Infinite Loop Fix - November 25, 2025

Problem Description

The Route selection tool was entering an **infinite render loop** when selecting specific technician/territory combinations (e.g., “David Bontrager” with “APS Chandler”). The error manifested as:

```
Error: Maximum update depth exceeded. This can happen when a component
repeatedly calls setState inside componentWillUpdate or componentDidUpdate.
React limits the number of nested updates to prevent infinite loops.
```

Console Logs Pattern

The console showed the same logs repeating endlessly:

```
🚗 Selected technician: David Bontrager, 6 stops
📍 Map centering to: 33.3652, -111.9440
🗺️ Rendering polygons for 3 unique ZIP codes
✅ Generated 4 polygon objects
🚗 Selected technician: David Bontrager, 6 stops
📍 Map centering to: 33.3652, -111.9440
🗺️ Rendering polygons for 3 unique ZIP codes
✅ Generated 4 polygon objects
... (repeating infinitely)
```

Root Cause Analysis

The infinite loop was caused by **unstable dependencies in React hooks**:

The Problem Code

```
// ❌ PROBLEMATIC: filteredRoutes recalculated every render
const filteredRoutes = routes.filter((route) => {
  // ... filtering logic
});

// ❌ This useEffect has filteredRoutes as a dependency
useEffect(() => {
  if (selectedTechnician && filteredRoutes.length > 0) {
    const avgLat = filteredRoutes.reduce((sum, r) => sum + r.latitude, 0) / filtered-
Routes.length;
    const avgLng = filteredRoutes.reduce((sum, r) => sum + r.longitude, 0) / filtered-
Routes.length;

    setMapCenter({ lat: avgLat, lng: avgLng });  // ⚠️ State update
    setMapZoom(12);                               // ⚠️ State update
  }
}, [selectedTechnician, filteredRoutes]);  // ❌ Unstable dependency!

// ❌ territoryPolygons also recalculated every render
const territoryPolygons = getTerritoryPolygons();
```

The Infinite Loop Cycle

1. **Initial Render**: Component renders with selected technician
2. **Filter Calculation**: `filteredRoutes` is calculated as a new array object
3. **useEffect Trigger**: React detects `filteredRoutes` changed (new object reference)
4. **State Update**: `useEffect` calls `setMapCenter()` and `setMapZoom()`
5. **Re-render**: State change triggers component re-render
6. **Back to Step 2**: New `filteredRoutes` array created → infinite loop

Why This Happens

Even if `filteredRoutes` contains the same data, JavaScript array/object comparisons use **reference equality**:

```
const array1 = [1, 2, 3];
const array2 = [1, 2, 3];

array1 === array2  // false! Different objects in memory
```

React sees each new `filteredRoutes` array as "different" and triggers the `useEffect`.

Solution Implemented

Fix 1: Memoize `filteredRoutes`

Before:

```
const filteredRoutes = routes.filter((route) => {
  // ... filtering logic
});
```

After:

```
const filteredRoutes = useMemo(() => {
  return routes.filter((route) => {
    // ... filtering logic
  });
}, [routes, selectedTechnician, areaFilter, selectedDay]);
```

Impact: `filteredRoutes` is now only recalculated when its actual dependencies change, not on every render.

Fix 2: Memoize `territoryPolygons`

Before:

```
const getTerritoryPolygons = () => {
  // ... polygon generation logic
};

const territoryPolygons = getTerritoryPolygons();
```

After:

```
const territoryPolygons = useMemo(() => {
  // ... polygon generation logic
}, [boundaries, selectedTechnician, filteredRoutes]);
```

Impact: Polygon generation now only runs when boundaries, technician, or routes actually change.

Fix 3: Memoize `stats`

Before:

```
const stats = {
  totalAccounts: filteredRoutes.length,
  territories: Array.from(new Set(filteredRoutes.map((r) => r.territory))),
  zips: Array.from(new Set(filteredRoutes.map((r) => r.zipCode))),
};
```

After:

```
const stats = useMemo(() => ({
  totalAccounts: filteredRoutes.length,
  territories: Array.from(new Set(filteredRoutes.map((r) => r.territory))),
  zips: Array.from(new Set(filteredRoutes.map((r) => r.zipCode))),
}), [filteredRoutes]);
```

Impact: Stats calculation is cached and only runs when `filteredRoutes` changes.

Technical Details

React `useMemo` Hook

`useMemo` caches the result of a calculation and only recalculates when dependencies change:

```
const memoizedValue = useMemo(() => {
  return expensiveCalculation(a, b);
}, [a, b]);  // Only recalculates when a or b changes
```

Dependency Array Impact

filteredRoutes Dependencies:
- `routes` : Source data
- `selectedTechnician` : Filter by technician
- `areaFilter` : Filter by territory
- `selectedDay` : Filter by service day

Only when one of these **actually changes** will `filteredRoutes` be recalculated.

territoryPolygons Dependencies:
- `boundaries` : ZIP boundary geometry data
- `selectedTechnician` : Determines which ZIPs to render
- `filteredRoutes` : Source of ZIP codes to process

Files Modified

- `/home/ubuntu/phoenix_territory_map/nextjs_space/components/routes-map-view.tsx`

Key Changes:

1. **Line 111-140**: Wrapped `filteredRoutes` calculation in `useMemo`
2. **Line 185-243**: Wrapped `territoryPolygons` calculation in `useMemo`
3. **Line 246-250**: Wrapped `stats` calculation in `useMemo`
4. **Line 161-182**: Moved `convertGeometryToPaths` outside memoization for reusability

Testing Results

Before Fix:

- ❌ Selecting “David Bontrager” + “APS Chandler” → Infinite loop
- ❌ Console logs repeating endlessly
- ❌ App becomes unresponsive
- ❌ React error: “Maximum update depth exceeded”

After Fix:

- ✅ Selecting “David Bontrager” + “APS Chandler” → Works perfectly
- ✅ Console logs appear once per selection
- ✅ App remains responsive
- ✅ No React errors
- ✅ All technician/territory combinations work

Build Status:

```
✓ Compiled successfully
✓ Generating static pages (5/5)
Route (app)                              Size     First Load JS
┌ ƒ /                                    80.3 kB         167 kB
```

Performance Impact

Before:

- Recalculating on every render (~60 times/second)
- Creating new arrays/objects constantly
- Triggering useEffect unnecessarily
- High CPU usage during infinite loop

After:

- Calculations only when dependencies change
- Stable object references
- useEffect triggers only when needed
- Minimal CPU usage, stable performance

Prevention Measures

Best Practices Implemented:

1. ✅ **Memoize expensive calculations**: Use `useMemo` for filter/map/reduce operations
2. ✅ **Stable dependencies**: Ensure useEffect dependencies are memoized
3. ✅ **Avoid inline objects**: Don't create new objects in render (use useMemo)
4. ✅ **Track dependency changes**: Log when memoized values recalculate during development

Code Pattern to Follow:

```
// ✅ GOOD: Memoized calculation
const filteredData = useMemo(() => {
  return data.filter(item => item.active);
}, [data]);  // Only recalculates when data changes

// ❌ BAD: Recalculated every render
const filteredData = data.filter(item => item.active);
```

Related Issues Fixed

This fix also resolved:
- ✅ Unnecessary polygon regeneration on every render
- ✅ Redundant stats calculations
- ✅ Excessive console logging
- ✅ High memory usage from object creation

Deployment

- **Status**: ✅ Successfully deployed
- **URL**: https://phoenixnewlocations.abacusai.app
- **Date**: November 25, 2025
- **Testing**: Verified with all technician/territory combinations

User Impact

Before: Certain technician selections caused the app to freeze and crash.

After: All technician selections work smoothly with instant map updates and accurate route visualization.

Technical Notes

Why useMemo vs useCallback?

- **useMemo**: Memoizes the **result** of a calculation
- **useCallback**: Memoizes the **function itself**

We used `useMemo` because we needed to cache the calculated arrays/objects, not the functions.

Dependency Array Guidelines

1. Include all values used inside the memoized calculation

2. Ensure dependencies are stable (memoized or primitive values)
3. Avoid including unstable objects/arrays as dependencies
4. Use ESLint `react-hooks/exhaustive-deps` rule to catch missing dependencies

React DevTools Profiler

The React DevTools Profiler can help identify infinite loops:
- Look for components rendering 50+ times in quick succession
- Check “Why did this render?” to see which props/state changed
- Use “Record why each component rendered” option

Contact

If you encounter similar infinite loop issues:

1. Check the browser console for repeating logs
2. Look for React error messages about “maximum update depth”
3. Review useEffect dependencies for unstable references
4. Consider memoizing expensive calculations with useMemo
5. Use React DevTools Profiler to identify the problematic component

---

Summary

The infinite loop was caused by **unstable object references** in useEffect dependencies. By using `useMemo` to stabilize the `filteredRoutes`, `territoryPolygons`, and `stats` calculations, we ensured that React only recalculates and re-renders when the actual data changes, not just when new object references are created.

Key Takeaway: Always memoize calculations that are used as dependencies in React hooks to prevent infinite render loops.