PUPT HRIS OAuth 2.0 Integration Guide

Overview

The PUPT Human Resource Information System (HRIS) provides OAuth 2.0 authentication services that allow third-party applications to access faculty data securely. This guide explains how to integrate your application with PUPT HRIS's OAuth service.

Prerequisites

Before you begin integration, you need to:

1. Register your application with PUPT HRIS to obtain:
    - Client ID
    - Client Secret
2. Provide your application's redirect URI(s)
3. Ensure your application can make HTTPS requests to the HRIS OAuth endpoints

OAuth Flow

PUPT HRIS implements the OAuth 2.0 Authorization Code flow, which is suitable for server-side applications. The flow consists of the following steps:

1. Direct the user to the authorization endpoint
2. User logs in and grants consent
3. Receive the authorization code
4. Exchange the code for an access token
5. Use the access token to access faculty data

Endpoints

Base URL: `https://pupt-hris.site/api`

Endpoint	Method	Description
`/oauth/authorize`	GET/POST	Authorization endpoint
`/oauth/login`	POST	User login endpoint
`/oauth/token`	POST	Token exchange endpoint
`/oauth/validate`	GET	Token validation endpoint
`/oauth/client/:clientId`	GET	Client information endpoint

Step-by-Step Integration

1. Authorization Request

Direct the user to the authorization endpoint:

```
GET /oauth/authorize
```

Required Parameters:

- `client_id` : Your application's client ID
- `redirect_uri` : Your application's callback URL
- `response_type` : Must be "code"
- `state` : A random string to prevent CSRF attacks

Example:

```
https://hris.example.com/oauth/authorize?
  client_id=your_client_id&
  redirect_uri=https://your-app.com/callback&
  response_type=code&
  state=random_state_string
```

2. User Authentication

The user will be presented with the HRIS login page. After successful login, they will be asked to grant consent for your application to access their faculty data.

3. Authorization Code

If the user grants consent, they will be redirected to your redirect_uri with:

- `code` : The authorization code
- `state` : The same state value you provided

Example:

```
https://your-app.com/callback?code=authorization_code&state=random_state_string
```

4. Token Exchange

Exchange the authorization code for an access token:

```
POST /oauth/token
Content-Type: application/json

{
  "grant_type": "authorization_code",
  "code": "authorization_code",
  "redirect_uri": "https://your-app.com/callback",
  "client_id": "your_client_id",
  "client_secret": "your_client_secret"
}
```

Response:

```
{
  "access_token": "jwt_token",
  "token_type": "Bearer",
  "expires_in": 86400,
  "faculty_data": {
    "sub": "user_id",
    "UserID": "faculty_id",
    "Fcode": "faculty_code",
    "FirstName": "first_name",
    "Surname": "surname",
    "MiddleName": "middle_name",
    "NameExtension": "name_extension",
    "Email": "email",
    "EmploymentType": "employment_type",
    "isActive": true
  }
}
```

5. Using the Access Token

Include the access token in the Authorization header for API requests:

```
GET /api/endpoint
Authorization: Bearer your_access_token
```

6. Token Validation

To validate an access token:

```
GET /oauth/validate
Authorization: Bearer your_access_token
```

Security Considerations

1. Always use HTTPS in production
2. Keep your client secret secure
3. Validate the state parameter
4. Store tokens securely
5. Implement token refresh mechanism
6. Handle token expiration (tokens expire in 24 hours)

Rate Limiting

The OAuth endpoints are rate-limited to 100 requests per 15 minutes per IP address.

Error Handling

Common error responses:

```
{
  "error": "invalid_request",
  "error_description": "Missing required parameter"
}
```

Error Types:

- `invalid_request` : Missing or invalid parameters
- `invalid_client` : Invalid client credentials
- `invalid_grant` : Invalid or expired authorization code
- `invalid_token` : Invalid or expired access token
- `unauthorized_client` : Client not authorized for this grant type
- `server_error` : Internal server error

Faculty Data Fields

The access token payload includes the following faculty data fields:

Field	Description
UserID	Unique identifier for the faculty
Fcode	Faculty code
FirstName	Faculty's first name
Surname	Faculty's surname
MiddleName	Faculty's middle name
NameExtension	Name extension (Jr., Sr., etc.)
Email	Faculty's email address
EmploymentType	Type of employment (Full-Time, Part-Time, etc.)
isActive	Faculty's active status

Example Implementation

Here's a basic Node.js example using axios:

```
const axios = require("axios");

// Step 1: Get authorization code (redirect user to authorization URL)
const authUrl = `${baseUrl}/oauth/authorize?
  client_id=${clientId}&
  redirect_uri=${redirectUri}&
  response_type=code&
  state=${state}`;

// Step 2: Exchange code for token
async function getToken(code) {
  try {
    const response = await axios.post(`${baseUrl}/oauth/token`, {
      grant_type: "authorization_code",
      code,
      redirect_uri: redirectUri,
      client_id: clientId,
      client_secret: clientSecret,
    });
    return response.data;
  } catch (error) {
    console.error("Token exchange failed:", error.response.data);
    throw error;
  }
}

// Step 3: Use the token
async function getFacultyData(accessToken) {
  try {
    const response = await axios.get(`${baseUrl}/api/faculty`, {
      headers: {
        Authorization: `Bearer ${accessToken}`,
      },
    });
    return response.data;
  } catch (error) {
    console.error("API request failed:", error.response.data);
    throw error;
  }
}
```

Support

For integration support or to report issues, please contact the PUPT HRIS team.

Changelog

- **v1.0.0** (Initial Release)
  - Authorization Code flow implementation
  - Faculty data access
  - Token validation endpoints
  - Rate limiting