Guía de Mejores Prácticas para Prevenir Hydration Errors en INMOVA

¿Qué son los Hydration Errors?

Los **hydration errors** ocurren cuando el HTML generado en el servidor (SSR) no coincide con el HTML generado en el cliente durante la primera renderización. Next.js compara el contenido y si encuentra diferencias, muestra un warning.

Causas Comunes

1. ❌ Uso de `new Date()` en el Estado Inicial

```
// MAL - Causa hydration error
function MyComponent() {
  const [selectedDate, setSelectedDate] = useState(new Date());
  return <div>{selectedDate.toLocaleDateString()}</div>;
}
```

Por qué falla: El servidor genera una fecha, el cliente genera otra fecha diferente.

Solución:

```
// BIEN - Usa null como inicial y actualiza en useEffect
import { useSafeDateState } from '@/lib/ssr-safe-date';

function MyComponent() {
  const [selectedDate, setSelectedDate] = useSafeDateState();

  if (!selectedDate) {
    return <div>Cargando...</div>;
  }

  return <div>{selectedDate.toLocaleDateString()}</div>;
}
```

2. ❌ Uso de `Math.random()` o `Date.now()` en Render

```
// MAL - Causa hydration error
function MyComponent() {
  const id = `item-${Math.random()}`;
  return <div id={id}>Contenido</div>;
}
```

Solución:

```
// BIEN - Genera ID en useEffect
import { generateSafeId } from '@/lib/ssr-safe-date';

function MyComponent() {
  const [id, setId] = useState<string>('');

  useEffect(() => {
    setId(generateSafeId('item'));
  }, []);

  return <div id={id}>Contenido</div>;
}
```

3. ❌ Acceso a APIs del Navegador Durante SSR

```
// MAL - Causa error en SSR
function MyComponent() {
  const width = window.innerWidth; // window no existe en servidor
  return <div>Width: {width}</div>;
}
```

Solución:

```
// BIEN - Solo accede a window en el cliente
import { ClientOnly } from '@/lib/ssr-safe-date';

function MyComponent() {
  const [width, setWidth] = useState(0);

  useEffect(() => {
    setWidth(window.innerWidth);
  }, []);

  return (
    <ClientOnly fallback={<div>Cargando...</div>}>
      <div>Width: {width}</div>
    </ClientOnly>
  );
}
```

4. ❌ Uso de `localStorage` o `sessionStorage` Durante SSR

```
// MAL
function MyComponent() {
  const theme = localStorage.getItem('theme') || 'light';
  return <div className={theme}>Contenido</div>;
}
```

Solución:

```tsx
// BIEN
function MyComponent() {
  const [theme, setTheme] = useState<string>('light');

  useEffect(() => {
    const savedTheme = localStorage.getItem('theme');
    if (savedTheme) {
      setTheme(savedTheme);
    }
  }, []);

  return <div className={theme}>Contenido</div>;
}
```

5. ❌ Formateo de Fechas Inconsistente

```tsx
// MAL - puede dar resultados diferentes según timezone del servidor/cliente
function MyComponent({ date }: { date: Date }) {
  return <div>{date.toLocaleString()}</div>;
}
```

Solución:

```tsx
// BIEN - usa date-fns para formateo consistente
import { format } from 'date-fns';
import { useSafeFormattedDate } from '@/lib/ssr-safe-date';

function MyComponent({ date }: { date: Date }) {
  const formatted = useSafeFormattedDate(date, 'yyyy-MM-dd HH:mm', '--');
  return <div>{formatted}</div>;
}
```

6. ❌ Componentes de Terceros que Usan APIs del Navegador

```tsx
// MAL - si ThirdPartyComponent usa window/document
import ThirdPartyComponent from 'third-party-lib';

function MyComponent() {
  return <ThirdPartyComponent />;
}
```

Solución:

```tsx
// BIEN - usa dynamic import con ssr: false
import dynamic from 'next/dynamic';

const ThirdPartyComponent = dynamic(
  () => import('third-party-lib'),
  { ssr: false, loading: () => <div>Cargando...</div> }
);

function MyComponent() {
  return <ThirdPartyComponent />;
}
```

Helpers Disponibles en INMOVA

El proyecto incluye helpers en `@/lib/ssr-safe-date` para casos comunes:

`useClientDate()`

Retorna la fecha actual, pero solo en el cliente:

```
const currentDate = useClientDate();
if (!currentDate) return <LoadingSpinner />;
return <div>{currentDate.toLocaleDateString()}</div>;
```

`useSafeDateState()`

Hook para estado de fecha seguro:

```
const [selectedDate, setSelectedDate] = useSafeDateState();
// selectedDate será null inicialmente, luego Date actual
```

`useSafeFormattedDate(date, format, placeholder)`

Formatea fechas de manera segura:

```
const formatted = useSafeFormattedDate(myDate, 'yyyy-MM-dd', '--');
// Muestra '--' durante SSR, fecha formateada en cliente
```

`ClientOnly`

Componente wrapper para renderizar solo en cliente:

```
<ClientOnly fallback={<Skeleton />}>
  <ComponentThatUsesWindow />
</ClientOnly>
```

`generateSafeId(prefix)`

Genera IDs únicos de manera segura:

```
const id = generateSafeId('modal'); // Seguro para SSR
```

Checklist de Verificación

Antes de hacer commit, verifica:

- [ ] ¿Usas `new Date()` en el estado inicial? → Usa `useSafeDateState()`
- [ ] ¿Usas `Math.random()` o `Date.now()` en render? → Muévelo a `useEffect`
- [ ] ¿Accedes a `window`, `document`, `localStorage`? → Usa `useEffect` o `ClientOnly`
- [ ] ¿Formateas fechas con `toLocaleString()`? → Usa `useSafeFormattedDate()`
- [ ] ¿Importas componentes que usan APIs del navegador? → Usa `dynamic` con `ssr: false`
- [ ] ¿El componente se ve diferente en la primera renderización? → Investiga la causa

Testing de Hydration

1. Verificar en Desarrollo

```
yarn dev
```

Abre DevTools y busca warnings de hydration en la consola.

2. Verificar en Build de Producción

```
yarn build
yarn start
```

Los hydration errors pueden aparecer solo en producción.

3. Usar React DevTools

Instala React DevTools y habilita “Highlight updates” para ver re-renders.

Debugging

Si encuentras un hydration error:

1. **Lee el mensaje completo** - Next.js indica qué no coincide
2. **Busca el componente** - El stack trace muestra dónde ocurre
3. **Verifica el estado inicial** - ¿Depende de valores dinámicos?
4. **Revisa useEffect** - ¿Estás mutando estado durante el montaje?
5. **Compara SSR vs CSR** - ¿El HTML es idéntico?

Recursos

- Next.js - Understanding Hydration (https://nextjs.org/docs/messages/react-hydration-error)
- React - Hydration (https://react.dev/reference/react-dom/client/hydrateRoot)
- Documentación interna (./lib/ssr-safe-date.ts)

Supresión de Warnings (Solo Producción)

Después de corregir todos los errores reales, puedes suprimir warnings cosméticos:

```
// En app/layout.tsx
import { HydrationErrorSuppressor } from '@/components/HydrationErrorSupressor';

export default function RootLayout({ children }) {
  return (
    <html suppressHydrationWarning>
      <body>
        <HydrationErrorSuppressor />
        {children}
      </body>
    </html>
  );
}
```

⚠️ **IMPORTANTE:** Solo usa esto en producción y después de verificar que no hay errores reales.