# **Developer Notes**

**Project**: Chat Application with Azure AD Authentication

**Framework**: Next.js 15.1.0 / React 18 / TypeScript

**Last Updated**: July 1, 2025

## **Project Overview**

### **Purpose**

A secure chat application that integrates with Azure Active Directory for authentication and connects to AI360 API for chat completions. The application provides a modern, responsive interface for AI-powered conversations.

### **Key Features**

* **Azure AD Authentication**: Secure login using Microsoft identities
* **AI Chat Interface**: Real-time chat with AI360 API
* **Responsive Design**: Mobile-first design with Tailwind CSS
* **TypeScript**: Full type safety throughout the application
* **Modern React**: Functional components with hooks
* **Next.js 15**: Latest features including App Router

### **Tech Stack**

Frontend:  
├── Next.js 15.1.0 (App Router)  
├── React 18  
├── TypeScript  
├── Tailwind CSS  
└── MSAL React (Azure AD)  
  
Backend/API:  
├── Next.js API Routes  
├── Azure AD Token Validation  
└── AI360 API Integration  
  
Development:  
├── ESLint  
├── Prettier  
├── Jest + React Testing Library  
├── Playwright (E2E)  
└── Husky (Git Hooks)

## **Architecture Overview**

### **Application Structure**

app/  
├── components/ # React components  
│ ├── AuthButton.tsx # Sign in/out button  
│ ├── AuthContent.tsx # Authentication wrapper  
│ ├── ChatInterface.tsx # Main chat component  
│ ├── MsalProvider.tsx # MSAL configuration wrapper  
│ └── ProtectedComponent.tsx # Route protection  
├── hooks/ # Custom React hooks  
│ └── useAccessToken.ts # Token management  
├── utils/ # Utility functions  
│ └── msal-config.ts # MSAL configuration  
├── api/ # API routes  
│ └── completions/ # Chat API endpoint  
├── types/ # TypeScript definitions  
├── globals.css # Global styles  
├── layout.tsx # Root layout  
└── page.tsx # Home page

### **Component Hierarchy**

App (layout.tsx)  
└── MsalProvider  
 └── AuthContent  
 ├── AuthButton  
 └── ProtectedComponent  
 └── ChatInterface

### **Data Flow**

1. User Authentication (Azure AD)  
 ↓  
2. Token Acquisition (MSAL)  
 ↓  
3. Chat Input (ChatInterface)  
 ↓  
4. API Request (/api/completions)  
 ↓  
5. AI360 API Call  
 ↓  
6. Response Processing  
 ↓  
7. UI Update (ChatInterface)

## **Development Environment Setup**

### **Prerequisites**

* **Node.js**: Version 18 or higher
* **npm**: Version 8 or higher
* **Git**: For version control
* **VS Code**: Recommended IDE with extensions

### **Required VS Code Extensions**

{  
 "recommendations": [  
 "bradlc.vscode-tailwindcss",  
 "esbenp.prettier-vscode",  
 "ms-vscode.vscode-typescript-next",  
 "ms-playwright.playwright",  
 "orta.vscode-jest"  
 ]  
}

### **Environment Variables**

Create .env.local in the project root:

# Azure AD Configuration  
AZURE\_CLIENT\_ID=your-azure-client-id  
AZURE\_TENANT\_ID=your-azure-tenant-id  
  
# AI360 API Configuration  
AI360\_API\_URL=https://api.lab45.ai/v1.1  
AI360\_API\_KEY=your-api-key  
  
# Development Settings  
NODE\_ENV=development  
NEXT\_PUBLIC\_APP\_ENV=development

### **Installation Steps**

# 1. Clone the repository  
git clone <repository-url>  
cd chat-app  
  
# 2. Install dependencies  
npm install  
  
# 3. Set up environment variables  
cp .env.example .env.local  
# Edit .env.local with your values  
  
# 4. Install Git hooks  
npm run prepare  
  
# 5. Start development server  
npm run dev

### **Development Scripts**

# Development  
npm run dev # Start development server  
npm run build # Build for production  
npm run start # Start production server  
  
# Code Quality  
npm run lint # Run ESLint  
npm run lint:fix # Fix ESLint issues  
npm run lint:strict # ESLint with max warnings = 0  
npm run format # Format code with Prettier  
npm run format:check # Check formatting  
npm run type-check # TypeScript type checking  
  
# Testing  
npm run test # Run all tests  
npm run test:watch # Run tests in watch mode  
npm run test:coverage # Run tests with coverage  
npm run test:unit # Run unit tests only  
npm run test:integration # Run integration tests  
npm run test:e2e # Run E2E tests  
npm run test:e2e:ui # Run E2E tests with UI  
  
# Git Hooks  
npm run pre-commit # Run pre-commit checks

## **Code Organization**

### **File Naming Conventions**

Components: PascalCase.tsx (e.g., ChatInterface.tsx)  
Hooks: camelCase.ts (e.g., useAccessToken.ts)  
Utils: kebab-case.ts (e.g., msal-config.ts)  
Types: camelCase.ts (e.g., chatTypes.ts)  
Tests: \*.test.tsx/ts (e.g., ChatInterface.test.tsx)  
API Routes: route.ts (e.g., api/completions/route.ts)

### **Import Organization**

Follow this order for imports:

// 1. React and Next.js imports  
import React, { useState, useEffect } from 'react';  
import { NextResponse } from 'next/server';  
  
// 2. Third-party libraries  
import { useMsal } from '@azure/msal-react';  
  
// 3. Internal components and hooks  
import { useAccessToken } from '@/app/hooks/useAccessToken';  
import ChatInterface from '@/app/components/ChatInterface';  
  
// 4. Utils and types  
import { msalConfig } from '@/app/utils/msal-config';  
import type { ChatMessage } from '@/app/types';  
  
// 5. Relative imports  
import './styles.css';

### **Component Structure Template**

"use client";  
  
import React, { useState, useCallback } from 'react';  
// Other imports...  
  
interface ComponentProps {  
 // Define props with types  
 title: string;  
 onSubmit?: (data: FormData) => void;  
}  
  
export default function ComponentName({ title, onSubmit }: ComponentProps) {  
 // 1. Hooks (state, effects, custom hooks)  
 const [isLoading, setIsLoading] = useState(false);  
 const { accessToken } = useAccessToken();  
  
 // 2. Event handlers and functions  
 const handleSubmit = useCallback(async (e: React.FormEvent) => {  
 e.preventDefault();  
 // Handle submission  
 }, []);  
  
 // 3. Effects  
 useEffect(() => {  
 // Side effects  
 }, []);  
  
 // 4. Early returns  
 if (!accessToken) {  
 return <div>Please sign in</div>;  
 }  
  
 // 5. Render  
 return (  
 <div className="container">  
 <h1>{title}</h1>  
 {/\* Component content \*/}  
 </div>  
 );  
}

### **Custom Hook Template**

"use client";  
  
import { useState, useEffect, useCallback } from 'react';  
  
interface HookReturn {  
 data: DataType | null;  
 isLoading: boolean;  
 error: string | null;  
 execute: () => Promise<void>;  
}  
  
export function useCustomHook(param: string): HookReturn {  
 // State  
 const [data, setData] = useState<DataType | null>(null);  
 const [isLoading, setIsLoading] = useState(false);  
 const [error, setError] = useState<string | null>(null);  
  
 // Functions  
 const execute = useCallback(async () => {  
 setIsLoading(true);  
 setError(null);  
   
 try {  
 // Async operation  
 const result = await fetchData(param);  
 setData(result);  
 } catch (err) {  
 setError(err instanceof Error ? err.message : 'Unknown error');  
 } finally {  
 setIsLoading(false);  
 }  
 }, [param]);  
  
 // Effects  
 useEffect(() => {  
 execute();  
 }, [execute]);  
  
 return { data, isLoading, error, execute };  
}

## **Authentication Flow**

### **MSAL Configuration**

// app/utils/msal-config.ts  
import { Configuration, PopupRequest } from '@azure/msal-browser';  
  
export const msalConfig: Configuration = {  
 auth: {  
 clientId: process.env.AZURE\_CLIENT\_ID!,  
 authority: `https://login.microsoftonline.com/${process.env.AZURE\_TENANT\_ID}`,  
 redirectUri: typeof window !== 'undefined' ? window.location.origin : '/',  
 },  
 cache: {  
 cacheLocation: 'sessionStorage',  
 storeAuthStateInCookie: false,  
 },  
};  
  
export const loginRequest: PopupRequest = {  
 scopes: ['User.Read'],  
};

### **Authentication States**

// Authentication flow states  
enum AuthState {  
 LOADING = 'loading', // Checking authentication status  
 UNAUTHENTICATED = 'unauthenticated', // User not signed in  
 AUTHENTICATED = 'authenticated', // User signed in successfully  
 ERROR = 'error' // Authentication error occurred  
}

### **Token Management**

// Current implementation in useAccessToken.ts  
export function useAccessToken() {  
 const { instance, accounts } = useMsal();  
 const [accessToken, setAccessToken] = useState<string | null>(null);  
  
 useEffect(() => {  
 const getToken = async () => {  
 if (accounts.length > 0) {  
 try {  
 // Try silent token acquisition first  
 const response = await instance.acquireTokenSilent({  
 ...loginRequest,  
 account: accounts[0]  
 });  
 setAccessToken(response.accessToken);  
 } catch (error) {  
 // Fallback to popup if silent fails  
 try {  
 const response = await instance.acquireTokenPopup(loginRequest);  
 setAccessToken(response.accessToken);  
 } catch (error) {  
 console.error("Failed to get access token:", error);  
 }  
 }  
 }  
 };  
  
 getToken();  
 }, [instance, accounts]);  
  
 return accessToken;  
}

### **Authentication Guard Pattern**

// Protecting components that require authentication  
function ProtectedComponent({ children }: { children: React.ReactNode }) {  
 const { accounts } = useMsal();  
 const accessToken = useAccessToken();  
  
 if (accounts.length === 0) {  
 return <AuthButton />;  
 }  
  
 if (!accessToken) {  
 return <div>Loading...</div>;  
 }  
  
 return <>{children}</>;  
}

## **API Integration**

### **API Route Structure**

// app/api/completions/route.ts  
import { NextRequest, NextResponse } from 'next/server';  
  
export async function POST(request: NextRequest) {  
 try {  
 // 1. Extract and validate request data  
 const { prompt } = await request.json();  
   
 if (!prompt) {  
 return NextResponse.json(  
 { error: 'Prompt is required' },  
 { status: 400 }  
 );  
 }  
  
 // 2. Validate authentication  
 const authHeader = request.headers.get('authorization');  
 if (!authHeader?.startsWith('Bearer ')) {  
 return NextResponse.json(  
 { error: 'Unauthorized' },  
 { status: 401 }  
 );  
 }  
  
 const token = authHeader.substring(7);  
 // Validate token with Azure AD (implementation needed)  
  
 // 3. Call external API  
 const response = await fetch(`${process.env.AI360\_API\_URL}/completions`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${process.env.AI360\_API\_KEY}`,  
 },  
 body: JSON.stringify({  
 model: 'gpt-3.5-turbo',  
 messages: [{ role: 'user', content: prompt }],  
 }),  
 });  
  
 // 4. Process and return response  
 const data = await response.json();  
 return NextResponse.json({ data });  
  
 } catch (error) {  
 console.error('API Error:', error);  
 return NextResponse.json(  
 { error: 'Internal Server Error' },  
 { status: 500 }  
 );  
 }  
}

### **Client-Side API Calls**

// Making API calls from components  
async function sendChatMessage(prompt: string, accessToken: string) {  
 const response = await fetch('/api/completions', {  
 method: 'POST',  
 headers: {  
 'Authorization': `Bearer ${accessToken}`,  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({ prompt }),  
 });  
  
 if (!response.ok) {  
 const error = await response.json();  
 throw new Error(error.error || 'Failed to send message');  
 }  
  
 return response.json();  
}

### **Error Handling Pattern**

// Standardized error handling  
interface ApiError {  
 message: string;  
 code?: string;  
 status?: number;  
}  
  
class ApiClient {  
 private async handleResponse<T>(response: Response): Promise<T> {  
 if (!response.ok) {  
 const errorData = await response.json().catch(() => ({}));  
 throw new Error(  
 errorData.error ||   
 errorData.message ||   
 `HTTP ${response.status}: ${response.statusText}`  
 );  
 }  
   
 return response.json();  
 }  
  
 async post<T>(url: string, data: any, token?: string): Promise<T> {  
 const response = await fetch(url, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 ...(token && { 'Authorization': `Bearer ${token}` }),  
 },  
 body: JSON.stringify(data),  
 });  
  
 return this.handleResponse<T>(response);  
 }  
}

## **State Management**

### **Local Component State**

// Simple state for component-level data  
function ChatInterface() {  
 const [input, setInput] = useState("");  
 const [messages, setMessages] = useState<Message[]>([]);  
 const [isLoading, setIsLoading] = useState(false);  
  
 // State updates  
 const addMessage = (message: Message) => {  
 setMessages(prev => [...prev, message]);  
 };  
  
 return (  
 // Component JSX  
 );  
}

### **Custom Hooks for State Logic**

// Extract complex state logic into custom hooks  
function useChat() {  
 const [messages, setMessages] = useState<Message[]>([]);  
 const [isLoading, setIsLoading] = useState(false);  
 const accessToken = useAccessToken();  
  
 const sendMessage = useCallback(async (content: string) => {  
 if (!content.trim() || !accessToken) return;  
  
 const userMessage: Message = {  
 role: 'user',  
 content: content.trim(),  
 timestamp: Date.now(),  
 id: crypto.randomUUID()  
 };  
  
 setMessages(prev => [...prev, userMessage]);  
 setIsLoading(true);  
  
 try {  
 const response = await sendChatMessage(content, accessToken);  
 const assistantMessage: Message = {  
 role: 'assistant',  
 content: response.data.content,  
 timestamp: Date.now(),  
 id: crypto.randomUUID()  
 };  
 setMessages(prev => [...prev, assistantMessage]);  
 } catch (error) {  
 // Handle error  
 } finally {  
 setIsLoading(false);  
 }  
 }, [accessToken]);  
  
 return { messages, isLoading, sendMessage };  
}

### **Context for Global State**

// For state that needs to be shared across components  
interface AppContextType {  
 user: User | null;  
 theme: 'light' | 'dark';  
 setTheme: (theme: 'light' | 'dark') => void;  
}  
  
const AppContext = createContext<AppContextType | undefined>(undefined);  
  
export function AppProvider({ children }: { children: React.ReactNode }) {  
 const [theme, setTheme] = useState<'light' | 'dark'>('light');  
 const { accounts } = useMsal();  
 const user = accounts[0] || null;  
  
 return (  
 <AppContext.Provider value={{ user, theme, setTheme }}>  
 {children}  
 </AppContext.Provider>  
 );  
}  
  
export function useAppContext() {  
 const context = useContext(AppContext);  
 if (!context) {  
 throw new Error('useAppContext must be used within AppProvider');  
 }  
 return context;  
}

## **Styling Guidelines**

### **Tailwind CSS Setup**

// tailwind.config.ts  
import type { Config } from 'tailwindcss'  
  
const config: Config = {  
 content: [  
 './pages/\*\*/\*.{js,ts,jsx,tsx,mdx}',  
 './components/\*\*/\*.{js,ts,jsx,tsx,mdx}',  
 './app/\*\*/\*.{js,ts,jsx,tsx,mdx}',  
 ],  
 theme: {  
 extend: {  
 colors: {  
 primary: {  
 50: '#eff6ff',  
 500: '#3b82f6',  
 900: '#1e3a8a',  
 }  
 }  
 },  
 },  
 plugins: [],  
}

### **Component Styling Patterns**

// Consistent spacing and responsive design  
function ChatInterface() {  
 return (  
 <div className="w-full max-w-2xl mx-auto p-4 space-y-4">  
 {/\* Messages container \*/}  
 <div className="bg-white/5 rounded-lg p-4 space-y-4 min-h-[300px] max-h-[500px] overflow-y-auto">  
 {/\* Messages \*/}  
 </div>  
  
 {/\* Input form \*/}  
 <form className="flex gap-2">  
 <input   
 className="flex-1 rounded-lg p-2 bg-white/5 border border-white/10 focus:outline-none focus:ring-2 focus:ring-blue-500"  
 />  
 <button   
 className="px-4 py-2 rounded-lg bg-blue-500 text-white disabled:opacity-50 hover:bg-blue-600 transition-colors"  
 >  
 Send  
 </button>  
 </form>  
 </div>  
 );  
}

### **Responsive Design Patterns**

/\* Mobile-first approach \*/  
.chat-container {  
 @apply w-full p-4;  
}  
  
/\* Tablet and up \*/  
@screen md {  
 .chat-container {  
 @apply max-w-2xl mx-auto;  
 }  
}  
  
/\* Desktop and up \*/  
@screen lg {  
 .chat-container {  
 @apply max-w-4xl;  
 }  
}

### **Dark Mode Support**

// Using CSS variables for theme support  
function ThemeProvider({ children }: { children: React.ReactNode }) {  
 const [theme, setTheme] = useState<'light' | 'dark'>('light');  
  
 useEffect(() => {  
 document.documentElement.setAttribute('data-theme', theme);  
 }, [theme]);  
  
 return (  
 <div className={theme}>  
 {children}  
 </div>  
 );  
}

## **Testing Strategy**

### **Unit Testing with Jest**

// \_\_tests\_\_/components/ChatInterface.test.tsx  
import { render, screen, fireEvent, waitFor } from '@testing-library/react';  
import ChatInterface from '@/app/components/ChatInterface';  
  
// Mock dependencies  
jest.mock('@/app/hooks/useAccessToken', () => ({  
 useAccessToken: () => 'mock-token'  
}));  
  
describe('ChatInterface', () => {  
 beforeEach(() => {  
 global.fetch = jest.fn();  
 });  
  
 afterEach(() => {  
 jest.resetAllMocks();  
 });  
  
 it('should render chat interface', () => {  
 render(<ChatInterface />);  
 expect(screen.getByPlaceholderText('Type your message...')).toBeInTheDocument();  
 });  
  
 it('should send message on form submit', async () => {  
 const mockResponse = {  
 ok: true,  
 json: () => Promise.resolve({ data: { content: 'AI response' } })  
 };  
 (global.fetch as jest.Mock).mockResolvedValue(mockResponse);  
  
 render(<ChatInterface />);  
   
 const input = screen.getByPlaceholderText('Type your message...');  
 const button = screen.getByText('Send');  
  
 fireEvent.change(input, { target: { value: 'Hello' } });  
 fireEvent.click(button);  
  
 await waitFor(() => {  
 expect(screen.getByText('Hello')).toBeInTheDocument();  
 });  
 });  
});

### **Integration Testing**

// \_\_tests\_\_/integration/auth-flow.test.tsx  
import { render, screen, fireEvent, waitFor } from '@testing-library/react';  
import { MsalProvider } from '@azure/msal-react';  
import App from '@/app/page';  
  
describe('Authentication Flow', () => {  
 it('should redirect unauthenticated users to sign in', async () => {  
 const mockMsalInstance = {  
 acquireTokenSilent: jest.fn(),  
 loginPopup: jest.fn()  
 };  
  
 render(  
 <MsalProvider instance={mockMsalInstance}>  
 <App />  
 </MsalProvider>  
 );  
  
 expect(screen.getByText('Sign In')).toBeInTheDocument();  
 });  
});

### **E2E Testing with Playwright**

// e2e/chat-flow.spec.ts  
import { test, expect } from '@playwright/test';  
  
test.describe('Chat Application', () => {  
 test.beforeEach(async ({ page }) => {  
 await page.goto('/');  
 });  
  
 test('should complete full chat flow', async ({ page }) => {  
 // Mock authentication  
 await page.route('\*\*/oauth2/v2.0/token', async route => {  
 await route.fulfill({  
 status: 200,  
 contentType: 'application/json',  
 body: JSON.stringify({  
 access\_token: 'mock-token',  
 token\_type: 'Bearer'  
 })  
 });  
 });  
  
 // Mock API response  
 await page.route('\*\*/api/completions', async route => {  
 await route.fulfill({  
 status: 200,  
 contentType: 'application/json',  
 body: JSON.stringify({  
 data: { content: 'AI response to your message' }  
 })  
 });  
 });  
  
 // Interact with the application  
 await page.fill('input[placeholder\*="Type your message"]', 'Hello AI');  
 await page.click('button:has-text("Send")');  
  
 // Verify results  
 await expect(page.locator('text=Hello AI')).toBeVisible();  
 await expect(page.locator('text=AI response to your message')).toBeVisible();  
 });  
});

## **Build and Deployment**

### **Build Configuration**

// next.config.ts  
import type { NextConfig } from 'next'  
  
const nextConfig: NextConfig = {  
 reactStrictMode: true,  
 swcMinify: true,  
   
 env: {  
 CUSTOM\_KEY: process.env.CUSTOM\_KEY,  
 },  
   
 experimental: {  
 optimizePackageImports: ['@azure/msal-react'],  
 },  
   
 webpack: (config, { buildId, dev, isServer, defaultLoaders, webpack }) => {  
 // Custom webpack configuration  
 return config;  
 },  
}  
  
export default nextConfig

### **Environment-Specific Builds**

# Development build  
npm run build  
  
# Production build with optimizations  
NODE\_ENV=production npm run build  
  
# Build with bundle analysis  
ANALYZE=true npm run build

### **Deployment Checklist**

* Environment variables configured
* Azure AD app registration updated
* API endpoints configured
* SSL certificates in place
* Build optimizations enabled
* Error tracking configured
* Performance monitoring setup

### **Vercel Deployment**

// vercel.json  
{  
 "buildCommand": "npm run build",  
 "devCommand": "npm run dev",  
 "installCommand": "npm install",  
 "framework": "nextjs",  
 "functions": {  
 "app/api/completions/route.ts": {  
 "maxDuration": 30  
 }  
 },  
 "headers": [  
 {  
 "source": "/api/(.\*)",  
 "headers": [  
 {  
 "key": "Access-Control-Allow-Origin",  
 "value": "\*"  
 }  
 ]  
 }  
 ]  
}

## **Common Development Tasks**

### **Adding a New Component**

1. Create component file in app/components/
2. Follow component template structure
3. Add TypeScript interfaces for props
4. Implement component logic
5. Add styles using Tailwind CSS
6. Write unit tests
7. Update documentation

### **Adding a New API Route**

1. Create route file in app/api/
2. Implement HTTP methods (GET, POST, etc.)
3. Add request validation
4. Implement authentication checks
5. Add error handling
6. Write integration tests
7. Update API documentation

### **Adding a New Hook**

1. Create hook file in app/hooks/
2. Follow hook template structure
3. Add TypeScript types
4. Implement hook logic
5. Add proper dependencies
6. Write hook tests
7. Update hooks documentation

### **Environment Variable Management**

# Development  
cp .env.example .env.local  
# Edit .env.local  
  
# Staging  
# Set environment variables in deployment platform  
  
# Production  
# Set environment variables in deployment platform  
# Ensure all secrets are properly secured

### **Database Migrations (if applicable)**

# Run migrations  
npm run db:migrate  
  
# Rollback migrations  
npm run db:rollback  
  
# Reset database  
npm run db:reset

## **Troubleshooting Guide**

### **Common Issues**

#### **1. Authentication Issues**

**Problem**: Token acquisition fails

// Debug token acquisition  
const { instance, accounts, inProgress } = useMsal();  
  
console.log('MSAL State:', {  
 accounts: accounts.length,  
 inProgress,  
 config: instance.getConfiguration()  
});

**Solutions**:

* Check Azure AD app registration
* Verify client ID and tenant ID
* Check redirect URI configuration
* Clear browser cache and session storage

#### **2. API Call Failures**

**Problem**: API requests return 401/403 errors

// Debug API calls  
const debugApiCall = async (url: string, token: string) => {  
 console.log('API Call Debug:', {  
 url,  
 tokenLength: token?.length,  
 tokenPrefix: token?.substring(0, 10)  
 });  
   
 try {  
 const response = await fetch(url, {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
 console.log('Response status:', response.status);  
 console.log('Response headers:', response.headers);  
 } catch (error) {  
 console.error('API call failed:', error);  
 }  
};

**Solutions**:

* Verify token is not expired
* Check API endpoint URL
* Validate request headers
* Check CORS configuration

#### **3. Build Errors**

**Problem**: TypeScript compilation errors

# Check TypeScript errors  
npm run type-check  
  
# Check for unused imports  
npm run lint  
  
# Check for formatting issues  
npm run format:check

**Solutions**:

* Fix TypeScript type errors
* Remove unused imports
* Update deprecated APIs
* Check for missing dependencies

#### **4. Performance Issues**

**Problem**: Slow component renders

// Debug component performance  
import { Profiler } from 'react';  
  
function onRenderCallback(id, phase, actualDuration) {  
 console.log('Component render:', {  
 id,  
 phase,  
 actualDuration  
 });  
}  
  
<Profiler id="ChatInterface" onRender={onRenderCallback}>  
 <ChatInterface />  
</Profiler>

**Solutions**:

* Add React.memo for expensive components
* Use useCallback for event handlers
* Use useMemo for expensive calculations
* Optimize re-render patterns

### **Debugging Tools**

#### **React Developer Tools**

// Add debug names for hooks  
function useAccessToken() {  
 const token = useState(null);  
   
 // This will show in React DevTools  
 useDebugValue(token ? 'Token acquired' : 'No token');  
   
 return token;  
}

#### **Network Debugging**

// Log all fetch requests  
const originalFetch = window.fetch;  
window.fetch = async (url, options) => {  
 console.log('Fetch:', { url, options });  
 const response = await originalFetch(url, options);  
 console.log('Response:', { status: response.status, url });  
 return response;  
};

#### **Local Storage Debugging**

// Monitor localStorage changes  
const originalSetItem = localStorage.setItem;  
localStorage.setItem = function(key, value) {  
 console.log('localStorage.setItem:', { key, value });  
 originalSetItem.call(this, key, value);  
};

## **Performance Considerations**

### **Bundle Size Optimization**

# Analyze bundle size  
npm run build  
npx @next/bundle-analyzer  
  
# Check for large dependencies  
npm ls --depth=0 --long

### **Code Splitting**

// Dynamic imports for large components  
const HeavyComponent = dynamic(() => import('./HeavyComponent'), {  
 loading: () => <div>Loading...</div>,  
 ssr: false  
});  
  
// Route-based code splitting (automatic with Next.js App Router)

### **Image Optimization**

// Use Next.js Image component  
import Image from 'next/image';  
  
<Image  
 src="/avatar.png"  
 alt="User Avatar"  
 width={32}  
 height={32}  
 priority // For above-the-fold images  
/>

### **Caching Strategies**

// API response caching  
const cache = new Map();  
  
async function cachedApiCall(url: string) {  
 if (cache.has(url)) {  
 return cache.get(url);  
 }  
   
 const response = await fetch(url);  
 const data = await response.json();  
   
 cache.set(url, data);  
 return data;  
}

## **Security Notes**

### **Input Validation**

// Sanitize user input  
import DOMPurify from 'dompurify';  
  
function sanitizeInput(input: string): string {  
 return DOMPurify.sanitize(input, { ALLOWED\_TAGS: [] });  
}

### **Token Security**

// Secure token storage  
// Good: Use secure, httpOnly cookies for sensitive tokens  
// Good: Use sessionStorage for temporary tokens  
// Bad: Don't store sensitive tokens in localStorage

### **API Security**

// Validate API requests  
export async function POST(request: NextRequest) {  
 // 1. Validate authentication  
 const token = request.headers.get('authorization');  
 if (!token) {  
 return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });  
 }  
  
 // 2. Validate input  
 const body = await request.json();  
 if (!body.prompt || typeof body.prompt !== 'string') {  
 return NextResponse.json({ error: 'Invalid input' }, { status: 400 });  
 }  
  
 // 3. Rate limiting (implement as needed)  
 // 4. Process request  
}

### **Content Security Policy**

// middleware.ts  
export function middleware(request: NextRequest) {  
 const response = NextResponse.next();  
   
 response.headers.set(  
 'Content-Security-Policy',  
 "default-src 'self'; " +  
 "script-src 'self' 'unsafe-eval'; " +  
 "style-src 'self' 'unsafe-inline'; " +  
 "connect-src 'self' <https://login.microsoftonline.com> <https://api.lab45.ai>;"  
 );  
   
 return response;  
}

## **Known Issues and Workarounds**

### **Issue 1: MSAL Popup Blocked**

**Problem**: Browser blocks authentication popup **Workaround**:

const handleSignIn = async () => {  
 try {  
 // Try popup first  
 await instance.loginPopup(loginRequest);  
 } catch (error) {  
 if (error.message.includes('popup\_window\_error')) {  
 // Fallback to redirect  
 await instance.loginRedirect(loginRequest);  
 }  
 }  
};

### **Issue 2: Next.js Hydration Mismatch**

**Problem**: Server/client rendering differences **Workaround**:

const [isMounted, setIsMounted] = useState(false);  
  
useEffect(() => {  
 setIsMounted(true);  
}, []);  
  
if (!isMounted) {  
 return <div>Loading...</div>;  
}

### **Issue 3: TypeScript Type Conflicts**

**Problem**: MSAL types conflict with Next.js types **Workaround**:

// Create type declaration file  
// types/msal.d.ts  
declare module '@azure/msal-react' {  
 // Re-export types to avoid conflicts  
}

### **Issue 4: Environment Variable Loading**

**Problem**: Environment variables not available in client **Workaround**:

// Use NEXT\_PUBLIC\_ prefix for client-side variables  
NEXT\_PUBLIC\_API\_URL=https://api.example.com  
  
// Access in components  
const apiUrl = process.env.NEXT\_PUBLIC\_API\_URL;