🚀 Cambios Implementados: Sistema de Límites de Anuncios y Modal de Pagos

📅 Fecha de Implementación

13 de Noviembre, 2025

📋 Resumen de Cambios

Se han implementado mejoras significativas en el sistema de anuncios y el proceso de ascenso de rangos, incluyendo:

1. ✅ **Sistema de límites de anuncios por día según rango**
2. ✅ **Contador visible actualizado con anuncios disponibles restantes**
3. ✅ **Mensaje cuando se alcanza el límite diario**
4. ✅ **Modal de pagos con opciones diferenciadas por rango**
5. ✅ **Simulación de pago con MetaMask**
6. ✅ **Otorgamiento automático de rango después del pago**

🔧 Cambios Detallados

1. Sistema de Límites de Anuncios por Día

Archivo: `hooks/useSimulation.ts`

Se corrigieron los límites diarios de anuncios para cada rango:

```
const RANKS: Record<string, UserRank> = {
  registrado: {
    dailyAdsLimit: 5,    // 5 anuncios por día
  },
  invitado: {
    dailyAdsLimit: 10,   // 10 anuncios por día
  },
  basico: {
    dailyAdsLimit: 15,   // ✨ ACTUALIZADO: 15 anuncios por día (antes: 10)
  },
  vip: {
    dailyAdsLimit: 20,   // ✨ ACTUALIZADO: 20 anuncios por día (antes: 10)
  },
  premium: {
    dailyAdsLimit: 30,   // ✨ ACTUALIZADO: 30 anuncios por día (antes: 15)
  },
  elite: {
    dailyAdsLimit: -1,   // ✨ ACTUALIZADO: Ilimitado (antes: 20)
  },
};
```

2. Actualización del Contador de Anuncios Disponibles

Archivo: `components/PublicidadSection.tsx`

Se implementaron mejoras en el contador para mostrar correctamente los anuncios restantes:

Funciones Actualizadas:

```
// Verificar si alcanzó el límite diario
const hasReachedDailyLimit = () => {
  const today = new Date().toISOString().split('T')[0];
  if (!currentRankData) return false;

  // Elite tiene límite ilimitado (-1)
  if (currentRankData.dailyAdsLimit === -1) return false;

  // Verificar si el tracking es del día actual
  if (simulationState.dailyAdTracking.date !== today) {
    return false; // Es un nuevo día, aún no ha alcanzado el límite
  }

  return simulationState.dailyAdTracking.adsViewed >= currentRankData.dailyAdsLimit;
};

// Obtener anuncios restantes del día
const getAdsRemainingToday = () => {
  const today = new Date().toISOString().split('T')[0];
  if (!currentRankData) return 0;

  // Elite tiene límite ilimitado (-1)
  if (currentRankData.dailyAdsLimit === -1) return -1; // Retornar -1 para indicar
ilimitado

  // Verificar si el tracking es del día actual
  if (simulationState.dailyAdTracking.date !== today) {
    return currentRankData.dailyAdsLimit; // Es un nuevo día, todos disponibles
  }

  return Math.max(0, currentRankData.dailyAdsLimit - simulation-
State.dailyAdTracking.adsViewed);
};
```

Interfaz Actualizada:

```
<div className="text-right">
  {currentRankData?.dailyAdsLimit === -1 ? (
    <>
      <span className="text-lg font-bold text-purple-600">
        ∞ Ilimitado
      </span>
      <p className="text-xs text-gray-600">
        Plan {currentRankData?.name}
      </p>
    </>
  ) : (
    <>
      <span className={`text-lg font-bold ${hasReachedDailyLimit() ? 'text-red-600' :
'text-green-600'}`}>
        {getAdsRemainingToday()} / {currentRankData?.dailyAdsLimit || 0}
      </span>
      <p className="text-xs text-gray-600">
        Plan {currentRankData?.name}
      </p>
    </>
  )}
</div>
```

---

3. Mensaje de Límite Diario Alcanzado

Archivo: `components/PublicidadSection.tsx`

Se implementó un mensaje informativo cuando el usuario alcanza su límite diario:

```
{hasReachedDailyLimit() && (
  <div className="mt-3 p-3 rounded-lg bg-red-50 border-red-200 border">
    <div className="flex items-start space-x-2">
      <i className="ri-time-line text-red-600 text-lg mt-0.5"></i>
      <div className="flex-1">
        <p className="font-semibold text-red-800">
          Límite diario alcanzado
        </p>
        <p className="text-sm mt-1 text-red-700">
          Los anuncios se habilitarán después de las 00:00
        </p>
        <div className="flex items-center space-x-1 mt-2">
          <i className="ri-timer-line text-sm"></i>
          <span className="text-sm font-medium text-red-700">
            Tiempo restante: {timeUntilMidnight}
          </span>
        </div>
      </div>
    </div>
  </div>
)}
```

Funcionalidad:
- Muestra un mensaje claro cuando se alcanza el límite
- Indica cuánto tiempo falta para las 00:00 (reseteo diario)
- El reseteo ya estaba implementado previamente en el sistema

4. Modal de Pagos con Opciones Diferenciadas

Archivos Modificados:

- `components/payments/PaymentProcessor.tsx`
- `components/ModalPago.tsx`
- `app/page.tsx`

A. PaymentProcessor.tsx - Nuevos Props

```
interface PaymentProcessorProps {
  // ... props existentes
  showPayPal?: boolean; // ✨ NUEVO: Controlar visibilidad de PayPal
  currentRank?: string; // ✨ NUEVO: Mostrar información del rango
  nextRank?: string; // ✨ NUEVO: Mostrar información del rango siguiente
}
```

B. Lógica de Visualización de Métodos de Pago

Regla de Negocio:

- **Rangos** `registrado` → `invitado` → `basico`**:** Mostrar PayPal y MetaMask
- **Rangos** `basico` → `vip` → `premium` → `elite`**:** Solo MetaMask

```
// En app/page.tsx
const handleUpgrade = () => {
  if (nextRankData) {
    // Determinar si debe mostrar PayPal según el rango actual
    const showPayPalOption = ['registrado', 'invitado'].in-
cludes(simulationState.currentRank);

    // Configurar y abrir el modal de pagos
    setPaymentModalConfig({
      amount: nextRankData.price,
      nextRank: nextRankData.id,
      showPayPal: showPayPalOption,
    });
    setShowPaymentModal(true);
  }
};
```

C. Interfaz del Modal Actualizada

```
{/* CONDICIONAL: Botón PayPal solo si showPayPal es true */}
{showPayPal && (
  <button
    onClick={() => handleSimulatedPayment('paypal')}
    disabled={isLoading}

className="w-full p-4 border rounded-lg transition-all cursor-pointer bg-yellow-50
border-yellow-300 hover:bg-yellow-100"
  >
    <div className="flex items-center justify-center space-x-3">
      <i className="ri-paypal-line text-2xl text-yellow-600"></i>
      <div>
        <p className="font-semibold text-gray-800">Pagar con PayPal</p>
        <p className="text-sm text-gray-600">Pago rápido y seguro</p>
      </div>
      <i className="ri-arrow-right-line text-gray-400"></i>
    </div>
  </button>
)}

{/* Botón MetaMask - Siempre visible */}
<button
  onClick={() => handleSimulatedPayment('metamask')}
  disabled={isLoading}
  className="w-full p-4 border rounded-lg transition-all cursor-pointer bg-orange-50
border-orange-300 hover:bg-orange-100"
>
  <div className="flex items-center justify-center space-x-3">
    <i className="ri-currency-line text-2xl text-orange-600"></i>
    <div>
      <p className="font-semibold text-gray-800">Pagar con MetaMask</p>
      <p className="text-sm text-gray-600">Pago con criptomonedas</p>
    </div>
    <i className="ri-arrow-right-line text-gray-400"></i>
  </div>
</button>
```

---

5. Simulación de Pago con MetaMask

Archivo: `components/payments/PaymentProcessor.tsx`

La función `handleSimulatedPayment` ya existía y maneja correctamente los pagos simulados con MetaMask:

```
const handleSimulatedPayment = async (method: 'paypal' | 'metamask') => {
  setIsLoading(true);

  try {
    // Simular tiempo de procesamiento
    await new Promise((resolve) => setTimeout(resolve, 1500));

    const paymentDetails: PaymentDetails = {
      amount,
      orderID: method === 'paypal' ? `PAYPAL_${paymentType.toUpperCase()}_$
{Date.now()}` : undefined,
      transactionHash: method === 'metamask' ? `0x$
{Math.random().toString(16).substr(2, 64)}` : undefined,
      userData: {
        userId,
        paymentType,
        method,
        timestamp: new Date().toISOString(),
      },
      successMessage: `¡Pago exitoso por $${amount} USD!`,
    };

    // Guardar registro del pago
    await saveGeneralPaymentRecord(paymentDetails, method);
    onSuccess(paymentDetails);
  } catch (error) {
    onError({
      message: 'Error en el pago simulado',
      code: 'SIMULATION_ERROR',
      paymentType,
    });
  } finally {
    setIsLoading(false);
  }
};
```

Características:
- ✅ Genera hash de transacción simulado para MetaMask
- ✅ Simula tiempo de procesamiento (1.5 segundos)
- ✅ Guarda registro del pago en localStorage
- ✅ Maneja errores correctamente

---

6. Otorgamiento Automático de Rango Después del Pago

Archivo: `app/page.tsx`

Se implementó el handler `handlePaymentSuccess` que:
1. Procesa el pago exitoso
2. Asciende al usuario al siguiente rango
3. Actualiza las pestañas disponibles
4. Resetea el timer si es necesario
5. Muestra modal de confirmación

```
const handlePaymentSuccess = (details: any) => {
  if (nextRankData) {
    const success = upgradeToRank(nextRankData.id);
    if (success) {
      // Actualizar datos del modal de felicitación
      setUpgradeModalData({
        rank: nextRankData.id,
        name: nextRankData.name,
      });

      // Mostrar modal de felicitación
      setShowUpgradeModal(true);

      // Animaciones
      setTimeout(() => {
        const modalElement = document.querySelector('.upgrade-modal-animated');
        if (modalElement) {
          modalElement.classList.remove('animate-bounce');
          modalElement.classList.add('animate-diminishing-bounce');
        }
      }, 100);

      // Actualizar pestañas disponibles
      const previousTabs = tabs.map((t) => t.id);
      setTimeout(() => {
        const newAvailableTabs = getAvailableTabsForRank(nextRankData.id);
        const newTabs = newAvailableTabs.filter(
          (tabId) =>
            !previousTabs.includes(tabId) &&
            ['balance', 'niveles', 'beneficios', 'panel', 'publicidad'].in-
cludes(tabId)
        );

        if (newTabs.length > 0) {
          const updatedHighlighted = [...highlightedTabs, ...newTabs];
          setHighlightedTabs(updatedHighlighted);
          localStorage.setItem('highlighted_tabs', JSON.stringi-
fy(updatedHighlighted));
        }
      }, 100);

      // Resetear timer si es necesario
      if (nextRankData.timerDuration !== -1) {
        timerState.resetTimer();
      }
    }
  }

  // Cerrar el modal de pagos
  setShowPaymentModal(false);
};
```

💰 Precios por Rango

Rango	Precio (USD)	Límite Diario de Anuncios	Métodos de Pago
Registrado	$3	5	-
Invitado	$5	10	PayPal, MetaMask
Básico	$10	15	PayPal, MetaMask
VIP	$50	20	MetaMask
Premium	$400	30	MetaMask
Elite	$6,000	∞ (Ilimitado)	MetaMask

🔄 Flujo de Usuario Completo

Escenario 1: Usuario Registrado → Invitado

1. Usuario hace clic en botón “Ascender a Invitado - $5 USD”
2. Se abre el modal de pagos mostrando:
   - ✅ Opción PayPal
   - ✅ Opción MetaMask
3. Usuario selecciona método de pago
4. Se simula el pago (1.5 segundos)
5. Se asciende automáticamente a Invitado
6. Se muestra modal de felicitación
7. Límite de anuncios aumenta de 5 a 10 por día

Escenario 2: Usuario VIP → Premium

1. Usuario hace clic en botón “Ascender a Premium - $400 USD”
2. Se abre el modal de pagos mostrando:
   - ❌ PayPal (no disponible)
   - ✅ Opción MetaMask (única opción)
3. Usuario selecciona MetaMask
4. Se simula el pago (1.5 segundos)
5. Se asciende automáticamente a Premium
6. Se muestra modal de felicitación
7. Límite de anuncios aumenta de 20 a 30 por día

Escenario 3: Usuario alcanza límite diario

1. Usuario ha visto 10 anuncios (límite para Invitado)
2. Contador muestra: **0 / 10** en rojo
3. Se muestra mensaje: “Límite diario alcanzado”

4. Se indica tiempo restante hasta las 00:00
5. Botones de “Ver Anuncio” se deshabilitan con mensaje “Límite diario alcanzado”
6. A las 00:00, el contador se resetea automáticamente

📁 Archivos Modificados

1. Hooks

- ✅ `hooks/useSimulation.ts` - Límites de anuncios actualizados

2. Componentes

- ✅ `components/PublicidadSection.tsx` - Contador y mensajes de límite
- ✅ `components/payments/PaymentProcessor.tsx` - Lógica de métodos de pago
- ✅ `components/ModalPago.tsx` - Props para control de pagos

3. Páginas

- ✅ `app/page.tsx` - Integración del modal de pagos

🧪 Testing y Validación

Pruebas Realizadas

✅ **Límites de Anuncios:**
- Verificación de límites correctos para cada rango
- Reseteo automático a medianoche
- Contador de anuncios restantes
- Manejo de límite ilimitado para Elite

✅ **Modal de Pagos:**
- Visualización correcta según rango
- PayPal visible solo para registrado, invitado, basico
- MetaMask visible para todos los rangos
- Simulación de pagos funcional

✅ **Ascenso de Rango:**
- Ascenso automático después del pago
- Actualización de límites
- Actualización de pestañas
- Modal de felicitación

🚀 Próximos Pasos Sugeridos

1. **Integración Real de PayPal**
   - Implementar PayPal SDK completo
   - Configurar credenciales de producción

2. **Integración Real de MetaMask**
   - Implementar Web3 para transacciones reales
   - Configurar smart contracts

3. **Persistencia en Backend**
   - Sincronizar límites diarios con API
   - Guardar historial de pagos en base de datos

4. **Notificaciones**
   - Alertas cuando quedan pocos anuncios disponibles
   - Notificaciones de reseteo diario

5. **Analytics**
   - Tracking de conversión de pagos
   - Métricas de uso de anuncios por rango

📝 Notas Importantes

⚠️ Sistema de Reseteo Diario:
El reseteo automático a las 00:00 ya estaba implementado en el sistema y NO fue modificado. Los cambios solo afectan:
- Límites específicos por rango
- Visualización del contador
- Mensajes de límite alcanzado

⚠️ Simulación de Pagos:
Los pagos actuales son simulados. Para producción, se debe:
- Implementar integraciones reales con PayPal y MetaMask
- Validar transacciones en el backend
- Implementar webhooks para confirmaciones

⚠️ Almacenamiento Local:
Los datos se guardan en localStorage. Para producción:
- Migrar a base de datos del servidor
- Implementar API endpoints
- Añadir autenticación y autorización

👨‍💻 Desarrollador

Implementación realizada por: DeepAgent AI
Fecha: 13 de Noviembre, 2025
Versión: 1.0.0

📞 Soporte

Para preguntas o problemas relacionados con esta implementación, por favor consultar:
- Documentación del proyecto

- Sistema de issues en el repositorio
- Equipo de desarrollo

¡Implementación Completa y Lista para Uso! ✨