Backend API Endpoints

Overview

This document outlines all necessary backend endpoints for managing OAuth connections, tokens, and automation workflows.

---

1. Authentication & Authorization Endpoints

1.1 Initiate OAuth Flow

Endpoint: `GET /api/oauth/authorize`

Purpose: Generate authorization URL for OAuth flow

Query Parameters:

- `userId` (required): User identifier
- `connectorId` (required): Connector identifier (e.g., 'gmail', 'linkedin')
- `scopes` (required): Space-separated list of requested scopes
- `redirectUri` (optional): Custom redirect URI

Response:

```
{
  "authUrl": "https://accounts.google.com/o/oauth2/v2/auth?...",
  "state": "random_state_token",
  "connectorId": "gmail"
}
```

Error Cases:

- 400: Invalid connector or missing parameters
- 500: Failed to generate auth URL

1.2 OAuth Callback Handler

Endpoint: `GET /api/oauth/callback`

Purpose: Handle OAuth provider callback and exchange code for tokens

Query Parameters:

- `code` (required): Authorization code from provider
- `state` (required): State token for CSRF protection
- `connectorId` (required): Connector identifier

Process:

1. Validate state token
2. Exchange code for access/refresh tokens
3. Store tokens in `oauth_connections` table
4. Retrieve and store connector-specific metadata
5. Return success page or redirect

Response:

```
<!-- Success page that closes the popup window -->
<script>
  window.opener.postMessage({ success: true, connectorId: 'gmail' }, '*');
  window.close();
</script>
```

Database Operations:

```
INSERT INTO oauth_connections (
  user_id, connector_id, access_token, refresh_token,
  token_type, expires_at, scopes, status,
  connector_metadata, granted_at
) VALUES (...);
```

---

2. Connection Management Endpoints

2.1 Check Connection Status

Endpoint: `GET /api/oauth/connections/:userId/:connectorId`

Purpose: Check if user has active connection to a connector

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Response:

```
{
  "connected": true,
  "connectorId": "gmail",
  "status": "active",
  "grantedScopes": [
    "openid",
    "email",
    "profile",
    "https://www.googleapis.com/auth/gmail.send"
  ],
  "grantedAt": "2025-01-10T12:00:00Z",
  "lastUsedAt": "2025-01-12T08:30:00Z",
  "expiresAt": "2025-01-12T13:00:00Z",
  "metadata": {
    "email": "user@gmail.com",
    "accountId": "123456789"
  }
}
```

Response (Not Connected):

```
{
  "connected": false,
  "connectorId": "gmail"
}
```

2.2 List All User Connections

Endpoint: `GET /api/oauth/connections/:userId`

Purpose: Get all active connections for a user

Path Parameters:

- `userId`: User identifier

Query Parameters:

- `status` (optional): Filter by status (active, expired, revoked, error)
- `includeMetadata` (optional): Include connector metadata (default: false)

Response:

```
{
  "userId": "user_123",
  "connections": [
    {
      "connectorId": "gmail",
      "status": "active",
      "grantedScopes": ["openid", "email", "profile", "..."],
      "grantedAt": "2025-01-10T12:00:00Z",
      "lastUsedAt": "2025-01-12T08:30:00Z",
      "expiresAt": "2025-01-12T13:00:00Z"
    },
    {
      "connectorId": "linkedin",
      "status": "active",
      "grantedScopes": ["openid", "profile", "email"],
      "grantedAt": "2025-01-11T14:00:00Z",
      "lastUsedAt": "2025-01-12T09:00:00Z",
      "expiresAt": "2025-01-12T14:00:00Z"
    }
  ],
```

```
  "total": 2
}
```

2.3 Revoke Connection

Endpoint: `DELETE /api/oauth/connections/:userId/:connectorId`

Purpose: Revoke OAuth connection and optionally revoke from provider

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Query Parameters:

- `revokeFromProvider` (optional): Revoke token from OAuth provider (default: true)

Process:

1. Update status to 'revoked' in database
2. Optionally call provider's revocation endpoint
3. Clean up associated data

Response:

```
{
  "success": true,
  "connectorId": "gmail",
  "revokedAt": "2025-01-12T10:00:00Z"
}
```

2.4 Token Update Workflow (Scope Upgrade)

Endpoint: `PATCH /api/oauth/connections/:userId/:connectorId/scopes`

Path Parameters:

- `userId` – User identifier
- `connectorId` – Connector identifier (e.g. `gmail`, `linkedin`, `facebook`)

Request Body:

```
{
  "additionalScopes": [
    "https://www.googleapis.com/auth/calendar",
    "https://www.googleapis.com/auth/drive.file"
  ]
}
```

---

3. Token Management Endpoints

3.1 Get Valid Access Token

Endpoint: `GET /api/oauth/token/:userId/:connectorId`

Purpose: Get a valid access token, refreshing if necessary

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Process:

1. Retrieve connection from database
2. Check if token is expired
3. If expired, refresh the token
4. Update `last_used_at` timestamp
5. Return valid token

Response:

```
{
  "accessToken": "ya29.a0AfH6SMBx...",
  "tokenType": "Bearer",
  "expiresAt": "2025-01-12T13:00:00Z",
```

```
  "scopes": ["openid", "email", "profile", "..."]
}
```

Error Cases:

- 404: Connection not found
- 401: Connection revoked or expired (refresh failed)
- 500: Token refresh failed

3.2 Force Refresh Token

Endpoint: `POST /api/oauth/token/:userId/:connectorId/refresh`

Purpose: Manually trigger token refresh

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Process:

1. Retrieve refresh token from database
2. Call provider's token refresh endpoint
3. Update `oauth_connections` table with new tokens
4. Log refresh attempt in `oauth_refresh_log`

Response:

```
{
  "success": true,
  "accessToken": "ya29.a0AfH6SMBx...",
  "expiresAt": "2025-01-12T13:00:00Z",
  "refreshedAt": "2025-01-12T12:00:00Z"
}
```

Database Operations:

```
-- Update tokens
UPDATE oauth_connections
SET access_token = ?,
    expires_at = ?,
    last_refresh_at = NOW(),
    status = 'active'
WHERE user_id = ? AND connector_id = ?;

-- Log refresh
INSERT INTO oauth_refresh_log (connection_id, success, refresh_attempted_a
t)
VALUES (?, true, NOW());
```

3.3 Batch Token Refresh (Background Job)

Endpoint: `POST /api/oauth/token/refresh-batch`

Purpose: Background job to refresh expiring tokens

Request Body:

```
{
  "expiresWithinMinutes": 30,
  "limit": 100
}
```

Process:

1. Find connections expiring within specified time
2. Refresh tokens in batch
3. Update database
4. Log results

Query:

```
SELECT * FROM oauth_connections
WHERE status = 'active'
  AND expires_at < DATE_ADD(NOW(), INTERVAL ? MINUTE)
  AND refresh_token IS NOT NULL
LIMIT ?;
```

Response:

```
{
  "processed": 15,
  "successful": 14,
  "failed": 1,
  "failures": [
    {
      "userId": "user_456",
      "connectorId": "linkedin",
      "error": "Invalid refresh token"
    }
  ]
}
```

---

4. Scope Management Endpoints

4.1 Update Connection Scopes

Endpoint: `PATCH /api/oauth/connections/:userId/:connectorId/scopes`

Purpose: Add additional scopes to existing connection (requires re-auth)

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Request Body:

```
{
  "additionalScopes": [
    "https://www.googleapis.com/auth/calendar",
    "https://www.googleapis.com/auth/drive.file"
  ]
}
```

Response:

```
{
  "requiresReauth": true,
  "authUrl": "https://accounts.google.com/o/oauth2/v2/auth?...",
  "message": "Additional scopes require re-authorization"
}
```

4.2 Check Granted Scopes

Endpoint: `GET /api/oauth/connections/:userId/:connectorId/scopes`

Purpose: Check if user has granted specific scopes

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Query Parameters:

- `requiredScopes`: Comma-separated list of required scopes

Response:

```
{
  "connectorId": "gmail",
  "grantedScopes": [
    "openid",
    "email",
    "profile",
```

```
    "https://www.googleapis.com/auth/gmail.send"
  ],
  "requiredScopes": [
    "https://www.googleapis.com/auth/gmail.send",
    "https://www.googleapis.com/auth/calendar"
  ],
  "hasAllRequired": false,
  "missingScopes": [
    "https://www.googleapis.com/auth/calendar"
  ]
}
```

---

5. Health & Monitoring Endpoints

5.1 Connection Health Check

Endpoint: `GET /api/oauth/health/:userId/:connectorId`

Purpose: Verify connection is working by making test API call

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Process:

1. Get valid token
2. Make test API call to provider
3. Update status based on response

Response:

```
{
  "healthy": true,
  "connectorId": "gmail",
  "lastChecked": "2025-01-12T10:00:00Z",
  "status": "active",
```

```
  "testEndpoint": "https://gmail.googleapis.com/gmail/v1/users/me/profile"
}
```

Error Response:

```
{
  "healthy": false,
  "connectorId": "gmail",
  "status": "error",
  "error": "Token invalid or expired",
  "lastChecked": "2025-01-12T10:00:00Z"
}
```

5.2 Get Connection Analytics

Endpoint: `GET /api/oauth/analytics/:userId`

Purpose: Get usage statistics for user's connections

Path Parameters:

- `userId`: User identifier

Query Parameters:

- `connectorId` (optional): Filter by specific connector
- `days` (optional): Days of history (default: 30)

Response:

```
{
  "userId": "user_123",
  "period": "30days",
  "connections": [
    {
      "connectorId": "gmail",
      "totalRequests": 1543,
      "lastUsedAt": "2025-01-12T08:30:00Z",
      "totalRefreshes": 12,
```

```
    "averageTokenLifetime": "3600s",
    "status": "active"
   }
  ]
}
```

5.3 Get Refresh History

Endpoint: `GET /api/oauth/refresh-history/:userId/:connectorId`

Purpose: Get token refresh history for debugging

Path Parameters:

- `userId`: User identifier
- `connectorId`: Connector identifier

Query Parameters:

- `limit` (optional): Number of records (default: 50)
- `offset` (optional): Pagination offset

Response:

```
{
 "connectorId": "gmail",
 "history": [
  {
   "refreshedAt": "2025-01-12T09:00:00Z",
   "success": true,
   "errorMessage": null
  },
  {
   "refreshedAt": "2025-01-12T06:00:00Z",
   "success": false,
   "errorMessage": "Network timeout"
  }
 ],
```

```
  "total": 145,
  "limit": 50,
  "offset": 0
}
```

6. Webhook/Notification Endpoints

6.1 Provider Webhook Handler

Endpoint: `POST /api/oauth/webhook/:connectorId`

Purpose: Handle webhooks from OAuth providers (token revocation, etc.)

Path Parameters:

- `connectorId`: Connector identifier

Request Body: (Varies by provider)

```
{
  "event": "token.revoked",
  "accountId": "123456789",
  "timestamp": "2025-01-12T10:00:00Z"
}
```

Process:

1. Verify webhook signature
2. Update connection status
3. Notify affected users

7. Admin/System Endpoints

7.1 Get System-Wide Connection Stats

Endpoint: `GET /api/oauth/admin/stats`

Purpose: Get platform-wide OAuth statistics

Query Parameters:

- `days` (optional): Days of history (default: 7)

Response:

```
{
  "totalConnections": 5432,
  "activeConnections": 5123,
  "expiredConnections": 234,
  "revokedConnections": 75,
  "byConnector": {
    "gmail": 1234,
    "linkedin": 987,
    "facebook": 876
  },
  "refreshSuccessRate": 98.5,
  "averageTokenLifetime": "3600s"
}
```

7.2 Cleanup Expired Connections

Endpoint: `POST /api/oauth/admin/cleanup`

Purpose: Remove old expired/revoked connections

Request Body:

```
{
  "olderThanDays": 90,
  "status": ["expired", "revoked"],
  "dryRun": false
}
```

Response:

```
{
  "deleted": 142,
```

```
  "dryRun": false,
  "deletedByStatus": {
    "expired": 89,
    "revoked": 53
  }
}
```

API Design Best Practices

Authentication

All endpoints should require authentication via:

- JWT tokens in `Authorization` header
- API keys for system/background jobs
- Webhook signature verification for provider webhooks

Rate Limiting

Implement rate limiting:

- Per user: 100 requests/minute
- Per IP: 1000 requests/minute
- Token refresh: 1 request per connection every 5 minutes

Error Handling

Standard error response format:

```
{
  "error": {
    "code": "TOKEN_EXPIRED",
    "message": "The access token has expired",
    "details": {
      "connectorId": "gmail",
      "userId": "user_123",
```

```
    "expiredAt": "2025-01-12T10:00:00Z"
   }
  }
 }
```

Logging

Log all token operations:

- Token grants
- Token refreshes (success/failure)
- Connection revocations
- Failed API calls

Security Considerations

1. **Encrypt tokens at rest** in database
2. **Use HTTPS** for all endpoints
3. **Implement CSRF protection** for OAuth flows
4. **Validate state tokens** in OAuth callbacks
5. **Rate limit token refresh** attempts
6. **Monitor for suspicious activity** (rapid token refreshes)
7. **Implement token rotation** where supported
8. **Store minimal user data** in connector_metadata

Background Jobs

Job 1: Token Refresh Job

Schedule: Every 15 minutes
Endpoint: `POST /api/oauth/token/refresh-batch` **Purpose:** Proactively refresh tokens expiring within 30 minutes

Job 2: Connection Health Check

Schedule: Every hour
Purpose: Verify active connections are still valid

Job 3: Cleanup Job

Schedule: Daily at 2 AM
Purpose: Remove expired connections older than 90 days

Job 4: Analytics Aggregation

Schedule: Daily at 3 AM
Purpose: Aggregate usage statistics for reporting