Microsoft Graph API Calendar Integration Guide

This guide will walk you through implementing Microsoft Graph API for calendar integration with your ITILITI website's consultation modal.

Overview

The implementation consists of three main components:

1. **Backend Service**: Creates calendar events and manages Microsoft Graph authentication
2. **API Endpoints**: Handles requests from the frontend
3. **Frontend Component**: Enhanced consultation modal that connects to the API

Step 1: Set Up Microsoft 365 App Registration

First, you need to register an application in Azure AD:

1. Go to the Azure Portal
2. Navigate to **Azure Active Directory** > **App registrations** > **New registration**
3. Enter a name like "ITILITI Calendar Integration"
4. Select "Web" as the platform type
5. Set the redirect URI to your backend API URL (e.g., `https://api.itiliti.io/auth/callback`)
6. Click "Register"

Configure Permissions

1. In your app registration, go to **API permissions**
2. Click "Add a permission"
3. Select "Microsoft Graph"
4. Choose "Application permissions" (for backend service)
5. Add:
   - `Calendars.ReadWrite` (to manage calendars)
   - `User.Read.All` (to access user information)
6. Click "Grant admin consent"

Create a Client Secret

1. Go to **Certificates & secrets**

2. Click "New client secret"
3. Add a description and choose expiration (recommend 1 year for rotation)
4. Click "Add"
5. **IMPORTANT**: Copy the secret value immediately - you won't see it again!

Step 2: Install Required Packages

For your Node.js backend:

```bash
npm install @microsoft/microsoft-graph-client @azure/identity
```

For your React frontend:

```bash
npm install framer-motion
```

Step 3: Implement the Backend

1. Create Microsoft Calendar Service

Create a file named `MicrosoftCalendarService.js` in your services directory with the provided code. This service handles:

- Authenticating with Microsoft Graph
- Creating calendar events with Teams meetings
- Sending meeting invites to attendees
- Getting available time slots

2. Set Up API Endpoints

Create an API controller for consultation scheduling with two endpoints:

- `POST /api/schedule-consultation`: Schedule a new consultation
- `GET /api/available-time-slots/:date`: Get available time slots for a date

3. Configure Environment Variables

Add these environment variables to your backend:

```
MS_GRAPH_TENANT_ID=your-tenant-id
MS_GRAPH_CLIENT_ID=your-client-id
MS_GRAPH_CLIENT_SECRET=your-client-secret
MS_GRAPH_CALENDAR_USER=sales@itiliti.io
```

Step 4: Integrate with the Frontend

1. Update ConsultationModal Component

Replace your current ConsultationModal with the enhanced version that includes:

- Three-step wizard interface
- Dynamic date and time selection
- Available time slots from the calendar
- Improved error handling and loading states
- Animated transitions with framer-motion

2. Update ServiceSolutions.js

Update your ServiceSolutions.js file to use the enhanced consultation modal:

```javascript
import EnhancedConsultationModal from '../components/enhanced/EnhancedConsultationModal';

// Inside your component:
const [isModalOpen, setIsModalOpen] = useState(false);

// In the JSX:
<motion.button
  onClick={() => setIsModalOpen(true)}
  className="bg-white text-blue-800 px-8 py-3 rounded-lg font-medium hover:bg-blue-50 transitic
  whileHover={{ scale: 1.05 }}
  whileTap={{ scale: 0.98 }}
>
  <Calendar className="w-5 h-5 mr-2" />
  Schedule a Consultation
  <ArrowRight className="ml-2 w-4 h-4" />
</motion.button>

{/* Add this at the end of your component */}
<EnhancedConsultationModal
  isOpen={isModalOpen}
  onClose={() => setIsModalOpen(false)}
/>
```

Step 5: Testing

1. Test the Backend Integration

Create a simple test script to verify your Microsoft Graph integration:

```javascript
// test-calendar.js
const calendarService = require('./services/MicrosoftCalendarService');

async function test() {
  try {
    await calendarService.initialize({
      tenantId: process.env.MS_GRAPH_TENANT_ID,
      clientId: process.env.MS_GRAPH_CLIENT_ID,
      clientSecret: process.env.MS_GRAPH_CLIENT_SECRET,
      calendarUser: 'sales@itiliti.io'
    });

    // Create a test meeting
    const result = await calendarService.createMeeting({
      name: 'Test User',
      email: 'your-email@example.com', // Use your email for testing
      company: 'Test Company',
      date: '2025-05-01',
      time: '10:00 AM',
      notes: 'This is a test meeting'
    });

    console.log('Meeting created:', result);
    console.log('Join URL:', result.onlineMeeting?.joinUrl);
  } catch (err) {
    console.error('Test failed:', err);
  }
}

test();
```

Run with:

```bash
node test-calendar.js
```

2. Test the Frontend Integration

1. Start your backend server
2. Run your frontend application

3. Click the "Schedule a Consultation" button in the ServiceSolutions component
4. Fill out the form and test the workflow

Troubleshooting

Common Microsoft Graph Issues

1. **Authentication Errors**:
   - Check your tenant ID, client ID, and client secret
   - Ensure admin consent was granted for the permissions
   - Verify the account has appropriate licenses
2. **Calendar Creation Errors**:
   - Check the format of your date/time strings
   - Ensure the calendar user exists and has a mailbox
   - Verify Teams is enabled for online meetings
3. **Connection Issues**:
   - Check network connectivity to Microsoft Graph API
   - Verify your firewall allows outbound connections to `graph.microsoft.com`

Frontend Integration Issues

1. **API Connection Errors**:
   - Check the API endpoint URLs
   - Verify CORS is properly configured
   - Look for network errors in the browser console
2. **Form Submission Issues**:
   - Check form validation
   - Ensure all required fields are being sent
   - Look for JSON parsing errors

Security Considerations

1. **Protect Your Credentials**:
   - Never expose Microsoft credentials in frontend code
   - Store secrets securely (use env variables, secrets manager)
   - Rotate client secrets periodically

2. **Input Validation**:
    - Validate all user inputs on both client and server
    - Sanitize data before sending to Microsoft Graph
3. **Error Handling**:
    - Don't expose sensitive information in error messages
    - Log errors for troubleshooting but sanitize logs

Next Steps

1. **Implement Logging**: Add detailed logging for troubleshooting
2. **Add Monitoring**: Monitor API usage and errors
3. **Create Fallback**: Implement a fallback method using email with iCalendar attachment
4. **Enhance UX**: Add confirmation emails and reminders

By following this guide, you'll have a robust Microsoft Graph calendar integration that provides a professional scheduling experience for your users.