Error Reporting Fix - Complete Summary

Date: December 10, 2025
Commit: c37addb
Status: ✅ DEPLOYED TO PRODUCTION

Problem Identified

The caregivers API was returning only `{"error":"Internal server error"}` without any details about what went wrong. This made debugging impossible.

Root Cause

The issue was in `/src/lib/auth-utils.ts` - the `handleAuthError()` function:

```
// OLD CODE - returned generic error
export function handleAuthError(error: unknown) {
  // ... auth checks ...

  // Unknown error
  console.error("Authorization error:", error);
  return Response.json(
    { error: "Internal server error" },  // ❌ No details!
    { status: 500 }
  );
}
```

The caregivers API was using `handleAuthError(e)` in its catch block, which stripped away all error details.

Solution Implemented

1. Enhanced `handleAuthError()` Function

File: `/src/lib/auth-utils.ts`

Changes:
- Added comprehensive console logging (visible in Render logs)
- Included error message, type, and stack trace in logs
- Added error details to response object
- Error response now includes:
- `error`: “Internal server error” (for backwards compatibility)
- `message`: The actual error message
- `type`: Error constructor name
- `stack`: Stack trace (in development only)

New Code:

```
export function handleAuthError(error: unknown) {
  if (error instanceof UnauthenticatedError) {
    return Response.json({ error: error.message }, { status: 401 });
  }

  if (error instanceof UnauthorizedError) {
    return Response.json({ error: error.message }, { status: 403 });
  }

  // ✅ NEW: Detailed logging for Render logs
  console.error("========================================");
  console.error("UNHANDLED ERROR IN AUTH UTILS");
  console.error("========================================");
  console.error("Error type:", error?.constructor?.name);
  console.error("Error object:", error);
  if (error instanceof Error) {
    console.error("Error message:", error.message);
    console.error("Error stack:", error.stack);
  }
  console.error("========================================");

  // ✅ NEW: Return detailed error information
  const errorResponse: any = {
    error: "Internal server error",
  };

  if (error instanceof Error) {
    errorResponse.message = error.message;  // ✅ Actual error message
    errorResponse.type = error.constructor.name;  // ✅ Error type
    if (process.env.NODE_ENV !== 'production') {
      errorResponse.stack = error.stack;  // ✅ Stack trace in dev
    }
  } else {
    errorResponse.details = String(error);
  }

  return Response.json(errorResponse, { status: 500 });
}
```

2. Simplified Caregivers API Error Handling

File: `/src/app/api/operator/caregivers/route.ts`

Changes:
- Removed duplicate error handling code
- Kept comprehensive console logging
- Consistently use `handleAuthError(e)` for all errors
- `handleAuthError` now provides detailed info, so no need for special handling

New Code:

```
} catch (e) {
  // ✅ Comprehensive logging (same as before)
  console.error('[Caregivers API] ========================================');
  console.error('[Caregivers API] CRITICAL ERROR OCCURRED');
  console.error('[Caregivers API] ========================================');
  console.error('[Caregivers API] Error type:', e?.constructor?.name);
  console.error('[Caregivers API] Error object:', e);
  if (e instanceof Error) {
    console.error('[Caregivers API] Error message:', e.message);
    console.error('[Caregivers API] Error stack:', e.stack);
  }
  console.error('[Caregivers API] ========================================');

  // ✅ SIMPLIFIED: Just use handleAuthError
  return handleAuthError(e);
}
```

3. Enhanced Frontend Error Display

File: `/src/app/operator/caregivers/page.tsx`

Changes:
- Parse error response JSON to extract details
- Display actual error message from API
- Include error type in message
- Graceful fallback if JSON parsing fails

New Code:

```
const fetchCaregivers = async () => {
  try {
    setLoading(true);
    const params = new URLSearchParams();
    if (employmentStatus !== 'ALL') params.append('status', employmentStatus);
    if (employmentType !== 'ALL') params.append('type', employmentType);

    const res = await fetch(`/api/operator/caregivers?${params.toString()}`);
    if (!res.ok) {
      // ✅ NEW: Parse error response
      try {
        const errorData = await res.json();
        const errorMessage = errorData.message || errorData.error || 'Failed to fetch
caregivers';
        const errorDetails = errorData.type ? ` (${errorData.type})` : '';
        throw new Error(errorMessage + errorDetails);
      } catch (parseError) {
        // ✅ Graceful fallback
        throw new Error(`Failed to fetch caregivers (${res.status})`);
      }
    }
    const data = await res.json();
    setCaregivers(data.caregivers || []);
  } catch (error) {
    console.error('Error fetching caregivers:', error);
    // ✅ Display actual error message
    const errorMessage = error instanceof Error ? error.message : 'Failed to load
caregivers';
    toast.error(errorMessage);
  } finally {
    setLoading(false);
  }
};
```

What Will Happen Now

1. In Render Logs

You will now see detailed error information:

```
========================================
UNHANDLED ERROR IN AUTH UTILS
========================================
Error type: PrismaClientKnownRequestError
Error object: [full error object]
Error message: Invalid `prisma.caregiver.findMany()` invocation...
Error stack: [full stack trace]
========================================
```

2. In API Response

The API will return:

```
{
  "error": "Internal server error",
  "message": "Invalid `prisma.caregiver.findMany()` invocation...",
  "type": "PrismaClientKnownRequestError"
}
```

3. In Frontend

Users will see:
- Toast notification with actual error message
- Example: “Invalid prisma query (PrismaClientKnownRequestError)”
- Instead of generic “Failed to load caregivers”

4. In Browser Console

Developers will see:
- Detailed error information
- Full error message and type
- Can copy/paste for debugging

Testing the Fix

Automatic Deployment

Render will automatically deploy this commit (c37addb) within 3-5 minutes.

Verification Steps

1. **Wait for Deployment**
   - Check Render dashboard for deployment status
   - Wait for “Build successful” and “Live” status

2. **Test the Page**
   - Visit: https://carelinkai.onrender.com/operator/caregivers
   - If error still occurs, you’ll now see:
     - Detailed error in browser console
     - Specific error message in toast notification
     - Full error details in Render logs

3. **Check Render Logs**
   - Go to Render dashboard → Logs
   - Look for the detailed error logging with “===” separators
   - You’ll see the actual error message, type, and stack trace

4. **Check API Response**
   - Open browser DevTools → Network tab
   - Reload the caregivers page
   - Click on the `/api/operator/caregivers` request
   - Look at the Response tab
   - You should see `message` and `type` fields in addition to `error`

What to Do Next

If You Still Get an Error

Good News: You will now see WHAT the error is!

1. **Check the Toast Message**
   - It will show the actual error (not “Internal server error”)
   - Example: “Cannot read property ‘languages’ of undefined (TypeError)”

2. **Check Browser Console**
   - Press F12 → Console tab
   - Look for “Error fetching caregivers:”
   - You’ll see the full error message

3. **Check Render Logs**
   - Most detailed information will be here
   - Look for the “===” separator lines
   - Copy the error message and stack trace
   - Share with me for quick fix

4. **Check API Response**
   - Network tab → Find the caregivers API call
   - Look at Response tab
   - Copy the JSON with error details
   - This tells us exactly what failed

If Everything Works

The page should load normally with the caregiver list.

Files Modified

1. ✅ `/src/lib/auth-utils.ts`
   - Enhanced `handleAuthError()` function
   - Added detailed logging
   - Included error details in response

2. ✅ `/src/app/api/operator/caregivers/route.ts`
   - Simplified error handling
   - Kept comprehensive logging
   - Consistent use of `handleAuthError`

3. ✅ `/src/app/operator/caregivers/page.tsx`
   - Parse error response JSON
   - Display detailed error messages
   - Better error handling

Expected Outcomes

Scenario A: Authentication Error

```
{
  "error": "You must be authenticated to access this resource",
  "type": "UnauthenticatedError"
}
```

Scenario B: Permission Error

```
{
  "error": "You do not have permission to view caregivers",
  "type": "UnauthorizedError"
}
```

Scenario C: Database Error

```
{
  "error": "Internal server error",
  "message": "Invalid `prisma.caregiver.findMany()` invocation: ...",
  "type": "PrismaClientKnownRequestError"
}
```

Scenario D: General Error

```
{
  "error": "Internal server error",
  "message": "Cannot read property 'languages' of undefined",
  "type": "TypeError"
}
```

Deployment Status

- ✅ Code committed: `c37addb`
- ✅ Pushed to GitHub: `profyt7/carelinkai`
- 🔄 Render deployment: **IN PROGRESS** (auto-deploys from main branch)
- ⏳ ETA: 3-5 minutes from now

Monitor at: https://dashboard.render.com/

Success Criteria

✅ **This fix is successful if**:
- Error messages are visible in Render logs
- API responses include `message` and `type` fields
- Frontend displays specific error messages
- We can identify the exact problem

❌ **This fix is NOT about**:
- Fixing the underlying error (we don't know what it is yet!)
- Making the page work (that comes after we identify the error)

Next Steps

1. **Wait 5 minutes** for Render to deploy
2. **Visit the page** again
3. **Copy the detailed error message** you see
4. **Share it with me** so I can fix the actual issue

The whole point of this fix is to make the error VISIBLE so we can fix it properly!