cards: [], format: 'unknown' };

  for (const line of lines) {
    // Regex para detectar: "4x OP01-025 Nombre" o "4 OP01-025 Nombre"
    const match = line.match(/(\d+)\s*[x\t]?\s+([A-Z0-9-]+)(?:\s+(.+))?/i);

    if (match) {
      const [_, qty, code, name] = match;
      const quantity = parseInt(qty);

      // Detectar si es Leader
      if (quantity === 1 && (line.includes('[Leader]') || line.includes('Leader'))) {
        result.leader = code;
      } else {
        result.cards.push({ code, quantity, name });
      }
    }
  }

  return result;
}

function parseJSON(text: string): ParsedDecklist {
  try {
    const data = JSON.parse(text);
    return {
      leader: data.leader,
      cards: data.cards,
      format: 'json'
    };
  } catch {
  }
}
```

```
    return { cards: [], format: 'unknown' };
}
}
```

Servicio de Importación

typescript

```
// services/deckService.ts (añadir)
async importDeck(name: string, decklistText: string) {
  const parsed = parseDecklistText(decklistText);

  if (parsed.cards.length === 0 && !parsed.leader) {
    throw new Error('No se pudo parsear el decklist');
  }

  // 1. Crear deck vacío
  const deck = await this.createDeck(name);

  // 2. Buscar Leader en BD
  if (parsed.leader) {
    const { data: leaderCard } = await supabase
      .from('cards')
      .select('id')
      .eq('code', parsed.leader)
      .eq('type', 'Leader')
      .single();

    if (leaderCard) {
      await supabase
        .from('decks')
        .update({ leader_card_id: leaderCard.id })
        .eq('id', deck.id);
    }
  }

  // 3. Buscar y añadir cartas
  for (const cardEntry of parsed.cards) {
    const { data: card } = await supabase
      .from('cards')
      .select('id')
      .eq('code', cardEntry.code)
      .limit(1)
      .single();

    if (card) {
      await supabase
        .from('deck_cards')
        .insert({
          deck_id: deck.id,
          card_id: card.id,
          quantity: cardEntry.quantity
        });
    } else {
  
```

```
    console.warn(`Carta no encontrada: ${cardEntry.code}`);
}

}

// 4. Validar deck
const validation = await this.validateDeck(deck.id);

return { deck, validation, missingCards: parsed.cards.filter(c => !c.code) };
}
```

Pantalla de Importación

typescript

```
// screens/ImportDeckScreen.tsx
export const ImportDeckScreen = () => {
  const [decklistText, setDecklistText] = useState("");
  const [deckName, setDeckName] = useState("");
  const [importing, setImporting] = useState(false);

  const handlePaste = async () => {
    const text = await Clipboard.getString();
    setDecklistText(text);
  };

  const handleImport = async () => {
    setImporting(true);
    try {
      const result = await deckService.importDeck(deckName, decklistText);

      if (result.missingCards.length > 0) {
        Alert.alert(
          'Importación Parcial',
          `${result.missingCards.length} cartas no encontradas en la BD`
        );
      }

      navigation.navigate('DeckBuilder', { deckId: result.deck.id });
    } catch (error) {
      Alert.alert('Error', error.message);
    } finally {
      setImporting(false);
    }
  };
}

return (
  <ScrollView>
    <TextInput
      placeholder="Nombre del deck"
      value={deckName}
      onChangeText={setDeckName}
    />

    <TextInput
      placeholder="Pega aquí el decklist..."
      multiline
      numberOfLines={15}
      value={decklistText}
      onChangeText={setDecklistText}
    />
)
```

```
<Button title="Pegar desde portapapeles" onPress={handlePaste} />
<Button title="Importar" onPress={handleImport} disabled={importing} />
<ScrollView>
);
};
```

Testing

Probar con decklists de:

- One Piece Top Decks
- Limitless TCG
- Copiar de nuestra propia app (export)

Checklist

- Implementar parseDecklistText con soporte múltiples formatos
 - Crear método importDeck en deckService
 - Implementar ImportDeckScreen
 - Añadir botón de importar en DecksListScreen
 - Testing con decklists reales de diferentes fuentes
 - Gestionar cartas no encontradas (warning)
 - Documentar formatos soportados
 - Añadir preview antes de confirmar importación
-

12. Publicar deck

Lista: DECKS

Prioridad:  Baja

Estimación: 8-10 horas

Objetivo

Permitir compartir mazos públicamente con la comunidad.

Funcionalidades

1. Hacer público un deck

- Toggle is_public
- Solo decks válidos pueden ser públicos

2. Explorar decks públicos

- Lista de decks públicos

- Filtros: por Leader, por color, por fecha
- Búsqueda por nombre

3. Clonar deck público

- Copiar deck ajeno a tu colección
- Indicar qué cartas te faltan

Modelo de Datos

Actualizar tabla decks

sql

```
ALTER TABLE decks ADD COLUMN IF NOT EXISTS views integer DEFAULT 0;
ALTER TABLE decks ADD COLUMN IF NOT EXISTS clones integer DEFAULT 0;
ALTER TABLE decks ADD COLUMN IF NOT EXISTS likes integer DEFAULT 0;
```

Nueva Tabla: deck_likes (opcional)

sql

```
CREATE TABLE deck_likes (
    user_id uuid REFERENCES profiles(id) ON DELETE CASCADE,
    deck_id uuid REFERENCES decks(id) ON DELETE CASCADE,
    created_at timestamp DEFAULT now(),
    PRIMARY KEY (user_id, deck_id)
);
```

Implementación

Servicio

typescript

```
// services/deckService.ts (añadir)
async publishDeck(deckId: string) {
    // 1. Verificar que es válido
    const { isValid } = await this.validateDeck(deckId);
    if (!isValid) {
        throw new Error('Solo decks válidos pueden ser públicos');
    }

    // 2. Hacer público
    await supabase
        .from('decks')
        .update({ is_public: true })
        .eq('id', deckId);
}

async unpublishDeck(deckId: string) {
    await supabase
        .from('decks')
        .update({ is_public: false })
        .eq('id', deckId);
}

async getPublicDecks(filters?: {
    leaderId?: string;
    color?: string;
    search?: string;
}) {
    let query = supabase
        .from('decks')
        .select(`

            id, name, description, views, clones, likes,
            leader:cards!leader_card_id(name, image_url, color),
            user:profiles!user_id(username)

        `)
        .eq('is_public', true)
        .eq('is_valid', true)
        .order('created_at', { ascending: false });

    if (filters?.leaderId) {
        query = query.eq('leader_card_id', filters.leaderId);
    }

    // ... otros filtros

    const { data } = await query;
    return data;
}
```

```

    },
}

async cloneDeck(deckId: string) {
  const userId = await supabaseService.getCurrentUserId();

  // 1. Obtener deck original
  const originalDeck = await this.getDeckById(deckId);

  // 2. Crear nuevo deck
  const { data: newDeck } = await supabase
    .from('decks')
    .insert({
      user_id: userId,
      name: `${originalDeck.name} (Copia)`,
      description: originalDeck.description,
      leader_card_id: originalDeck.leader_card_id,
      is_public: false
    })
    .select()
    .single();

  // 3. Copiar cartas
  const deckCards = originalDeck.cards.map(dc => ({
    deck_id: newDeck.id,
    card_id: dc.card.id,
    quantity: dc.quantity
  }));
  await supabase.from('deck_cards').insert(deckCards);

  // 4. Incrementar contador de clones
  await supabase.rpc('increment_deck_clones', { deck_id: deckId });

  return newDeck;
},
}

async incrementViews(deckId: string) {
  await supabase.rpc('increment_deck_views', { deck_id: deckId });
}

```

RPC Functions en Supabase

sql

```
CREATE OR REPLACE FUNCTION increment_deck_views(deck_id uuid)
RETURNS void AS $$

BEGIN
    UPDATE decks SET views = views + 1 WHERE id = deck_id;
END;

$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION increment_deck_clones(deck_id uuid)
RETURNS void AS $$

BEGIN
    UPDATE decks SET clones = clones + 1 WHERE id = deck_id;
END;

$$ LANGUAGE plpgsql;
```

Pantallas

```
screens/
└── PublicDecksScreen.tsx (explorar)
    └── PublicDeckDetailScreen.tsx (ver + clonar)
```

✓ Checklist

- Añadir columnas views, clones, likes a decks
- Crear tabla deck_likes
- Implementar métodos publishDeck, getPublicDecks, cloneDeck
- Crear RPC functions en Supabase
- Implementar PublicDecksScreen con filtros
- Crear PublicDeckDetailScreen
- Añadir toggle is_public en DeckBuilderScreen
- Implementar sistema de likes (opcional)
- Testing de permisos y seguridad
- Optimizar queries para rendimiento

13. Exportar deck

Lista: DECKS

Prioridad: ● Baja

Estimación: 4-6 horas

Objetivo

Exportar mazos a diferentes formatos para compartir fuera de la app.

Formatos de Exportación

1. **Texto (Decklist)**
2. **JSON**
3. **Imagen** (screenshot del deck)
4. **URL compatible** (deep link)

Implementación

Formato Texto

```
typescript

// services/deckService.ts (añadir)
async exportDeckText(deckId: string): Promise<string> {
  const deck = await this.getDeckById(deckId);

  let output = `// ${deck.name}\n`;
  if (deck.description) {
    output += `// ${deck.description}\n`;
  }
  output += '\n';

  // Leader
  if (deck.leader) {
    output += `// Leader\n` + `${deck.leader.code} ${deck.leader.name}\n\n`;
  }

  // Cartas agrupadas por tipo
  const groupedCards = _.groupBy(deck.cards, 'card.type');

  for (const [type, cards] of Object.entries(groupedCards)) {
    output += `// ${type}\n`;
    for (const dc of cards) {
      output += `${dc.quantity}x ${dc.card.code} ${dc.card.name}\n`;
    }
    output += '\n';
  }

  return output;
}
```

Formato JSON

```
typescript
```

```
async exportDeckJSON(deckId: string): Promise<string> {
  const deck = await this.getDeckById(deckId);

  const exportData = {
    name: deck.name,
    description: deck.description,
    leader: deck.leader?.code,
    cards: deck.cards.map(dc => ({
      code: dc.card.code,
      quantity: dc.quantity
    })),
    exportedAt: new Date().toISOString(),
    app: 'OPSCANNER'
  };

  return JSON.stringify(exportData, null, 2);
}
```

Compartir

typescript

```
// screens/DeckBuilderScreen.tsx

const handleExport = async () => {
  const options = [
    'Copiar como texto',
    'Exportar JSON',
    'Compartir URL',
    'Cancelar'
  ];

  const buttonIndex = await ActionSheetIOS.showActionSheetWithOptions(
    { options, cancelButtonIndex: 3 },
    (buttonIndex) => {}
  );

  switch(buttonIndex) {
    case 0:
      const text = await deckService.exportDeckText(deckId);
      await Clipboard.setString(text);
      Alert.alert('Copiado', 'Decklist copiado al portapapeles');
      break;

    case 1:
      const json = await deckService.exportDeckJSON(deckId);
      await Share.share({ message: json });
      break;

    case 2:
      const url = `opscanner://deck/${deckId}`;
      await Share.share({ message: url });
      break;
  }
};
```

Exportar como Imagen

typescript

```
// Usando react-native-view-shot
import ViewShot from 'react-native-view-shot';

const deckViewRef = useRef();

const exportAsImage = async () => {
  const uri = await deckViewRef.current.capture();
  await Share.share({ url: uri });
};

<ViewShot ref={deckViewRef} options={{ format: 'png', quality: 0.9 }}>
  <DeckCardList cards={deck.cards} />
</ViewShot>
```

Checklist

- Implementar exportDeckText()
 - Implementar exportDeckJSON()
 - Añadir botón de exportar en DeckBuilderScreen
 - Crear ActionSheet con opciones
 - Implementar share a través de Share API
 - Opcional: Exportar como imagen con view-shot
 - Testing en iOS y Android
 - Documentar formatos de exportación
-