

# Cambios Realizados: Sistema de Temporizador con Segundos

---

**Fecha:** 25 de noviembre de 2025

## Resumen

---

Se ha modificado completamente el sistema de temporizador de actividad del proyecto YigiCoin para trabajar internamente con **segundos** en lugar de horas/minutos, mientras que la interfaz de usuario sigue mostrando “horas” para mantener la experiencia del usuario consistente.

---

## 1. Configuración Centralizada de Tiempos por Rango

---

**Archivo:** `lib/economyConfig.ts`

### Cambios realizados:

- ✓ Agregada nueva configuración `counterSeconds` con valores en segundos para cada rango:
    - `registrado` : 168 segundos
    - `invitado` : 72 segundos
    - `miembro` : 84 segundos
    - `vip` : 96 segundos
    - `premium` : 120 segundos
    - `elite` : 168 segundos
  - ✓ Agregada configuración del botón refrescar:

```
typescript
refreshButton: {
  cost: 40,           // Costo en puntos
  timeAddedSeconds: 48, // +48 segundos al refrescar
}
```
  - ✓ Creada nueva función `counterSecondsForRank(rank)` que retorna la duración del contador en segundos
  - ✓ Creada función helper `formatSecondsAsHours(seconds)` para mostrar segundos como “horas” en la UI
  - ✓ Mantenida compatibilidad con `counterMsForRank()` (deprecated)
- 

## 2. Modificación de la Lógica del Temporizador

---

**Archivo:** `app/actions/counter.ts`

### Cambios realizados:

- ✓ Actualizada función `refreshCounter()` con nueva lógica:

- Ya NO reinicia el contador a tiempo completo
- Ahora **suma +48 segundos** al tiempo actual
- Respeta el **techo máximo** del rango (no puede superarlo)
- Calcula: `nuevoTiempo = min(tiempoActual + 48, techoDelRango)`

```
// Código anterior (OBSOLETO):
const resetMs = counterMsForRank(user.rank)
const counterExpiresAt = new Date(Date.now() + resetMs)

// Código nuevo:
const currentRemainingSeconds = Math.floor(currentRemainingMs / 1000)
const ceilingSeconds = counterMsForRank(user.rank as any) / 1000
const newTimeSeconds = Math.min(currentRemainingSeconds + timeToAddSeconds, ceilingSeconds)
```

### 3. Nuevo Botón “Refrescar Temporizador”

**Archivo:** `components/RefreshCounterButton.tsx`

**Cambios realizados:**

- ☒ Implementado modal de confirmación antes de ejecutar el refresh
- ☒ Actualizada lógica para sumar +48 segundos sin superar el techo
- ☒ Implementadas validaciones:
  - Botón deshabilitado si `puntos < 40`
  - Botón deshabilitado si `tiempoActual >= techoDelRango`
- ☒ Tooltip dinámico que explica por qué el botón está deshabilitado
- ☒ Integración con localStorage para modo demo

### 4. Modal de Confirmación del Botón Refrescar

**Archivo:** `components/modals/RefreshConfirmModal.tsx` (NUEVO)

**Características:**





- ☒ Muestra toda la información antes de confirmar:
  - Puntos actuales
  - Costo (40 puntos)
  - Puntos después del pago
  - Tiempo actual restante
  - Tiempo que se agregará (+48 horas visualizado)
  - Nuevo tiempo estimado
  - Techo máximo del rango
  - Explicación de que no puede superar el techo
- ☒ Botones “Confirmar” y “Cancelar”
- ☒ Estado de carga mientras procesa

-  Diseño responsive y amigable

## 5. Modificación del Botón “Extender Tiempo”





Archivo: `app/page.tsx`

Cambios realizados:

-  **Eliminada** completamente la opción de pago con puntos (100 puntos)
-  Solo quedan opciones de pago con USD:
- **\$2 USD** = 48 horas (172,800 segundos)
- **\$5 USD** = 100 horas (360,000 segundos)
-  Estas opciones **SÍ pueden superar** el techo del rango
-  Actualizada función `handleTimeExtensionPayment()` para usar segundos correctos

## 6. Validaciones Implementadas

Validaciones del Botón Refrescar:

1.  **Puntos insuficientes:** Deshabilitado si `puntos < 40`
2.  **Tiempo en/sobre el techo:** Deshabilitado si `tiempoActual >= techoDelRango`
3.  **Durante procesamiento:** Deshabilitado mientras está en estado de carga
4.  **Tooltip informativo:** Muestra la razón por la cual está deshabilitado



Validaciones en Backend:

```
// En app/actions/counter.ts
if (user.points < cost) throw new Error('Puntos insuficientes')
if (currentRemainingSeconds >= ceilingSeconds) {
  throw new Error('Ya estás en el tiempo máximo de tu rango')
}
```

## 7. Sincronización de Tiempos en Todos los Componentes

Archivo: `hooks/useSimulation.ts`






Cambios realizados:

-  Importada función `counterSecondsForRank` de `economyConfig`
-  Actualizado objeto `RANKS` para usar la configuración centralizada:

```
registrado: {
  timerDuration: counterSecondsForRank('registrado'), // 168 segundos
}
invitado: {
  timerDuration: counterSecondsForRank('invitado'), // 72 segundos
}
// ... etc para todos los rangos
```





## 8. UI: Mostrando “Horas” aunque Internamente sean Segundos

### Estrategia implementada:

1.  **Internamente:** Todo se maneja en segundos
2.  **En la UI:** Los textos siguen diciendo “horas”
3.  **Función helper:** `formatSecondsAsHours(seconds)` en `economyConfig.ts`
4.  **Modal de confirmación:** Usa esta función para mostrar “48 horas” aunque sean 48 segundos
5.  **Modal de extender tiempo:** Muestra “48 horas (172,800 segundos)” para claridad en pruebas







## 9. Verificación de Compilación

### Resultados:




-  **TypeScript:** Sin errores ( `npx tsc --noEmit` )
-  **Next.js Build:** Compilado exitosamente ( `npm run build` )
-  **Todas las páginas:** Generadas correctamente
-  **Tipos:** Verificados y consistentes

## 10. Archivos Modificados

### Archivos principales modificados:

1.  `lib/economyConfig.ts` - Configuración centralizada
2.  `app/actions/counter.ts` - Lógica de backend del temporizador
3.  `components/RefreshCounterButton.tsx` - Botón refrescar actualizado
4.  `components/modals/RefreshConfirmModal.tsx` - **NUEVO** modal de confirmación
5.  `app/page.tsx` - Modal de extender tiempo modificado
6.  `hooks/useSimulation.ts` - Sincronización de tiempos por rango

### Archivos NO modificados (mantienen compatibilidad):

-  `components/ContadorUsuario.tsx` - Sin cambios necesarios
-  `hooks/useTimer.ts` - Sin cambios necesarios
-  `app/api/refresh/route.ts` - Sin cambios necesarios

## 11. Comportamiento Final del Sistema

### Botón “Refrescar Temporizador” (40 puntos):

Condiciones:

- Costo: 40 puntos
- Acción: Suma +48 segundos al tiempo actual
- Límite: NO puede superar el techo del rango
- Habilitado si: puntos  $\geq$  40 Y tiempoActual  $<$  techoDelRango

### Botón “Extender Tiempo” (USD):

Opciones:

- \$2 USD = +172,800 segundos (48 horas)
- \$5 USD = +360,000 segundos (100 horas)
- Sí puede superar el techo del rango
- Solo pagos en USD (PayPal/MetaMask)

### Regla Clave:

Mientras el tiempo esté por encima del techo del rango, el botón de refrescar queda **deshabilitado**.

## 12. Ejemplos de Funcionamiento

### Ejemplo 1: Usuario Invitado (Techo: 72 segundos)

Tiempo actual: 50 segundos  
 Acción: Click en "Refrescar"  
 Cálculo:  $\min(50 + 48, 72) = 72$  segundos  
 Resultado: Tiempo nuevo = 72 segundos (alcanzó el techo)

### Ejemplo 2: Usuario VIP (Techo: 96 segundos)

Tiempo actual: 80 segundos  
 Acción: Click en "Refrescar"  
 Cálculo:  $\min(80 + 48, 96) = 96$  segundos  
 Resultado: Tiempo nuevo = 96 segundos (alcanzó el techo)

### Ejemplo 3: Usuario Premium (Techo: 120 segundos)

Tiempo actual: 50 segundos  
 Acción: Click en "Refrescar"  
 Cálculo:  $\min(50 + 48, 120) = 98$  segundos  
 Resultado: Tiempo nuevo = 98 segundos (NO alcanzó el techo)

## Ejemplo 4: Usuario con tiempo en el techo

Tiempo actual: 96 segundos (usuario VIP con techo de 96)  
 Acción: Click en "Refrescar"  
 Resultado: Botón deshabilitado - "Ya estás en el tiempo máximo de tu rango"

## 13. Testing y Validación

### Para probar el sistema:

1. **Registrar usuario** con rango específico
2. **Observar temporizador** decrementando en segundos
3. **Probar botón refrescar:**
  - Con puntos suficientes ( $\geq 40$ )
  - Con puntos insuficientes ( $< 40$ )
  - Con tiempo bajo el techo
  - Con tiempo en/sobre el techo
4. **Verificar modal de confirmación:**
  - Muestra información correcta
  - Cálculos de tiempo son precisos
  - Respetar el techo del rango
5. **Probar extender tiempo:**
  - Solo aparecen opciones USD
  - Puede superar el techo del rango

## 14. Fuente Única de Verdad

### Archivo Central: `lib/economyConfig.ts`

Todos los componentes y acciones del servidor ahora consultan este archivo para obtener:

- ☒ Duración del contador por rango (segundos)
- ☒ Costo del refresh (40 puntos)
- ☒ Tiempo agregado por refresh (48 segundos)
- ☒ Funciones helper para formateo

**No hay valores "hardcoded"** en otros archivos.

## 15. Conclusión

- ☒ Todos los cambios implementados exitosamente
- ☒ Compilación sin errores
- ☒ Configuración centralizada funcional
- ☒ UI muestra "horas" mientras internamente usa segundos
- ☒ Validaciones completas implementadas

✓ **Modal de confirmación implementado**

✓ **Sistema listo para pruebas**

---

## Notas Importantes

---

1. **Para producción:** Cambiar los valores de `counterSeconds` en `economyConfig.ts` de segundos a los valores reales en segundos (por ejemplo, 7 días = 604,800 segundos)
  2. **Valores actuales (para pruebas):**
    - Son segundos que se mostrarán como “horas” en la UI
    - Facilitan las pruebas sin esperar días/horas reales
  3. **Extensibilidad:**
    - Agregar nuevos rangos solo requiere actualizar `economyConfig.ts`
    - El resto del sistema se adapta automáticamente
- 

**Implementado por:** DeepAgent (Abacus.AI)

**Fecha:** 25 de noviembre de 2025