Request Header Fields Too Large - Critical Fix

Problem Description

Users were experiencing a **“Request Header Fields Too Large” (431)** error when trying to access the application after uploading a profile picture. This error prevented them from using the application entirely.

Root Cause

The issue was caused by storing **base64-encoded profile pictures in the JWT token**, which is then sent as a cookie with every HTTP request.

Why This Failed

1. **JWT Token Size**: JWT tokens are stored as cookies and sent with every request
2. **Base64 Image Size**: Profile pictures encoded in base64 can be **100KB - 500KB+**
3. **HTTP Header Limits**:
   - Typical cookie size limit: **4KB - 8KB**
   - Total HTTP header limit: **8KB - 16KB**
4. **Result**: The JWT cookie exceeded the maximum header size, causing the 431 error

The Problematic Code

In `/home/ubuntu/data_retriever_app/nextjs_space/lib/auth.ts`:

```
// ❌ BEFORE - Storing full base64 image in JWT token
async jwt({ token, user, account, profile, trigger }: any) {
  if (user) {
    token.image = user.image  // This could be 100KB+ of base64 data!
  }

  if (trigger === 'update') {
    const dbUser = await prisma.user.findUnique({ where: { id: token.sub } })
    token.image = dbUser.image  // Still storing huge base64 in token
  }

  return token
}

async session({ session, token }: any) {
  session.user.image = token.image  // Passing huge image to session
  return session
}
```

Solution Implemented

Architecture Change: Separate Image Storage from JWT

The fix involves **completely removing the profile picture from the JWT token** and fetching it separately from the database on each page load.

Changes Made

1. Updated JWT Callback (`lib/auth.ts`)

```
// ✅ AFTER - NO image storage in JWT token
async jwt({ token, user, account, profile, trigger }: any) {
  if (user) {
    token.role = user.role
    token.firstName = user.firstName
    token.lastName = user.lastName
    // REMOVED: token.image = user.image ✅
    token.companyName = user.companyName
    token.jobTitle = user.jobTitle
  }

  // Same for trigger === 'update'
  // REMOVED: token.image = dbUser.image ✅

  return token
}

async session({ session, token }: any) {
  // REMOVED: session.user.image = token.image ✅
  return session
}
```

2. Created Profile Image API Endpoint

New file: `/home/ubuntu/data_retriever_app/nextjs_space/app/api/user/profile-image/route.ts`

```
export async function GET(req: NextRequest) {
  const session = await getServerSession(authOptions);

  if (!session?.user?.id) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }

  // Fetch ONLY the image field from database
  const user = await prisma.user.findUnique({
    where: { id: session.user.id },
    select: { image: true }
  });

  return NextResponse.json({ image: user.image });
}
```

3. Updated Dashboard Page (`app/dashboard/page.tsx`)

```
export default async function Dashboard() {
  const session = await getServerSession(authOptions)

  if (!session) {
    redirect('/')
  }

  // ✅ Fetch profile image separately from database
  let profileImage = null;
  try {
    const user = await prisma.user.findUnique({
      where: { id: session.user.id },
      select: { image: true }
    });
    profileImage = user?.image || null;
  } catch (error) {
    console.error('Error fetching profile image:', error);
  }

  return (
    <DashboardClient
      user={{
        ...session.user,
        image: profileImage  // ✅ Pass image separately
      }}
    />
  )
}
```

4. Simplified Profile Edit Dialog

```
// ✅ No need to trigger session update anymore
if (type === 'picture' && imageFile) {
  const response = await fetch('/api/user/profile-picture', {
    method: 'POST',
    body: formDataToSend,
  });

  toast.success('Profile picture updated! Refreshing page...');
  onClose();

  // Simple reload - image will be fetched fresh from DB
  setTimeout(() => {
    window.location.href = window.location.href;
  }, 500);
}
```

How It Works Now

Before (Broken)

```
User uploads picture
  ↓
Saved to database (100KB base64)
  ↓
Stored in JWT token (100KB added to cookie!)
  ↓
JWT cookie sent with EVERY request
  ↓
❌ 431 Error: Request Header Fields Too Large
```

After (Fixed)

```
User uploads picture
  ↓
Saved to database (100KB base64)
  ↓
JWT token contains ONLY user ID, name, role (< 1KB)
  ↓
Dashboard loads:
  1. Get session (small JWT token) ✅
  2. Fetch image from DB separately ✅
  ↓
Profile picture displays correctly ✅
All requests work (headers are small) ✅
```

Benefits

1. ✅ **Fixed 431 Error**: JWT cookies are now < 1KB
2. ✅ **Better Security**: Large binary data doesn't travel in every request
3. ✅ **Performance**: Smaller headers = faster requests
4. ✅ **Best Practice**: JWT tokens should only contain small, essential claims
5. ✅ **Scalability**: No risk of hitting header limits as user data grows

Files Modified

1. `/home/ubuntu/data_retriever_app/nextjs_space/lib/auth.ts`
   - Removed `token.image` from JWT callback
   - Removed `session.user.image` from session callback

2. `/home/ubuntu/data_retriever_app/nextjs_space/app/api/user/profile-image/route.ts` (NEW)
   - Created endpoint to fetch profile image separately

3. `/home/ubuntu/data_retriever_app/nextjs_space/app/dashboard/page.tsx`
   - Added server-side image fetch before rendering

4. `/home/ubuntu/data_retriever_app/nextjs_space/components/profile-edit-dialog.tsx`
   - Simplified profile picture upload (removed session update call)

Testing Results

- ✅ Build successful
- ✅ No TypeScript errors
- ✅ JWT token size reduced from ~100KB+ to < 1KB
- ✅ Profile pictures load correctly
- ✅ No more 431 errors
- ✅ All requests work normally

Deployment Status

✅ **DEPLOYED TO PRODUCTION**: https://ncc-1701.io

The fix is now live. Users can:
- Upload profile pictures without errors
- Access the dashboard without 431 errors
- View profile pictures correctly
- Use all app features normally

Important Note

This fix affects **only profile pictures**. All other user data (name, role, email, etc.) remains in the JWT token as it’s small and essential for authorization.

Best Practices Applied

1. **JWT Tokens Should Be Small**: Only store essential claims
2. **Separate Large Data**: Store images, files, etc. separately
3. **Fetch on Demand**: Load large data when needed, not on every request
4. **Security First**: Don’t expose sensitive data in tokens unnecessarily

---

Date: November 5, 2025
Status: Deployed & Live
Priority: Critical Fix (App Breaking Issue)
Impact: All users with profile pictures