Caregivers Page Error Fix

Date

December 10, 2025

Problem Description

The caregivers page at `/operator/caregivers` was displaying an error with two symptoms:

1. **Server Error (500)**: API endpoint `/api/operator/caregivers` was returning a 500 Internal Server Error
2. **Client Error**: Browser console showed `TypeError: Cannot destructure property 'auth' of 'e' as it is undefined`

Error Details from Logs

Browser Console (f12console.txt):

```
/api/operator/caregivers?:1   Failed to load resource: the server responded with a
status of 500 ()
Error fetching caregivers: Error: Failed to fetch caregivers
TypeError: Cannot destructure property 'auth' of 'e' as it is undefined.
    at i (5424-b754ec3d8739fc6d.js:1:126963)
```

Render Logs:
- Build completed successfully with warnings
- No specific runtime error shown, but API was failing in production

Root Cause

The primary issue was in `/src/app/api/operator/caregivers/route.ts`:

Before (Problematic Code):

```
import { PrismaClient, UserRole } from '@prisma/client';
// ...
const prisma = new PrismaClient();
```

Problems:
1. **Creating new PrismaClient instances** - This violates the singleton pattern and can cause connection pool exhaustion
2. **Manual disconnect calls** - The code was calling `await prisma.$disconnect()` in finally blocks, which can interfere with connection pooling
3. **Connection leaks** - Multiple PrismaClient instances can lead to database connection issues

After (Fixed Code):

```
import { UserRole } from '@prisma/client';
import { prisma } from '@/lib/prisma';
// No new PrismaClient()
// No manual $disconnect() calls
```

Changes Made

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

1. **Replaced PrismaClient import**:
   - Changed from: `import { PrismaClient, UserRole } from '@prisma/client';`
   - Changed to: `import { UserRole } from '@prisma/client';` + `import { prisma } from '@/lib/prisma';`

2. **Removed PrismaClient instantiation**:
   - Removed: `const prisma = new PrismaClient();`

3. **Removed manual disconnect calls**:
   - Removed all `await prisma.$disconnect();` from finally blocks in both GET and POST handlers

Why This Fixes Both Errors

Server Error (500)

- The singleton prisma instance from `@/lib/prisma` properly manages database connections
- Prevents connection pool exhaustion that was causing the 500 errors
- Ensures consistent connection management across the application

Client Error (Auth Destructuring)

- This was a **symptom** of the server error, not a separate issue
- When the API failed with 500, the client-side error handling code received an unexpected response
- The “auth destructuring” error was likely due to error propagation in the minified client code
- With the API working correctly, this error should no longer occur

Singleton Pattern Explanation

The correct prisma singleton (`/src/lib/prisma.ts`) provides:
- Single shared PrismaClient instance across the application
- Proper connection pooling
- Development hot-reload compatibility
- Production-optimized logging
- No need for manual connection management

Testing

Expected Results

1. ✅ API endpoint `/api/operator/caregivers` returns 200 OK
2. ✅ Caregivers list loads successfully

3. ✅ No 500 errors in console
4. ✅ No auth destructuring errors in browser
5. ✅ Page displays caregivers or shows empty state

Verification Steps

1. Navigate to https://carelinkai.onrender.com/operator/caregivers
2. Check browser console - should see no errors
3. Verify page loads without “Something went wrong” message
4. Confirm API calls complete successfully in Network tab

Deployment

Commit

- **Hash**: `67866bc`
- **Message**: “fix: Replace PrismaClient instantiation with singleton in caregivers API”
- **Branch**: `main`

Auto-Deploy Status

- Pushed to GitHub: ✅
- Render auto-deploy: In progress
- Expected completion: ~5-10 minutes

Prevention

To prevent this issue in the future:

1. **Always use the singleton prisma import**:
   ```typescript
   import { prisma } from '@/lib/prisma';
   ```

2. **Never create new PrismaClient instances** in API routes:
   ```typescript
   // ❌ DON’T DO THIS
   const prisma = new PrismaClient();

// ✅ DO THIS
import { prisma } from ‘@/lib/prisma’;
```

1. **Never manually disconnect** the singleton:
   ```typescript
   // ❌ DON’T DO THIS
   try {
   // ... prisma queries
   } finally {
   await prisma.$disconnect();
   }

// ✅ DO THIS
try {
   ```

```
// ... prisma queries
}
// Singleton manages connections automatically
```

1. **Code review checklist**:
   - Check all API routes use singleton prisma import
   - Verify no `new PrismaClient()` instantiations
   - Ensure no manual `$disconnect()` calls

Related Files

- `/src/app/api/operator/caregivers/route.ts` - Fixed file
- `/src/lib/prisma.ts` - Singleton implementation
- `/src/app/operator/caregivers/page.tsx` - Client page that calls the API

References

- Previous similar fix: `CAREGIVERS_API_FIX_SUMMARY.md`
- Prisma singleton pattern: `/src/lib/prisma.ts`
- RBAC implementation: `/src/lib/rbac.ts`

Success Criteria

- [x] Code changes committed
- [x] Changes pushed to GitHub
- [ ] Render deployment completes successfully
- [ ] Caregivers page loads without errors
- [ ] API endpoint returns valid data
- [ ] No console errors in production

Notes

- This is the third attempt to fix the caregivers page
- The root cause was different from previous attempts which focused on:
  1. Missing CaregiverEmployment table
  2. API data structure mismatches
- This fix addresses the fundamental issue of database connection management

Monitoring

After deployment, monitor:
1. Render deployment logs for any errors
2. Application logs for database connection issues
3. Browser console on caregivers page
4. API response times and success rates