Apps

Notifications

Learn how to send notifications to OMI users from your applications, including direct text notifications and best practices for implementation.

Types of Notifications 📬

1. 📱 Direct Text Notifications

Direct text notifications allow you to send immediate messages to specific OMI users. This is useful for alerts, updates, or responses to user actions.

Example Use Cases

- Send task reminders and event notifications
- Notify users about service updates or changes
- Deliver real-time alerts and warnings
- Respond to user queries or actions
- Announce new features or important changes

Implementing Notifications 🛠️

Step 1: Set Up Authentication 🔑

Before sending notifications, you'll need:

1. Your OMI App ID (`app_id`)
2. Your OMI App Secret (API Key)

Store these securely as environment variables:

```
OMI_APP_ID=your_app_id_here
OMI_APP_SECRET=your_app_secret_here
```

Step 2: Configure Your Endpoint 🔌

Base URL and Endpoint

```
* **Method:** `POST`
* **URL:** `/v2/integrations/{app_id}/notification`
* **Base URL:** `api.omi.me`
```

Required Headers

```
* **Authorization:** `Bearer <YOUR_APP_SECRET>`
* **Content-Type:** `application/json`
* **Content-Length:** `0`
```

Query Parameters

```
* `uid` (string, **required**): The target user's OMI ID
* `message` (string, **required**): The notification text
```

Step 3: Implement the Code 💻

Here's a complete Node.js implementation:

```
const https = require('https');

/**
 * Sends a direct notification to an Omi user.
 * @param {string} userId - The Omi user's unique ID
 * @param {string} message - The notification text
 * @returns {Promise<object>} Response data or error
 */
function sendOmiNotification(userId, message) {
    const appId = process.env.OMI_APP_ID;
    const appSecret = process.env.OMI_APP_SECRET;

    if (!appId) throw new Error("OMI_APP_ID not set");
    if (!appSecret) throw new Error("OMI_APP_SECRET not set");

    const options = {
        hostname: 'api.omi.me',
        path: `/v2/integrations/${appId}/notification?uid=${encodeURIComponent(userId)}&message=${enc
        method: 'POST',
        headers: {
            'Authorization': `Bearer ${appSecret}`,
            'Content-Type': 'application/json',
            'Content-Length': 0
        }
    };

    return new Promise((resolve, reject) => {
        const req = https.request(options, (res) => {
            let data = '';
            res.on('data', chunk => data += chunk);
            res.on('end', () => {
                if (res.statusCode >= 200 && res.statusCode < 300) {
                    try {
                        resolve(data ? JSON.parse(data) : {});
                    } catch (e) {
                        resolve({ raw: data });
                    }
                } else {
                    reject(new Error(`API Error (${res.statusCode}): ${data}`));
                }
            });
        });
        req.on('error', reject);
        req.end();
    });
}
```

Step 4: Test Your Implementation 🧪

1. Set up your environment variables:

```
export OMI_APP_ID="your_app_id"
export OMI_APP_SECRET="your_app_secret"
```

2. Test with a sample notification:

```
sendOmiNotification("user_id_here", "Test notification!")
    .then(response => console.log("Success:", response))
    .catch(error => console.error("Error:", error));
```

3. Verify the notification appears in the user's OMI app

Best Practices 🎯

1. **Rate Limiting**
   - Implement reasonable delays between notifications
   - Avoid sending duplicate notifications
   - Group related notifications when possible
2. **Content Guidelines**
   - Keep messages concise and clear
   - Include relevant context
   - Use appropriate urgency levels
3. **Error Handling**
   - Implement retry logic for failed attempts
   - Log errors for debugging
   - Monitor notification delivery status
4. **Security**
   - Store API credentials securely
   - Validate user IDs before sending
   - Implement request timeouts

Troubleshooting 🔍

Common Issues

1. **Authentication Errors**
   - Verify your API credentials
   - Check the Bearer token format
   - Ensure environment variables are set
2. **Delivery Issues**
   - Validate the user ID exists
   - Check message encoding
   - Verify network connectivity
3. **Rate Limiting**
   - Monitor API response headers
   - Implement exponential backoff
   - Track notification frequency

Error Response Codes

Status Code	Meaning	Action
401	Unauthorized	Check API credentials
404	User not found	Verify user ID
429	Too many requests	Implement rate limiting
500	Server error	Retry with backoff

Example Implementations 💡

1. Task Reminder

```
function sendTaskReminder(userId, taskName, dueDate) {
    const message = `Reminder: "${taskName}" is due ${dueDate}`;
    return sendOmiNotification(userId, message);
}
```

2. Service Update

```
function sendServiceUpdate(userId, serviceName, status) {
    const message = `${serviceName} status: ${status}`;
    return sendOmiNotification(userId, message);
}
```

Need Help? 🤝

- Check our **API Reference**
- Join our **Discord community**
- Contact **support**

‹ OMI OAuth 2.0 Integration

App Setup ›