# **Low-Level Design (LLD) Document**

## **Chat Application with Microsoft Authentication**

### **Document Information**

* **Project Name**: MSAL Chat Application
* **Version**: 0.1.0
* **Created**: June 30, 2025
* **Technology Stack**: Next.js 15.1.0, React 18, TypeScript 5, Tailwind CSS, Azure MSAL

## **1. System Overview**

### **1.1 Purpose**

This application is a web-based chat interface that integrates with Microsoft Azure Active Directory (AAD) for authentication and connects to an AI completion service for conversational AI capabilities.

### **1.2 Architecture Style**

* **Frontend**: Single Page Application (SPA) using Next.js with React
* **Authentication**: Microsoft Authentication Library (MSAL) with OAuth 2.0/OpenID Connect
* **API**: RESTful API with Next.js API routes
* **Deployment**: Client-side rendering with server-side API endpoints

### **1.3 Key Features**

* Microsoft SSO authentication
* Real-time chat interface
* AI-powered responses via Lab45 AI API
* Responsive UI with Tailwind CSS
* Token-based authorization

## **2. System Architecture**

### **2.1 High-Level Component Diagram**

┌─────────────────────────────────────────────────────────────────┐  
│ Browser (Client) │  
├─────────────────────────────────────────────────────────────────┤  
│ ┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐ │  
│ │ MsalProvider │ │ AuthButton │ │ ChatInterface │ │  
│ │ │ │ │ │ │ │  
│ └─────────────────┘ └─────────────────┘ └─────────────────┘ │  
│ │ │ │  
│ ┌─────────────────────────────────────────┐ │ │  
│ │ useAccessToken Hook │ │ │  
│ └─────────────────────────────────────────┘ │ │  
├─────────────────────────────────────────────────────────────────┤  
│ Next.js Application │  
├─────────────────────────────────────────────────────────────────┤  
│ API Routes Layer │  
│ ┌─────────────────────────────────────────────────────────────┐ │  
│ │ /api/completions/route.ts │ │  
│ └─────────────────────────────────────────────────────────────┘ │  
└─────────────────────────────────────────────────────────────────┘  
 │  
 ▼  
┌─────────────────────────────────────────────────────────────────┐  
│ External Services │  
├─────────────────────────────────────────────────────────────────┤  
│ ┌─────────────────┐ ┌─────────────────────────────┐ │  
│ │ Azure AAD │ │ Lab45 AI API │ │  
│ │ (MSAL Auth) │ │ (api.lab45.ai/v1.1) │ │  
│ └─────────────────┘ └─────────────────────────────┘ │  
└─────────────────────────────────────────────────────────────────┘

### **2.2 Data Flow Architecture**

User Action → Authentication → Token Acquisition → API Call → AI Response → UI Update  
 │ │ │ │ │ │  
 ▼ ▼ ▼ ▼ ▼ ▼  
[Click Login] → [MSAL Popup] → [Access Token] → [/api/completions] → [AI Response] → [Chat UI]

## **3. Component Design**

### **3.1 Frontend Components**

#### **3.1.1 MsalProvider Component**

**File**: app/components/MsalProvider.tsx

**Purpose**: Root-level wrapper that provides MSAL context to the entire application

**Dependencies**:

* @azure/msal-react
* @/app/utils/msal-config

**Properties**:

interface Props {  
 children: React.ReactNode;  
}  
 **Key Responsibilities**:

* Initialize MSAL instance
* Provide authentication context to child components
* Handle client-side only rendering

#### **3.1.2 AuthButton Component**

**File**: app/components/AuthButton.tsx

**Purpose**: Handle user authentication (login/logout) with Microsoft

**Dependencies**:

* @azure/msal-react (useMsal hook)
* @/app/utils/msal-config (loginRequest)

**State Management**:

// Derived from MSAL context  
const { instance, accounts } = useMsal();

**Key Methods**:

* handleLogin(): Initiates Microsoft login popup
* handleLogout(): Logs out user and clears session

**UI States**:

* **Unauthenticated**: "Sign In with Microsoft" button
* **Authenticated**: "Sign Out (username)" button

#### **3.1.3 AuthContent Component**

**File**: app/components/AuthContent.tsx

**Purpose**: Conditional rendering based on authentication status

**Dependencies**:

* @azure/msal-react (AuthenticatedTemplate, UnauthenticatedTemplate)
* ./ChatInterface

**Conditional Rendering**:

* **Authenticated**: Shows chat interface
* **Unauthenticated**: Shows welcome message

#### **3.1.4 ChatInterface Component**

**File**: app/components/ChatInterface.tsx

**Purpose**: Main chat functionality with AI interaction

**Dependencies**:

* @/app/hooks/useAccessToken
* React hooks (useState)

**State Management**:

interface Message {  
 role: 'user' | 'assistant';  
 content: string;  
}  
  
const [input, setInput] = useState<string>("");  
const [messages, setMessages] = useState<Message[]>([]);  
const [isLoading, setIsLoading] = useState<boolean>(false);

**Key Methods**:

* handleSubmit(): Processes user input and calls AI API
* Message rendering and UI management

### **3.2 Custom Hooks**

#### **3.2.1 useAccessToken Hook**

**File**: app/hooks/useAccessToken.ts

**Purpose**: Manages access token acquisition and refresh

**Dependencies**:

* @azure/msal-react (useMsal)
* ../utils/msal-config (loginRequest)

**Token Acquisition Flow**:

1. Check if user is authenticated
2. Attempt silent token acquisition
3. Fallback to popup token acquisition if silent fails
4. Return access token or null

**Return Type**:

string | null

### **3.3 Configuration Modules**

#### **3.3.1 MSAL Configuration**

**File**: app/utils/msal-config.ts

**Purpose**: Central configuration for Microsoft Authentication

**Configuration Object**:

interface MSALConfig {  
 auth: {  
 clientId: string; // Azure App Registration ID  
 authority: string; // Azure tenant authority URL  
 redirectUri: string; // Post-login redirect URL  
 };  
 cache: {  
 cacheLocation: "sessionStorage";  
 storeAuthStateInCookie: boolean;  
 };  
}

**Exports**:

* msalConfig: MSAL configuration object
* loginRequest: Scope configuration for token requests
* msalInstance: Initialized PublicClientApplication

## **4. API Design**

### **4.1 Completions API Endpoint**

#### **4.1.1 Endpoint Details**

**Route**: /api/completions

**File**: app/api/completions/route.ts

**Method**: POST

**Purpose**: Proxy API for AI completions with authentication validation

#### **4.1.2 Request Specification**

**Headers**:

{  
 'Authorization': 'Bearer <access\_token>',  
 'Content-Type': 'application/json'  
}

**Request Body**:

interface CompletionRequest {  
 prompt: string;  
}

**Example**:

{  
 "prompt": "What is artificial intelligence?"  
}

#### **4.1.3 Response Specification**

**Success Response** (200):

interface CompletionResponse {  
 data: {  
 content: string;  
 };  
}

**Error Responses**:

* **401 Unauthorized**: Missing authorization token
* **500 Internal Server Error**: API or processing errors

#### **4.1.4 External API Integration**

**Target API**: Lab45 AI API

**Endpoint**: <https://api.lab45.ai/v1.1/skills/completion/query>

**Request Transformation**:

// Client request  
{ prompt: "user message" }  
  
// Transformed to Lab45 API format  
{  
 messages: [{ role: "user", content: "user message" }],  
 skill\_parameters: {  
 model\_name: "gpt-35-turbo-16k",  
 max\_output\_tokens: 1000,  
 temperature: 0.7  
 }  
}

**Response Processing**:

* Handles streaming response from Lab45 API
* Parses NDJSON (Newline Delimited JSON) chunks
* Concatenates content fragments
* Returns unified response object

## **5. Security Design**

### **5.1 Authentication Flow**

#### **5.1.1 Microsoft OAuth 2.0 Flow**

1. User clicks "Sign In with Microsoft"  
2. MsalProvider redirects to Microsoft login  
3. User authenticates with Microsoft credentials  
4. Microsoft returns authorization code  
5. MSAL exchanges code for access token  
6. Token stored in browser session storage  
7. Subsequent API calls include Bearer token

#### **5.1.2 Token Management**

**Storage**: Browser sessionStorage (not localStorage for security) **Scope**: {clientId}/.default (application-specific scope) **Refresh**: Automatic silent refresh via MSAL **Expiration**: Handled by MSAL library

### **5.2 API Security**

#### **5.2.1 Authorization Validation**

// Every API call validates authorization header  
const authHeader = request.headers.get('authorization');  
if (!authHeader) {  
 return NextResponse.json(  
 { error: 'No authorization token provided' },  
 { status: 401 }  
 );  
}

#### **5.2.2 Cross-Origin Considerations**

* Same-origin policy for API routes
* CORS handled by Next.js defaults
* No sensitive data in client-side code

## **6. Data Models**

### **6.1 Frontend Data Models**

#### **6.1.1 Message Interface**

interface Message {  
 role: 'user' | 'assistant';  
 content: string;  
}

#### **6.1.2 MSAL Account Interface**

// Provided by @azure/msal-browser  
interface AccountInfo {  
 homeAccountId: string;  
 environment: string;  
 tenantId: string;  
 username: string;  
 localAccountId: string;  
 name?: string;  
}

**6.2 API Data Models**

#### **6.2.1 API Request Models**

interface CompletionRequest {  
 prompt: string;  
}  
  
interface Lab45Request {  
 messages: Array<{  
 role: string;  
 content: string;  
 }>;  
 skill\_parameters: {  
 model\_name: string;  
 max\_output\_tokens: number;  
 temperature: number;  
 };  
}

#### **6.2.2 API Response Models**

interface CompletionResponse {  
 data: {  
 content: string;  
 };  
}  
  
interface ErrorResponse {  
 error: string;  
}

## **7. State Management**

### **7.1 Component State**

#### **7.1.1 ChatInterface State**

// Message history  
const [messages, setMessages] = useState<Message[]>([]);  
  
// Current user input  
const [input, setInput] = useState<string>("");  
  
// Loading state for API calls  
const [isLoading, setIsLoading] = useState<boolean>(false);

#### **7.1.2 Authentication State**

// Managed by MSAL Provider  
interface MSALState {  
 instance: IPublicClientApplication;  
 accounts: AccountInfo[];  
 inProgress: InteractionStatus;  
}  
  
// Custom hook state  
const [accessToken, setAccessToken] = useState<string | null>(null);

### **7.2 State Flow**

#### **7.2.1 Authentication State Flow**

Initial Load → MSAL Initialize → Check Accounts → Acquire Token → Set Token State

**7.2.2 Chat State Flow**

User Input → Add User Message → API Call → Add AI Response → Update UI

## **8. Error Handling**

### **8.1 Frontend Error Handling**

#### **8.1.1 Authentication Errors**

// Silent token acquisition fallback  
try {  
 const response = await instance.acquireTokenSilent(request);  
} catch (error) {  
 // Fallback to popup  
 const response = await instance.acquireTokenPopup(request);  
}

**8.1.2 API Call Errors**

try {  
 const response = await fetch('/api/completions', config);  
 if (!response.ok) {  
 throw new Error(data.error || 'Failed to get response');  
 }  
} catch (error) {  
 // Show error message in chat  
 setMessages(prev => [...prev, {   
 role: 'assistant',   
 content: `Error: ${error.message}`   
 }]);  
}

### **8.2 Backend Error Handling**

#### **8.2.1 API Route Error Handling**

// Authorization validation  
if (!authHeader) {  
 return NextResponse.json(  
 { error: 'No authorization token provided' },  
 { status: 401 }  
 );  
}  
  
// External API errors  
if (!response.ok) {  
 return NextResponse.json(  
 { error: 'API request failed' },  
 { status: response.status }  
 );  
}  
  
// General error catching  
catch (error) {  
 console.error('API Error:', error);  
 return NextResponse.json(  
 { error: 'Internal server error' },  
 { status: 500 }  
 );  
}

## **9. Performance Considerations**

### **9.1 Frontend Performance**

#### **9.1.1 React Optimizations**

* **Component Memoization**: Consider React.memo for stable components
* **State Updates**: Functional updates to prevent unnecessary re-renders
* **Key Props**: Proper key usage in message list rendering

#### **9.1.2 Token Management**

* **Silent Refresh**: MSAL handles token refresh without user interaction
* **Session Storage**: Faster than localStorage, cleared on tab close

### **9.2 API Performance**

#### **9.2.1 Streaming Response Handling**

// Efficient streaming processing  
const reader = response.body?.getReader();  
while (true) {  
 const { done, value } = await reader.read();  
 if (done) break;  
 // Process chunks incrementally  
}

#### **9.2.2 Error Response Optimization**

* Early return for authorization failures
* Minimal error object creation
* Proper HTTP status codes

## **10. Deployment Configuration**

### **10.1 Environment Variables**

// Required for production deployment  
NEXT\_PUBLIC\_MSAL\_CLIENT\_ID=a919164d-8b7c-43fb-8119-f1997d45ca4f  
NEXT\_PUBLIC\_MSAL\_AUTHORITY=https://login.microsoftonline.com/258ac4e4-146a-411e-9dc8-79a9e12fd6da

### **10.2 Build Configuration**

#### **10.2.1 Next.js Configuration**

**File**: next.config.ts

* TypeScript support enabled
* API routes configured
* Static optimization for client components

#### **10.2.2 Build Scripts**

{  
 "scripts": {  
 "dev": "next dev",  
 "build": "next build",  
 "start": "next start",  
 "lint": "next lint"  
 }  
}

## **11. Testing Strategy**

### **11.1 Unit Testing (Recommended)**

* **Components**: Test rendering and user interactions
* **Hooks**: Test token acquisition logic
* **API Routes**: Test request/response handling

### **11.2 Integration Testing (Recommended)**

* **Authentication Flow**: End-to-end login/logout
* **Chat Flow**: Complete user message to AI response
* **Error Scenarios**: Network failures, invalid tokens

### **11.3 Manual Testing Checklist**

* Microsoft login/logout functionality
* Chat message sending and receiving
* Error handling for network issues
* Responsive design on mobile devices
* Token refresh scenarios

## **12. Future Enhancements**

### **12.1 Potential Features**

* **Message History Persistence**: Store chat history in database
* **Multiple Conversations**: Support for separate chat threads
* **File Uploads**: Support for document analysis
* **Real-time Updates**: WebSocket integration for live responses

### **12.2 Technical Improvements**

* **Caching**: Implement response caching for repeated queries
* **Offline Support**: Service worker for offline functionality
* **Performance Monitoring**: Add application insights
* **Accessibility**: ARIA labels and keyboard navigation

## **13. Dependencies and Versions**

### **13.1 Core Dependencies**

{  
 "@azure/msal-browser": "^3.28.0",  
 "@azure/msal-react": "^2.2.0",  
 "next": "15.1.0",  
 "react": "^18.0.0",  
 "react-dom": "^18.0.0"  
}

### **13.2 Development Dependencies**

{  
 "@types/node": "^20",  
 "@types/react": "^19",  
 "@types/react-dom": "^19",  
 "postcss": "^8",  
 "tailwindcss": "^3.4.1",  
 "typescript": "^5"  
}

**14. Conclusion**

This LLD document provides a comprehensive technical overview of the MSAL Chat Application. The architecture follows modern React patterns with TypeScript for type safety, integrates securely with Microsoft Azure AD for authentication, and provides a clean chat interface for AI interactions.

The modular component design allows for easy maintenance and future enhancements, while the security-first approach ensures proper handling of authentication tokens and API communications.

For any questions or clarifications regarding this design document, please refer to the inline code comments or reach out to the development team.