Clinion SSO Integration Guide

Overview

The Clinion SSO (Single Sign-On) endpoint allows users to authenticate with a central identity provider and be redirected back to the client application (such as a web app) after successful authentication. This guide explains how to integrate and use the SSO endpoint for your application.

SSO Endpoint

Endpoint URL:
`https://accounts1.clinion.com/sso`

Parameters

- **customer**: Specifies the customer using the SSO service. Each customer is identified with a unique code. Example: `SMA`.
- **callback**: The URL to which the user will be redirected after successful authentication. This is typically your application's URL where you want users to land after logging in.

Example Request

bash
Copy code

```
GET https://accounts1.clinion.com/sso?customer=SMA&callback=http://localhost:4200
```

Request Parameters

1. **Customer**
    - **Type**: String
    - **Required**: Yes
    - **Description**: The unique identifier for the customer. Replace `SMA` with the appropriate customer code.

Example:
bash
Copy code

```
customer=SMA
```

2.
3. **Callback URL**
    - **Type**: URL
    - **Required**: Yes
    - **Description**: The URL where the user will be redirected after successful login. This must be a valid URL within your application.

Example:
bash
Copy code
`callback=http://localhost:4200`

4.

Response Flow

1. **User Redirection**: When the client sends a request to the SSO endpoint with the necessary parameters, the user will be redirected to the login page of the SSO provider.
2. **Authentication**: The user logs in with their credentials on the Clinion SSO page.
3. **Callback**: After successful authentication, the user is redirected to the callback URL provided in the request. This redirection may include a token or other authentication data in the query string or headers (depending on the SSO setup).

Example Workflow

Request:
Your application initiates an SSO login request with the customer and callback parameters.
bash
Copy code
`GET https://accounts1.clinion.com/sso?customer=YOUR_CUSTOMER_CODE&callback=https://yourapp.com/home`

1.
2. **User Authentication**:
   The user is redirected to Clinion's login page, where they enter their credentials.
3. **Redirection to Application**:
   After successful login, Clinion redirects the user back to the URL provided in the callback, such as `https://yourapp.com/home`.

Error Handling

If authentication fails, the user may be redirected back to the callback URL with an error parameter in the query string.
Example:

bash
Copy code
`https://yourapp.com/home?error=invalid_credentials`

-

In this case, your application should handle the error by displaying an appropriate message to the user.

Notes

- Ensure that the **callback URL** is correctly configured in the Clinion SSO settings to allow redirection back to your application.
- If using a local environment (e.g., `http://localhost:4200`), ensure that this is whitelisted in the SSO provider's settings.

User Registration API Documentation

Overview

The `POST /register` endpoint registers a new user in AWS Cognito. The request payload includes the user's email, customer name, and callback URL. A temporary password is sent to the user's email, and they are required to change it upon their first login. Once the password is changed, the user will be redirected to the specified callback URL.

Endpoint

- **URL**: `/register`
- **Method**: `POST`

Request Body

The request body must contain the following fields:

Field	Type	Required	Description
`userName`	String	Yes	The email address of the user (used as the username).
`customer`	String	Yes	The customer or organization the user belongs to.

`callback`	URL	Yes	The URL to redirect the user after successful login.

Example Request

json
Copy code

```
{
  "userName": "user@example.com",
  "customer": "novartis",
  "callback": "https://www.clinion.com"
}
```

Process

1. **Fetch Pool Details**:
   Based on the `customer` value provided, the system retrieves the customer-specific Cognito User Pool details (`UserPoolId` and `ClientId`) from the connection pool. These details are used for the registration process.
2. **Validation**:
   If the `UserPoolId` or `ClientId` is not valid or if any mandatory field is missing from the request, the API responds with a `500` status and an error message.
3. **Temporary Password and User Registration**:
   A new user is created in the Cognito User Pool with a temporary password that is sent to their email. The command includes:
   - **UserPoolId**: The ID of the Cognito User Pool.
   - **ClientId**: The ID of the Cognito client application.
   - **Username**: The email address provided in `userName`.
   - **UserAttributes**: The user's email and a flag indicating the email is verified (`email_verified: true`).
4. **Force Password Change**:
   After logging in with the temporary password, the user is required to change their password. Upon successfully changing the password, they will be redirected to the **callback URL** provided in the request.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The user registration was successful, and a temporary password has been sent to their email.

- **Response Body**:

json
Copy code
```
{
  "message": "User registration success",
  "data": {
    // Cognito response data
  }
}
```

Error Response

- **Status**: `500 Internal Server Error`
- **Description**: The registration failed due to invalid pool details or missing mandatory fields.
- **Response Body**:

json
Copy code
```
{
  "error": "Failed to register user",
  "details": {
    // Error details
  }
}
```

Example Request

bash
Copy code
```
curl -X POST https://yourapi.com/register \
-H "Content-Type: application/json" \
-d '{
  "userName": "user@example.com",
  "customer": "novartis",
  "callback": "https://www.clinion.com"
}'
```

Temporary Password Flow

1. **User Registration**:
   The user is registered, and a temporary password is sent to their email.
2. **User Login**:
   The user logs in with the temporary password.
3. **Force Password Change**:
   Cognito forces the user to change the temporary password upon first login.
4. **Redirection to Callback URL**:
   After the password is successfully changed, the user is redirected to the `callback` URL provided during registration (e.g., `https://www.clinion.com`).

Notes

- **Mandatory Fields**:
  The fields `userName`, `customer`, and `callback` are all required for the registration process.
- **Email Verification**:
  The email is marked as verified during registration by default.
- **Callback URL**:
  The user is redirected to this URL after they successfully change their password.

Token Refresh API Documentation

Overview

The `POST /refreshToken` endpoint is used to refresh the user's authentication tokens (ID token, access token, and refresh token) using AWS Cognito. The request payload includes the user's email, customer name, and refresh token.

Endpoint

- **URL**: `/refreshToken`
- **Method**: `POST`

Request Body

The request body must contain the following fields:

Field	Type	Required	Description
`userName`	String	Yes	The email address of the user (used as the username).
`customer`	String	Yes	The customer or organization the user belongs to.
`refreshToken`	String	Yes	The refresh token used to obtain new authentication tokens.

Example Request

json
Copy code

```
{
  "userName": "user@email.com",
  "customer": "SME",
  "refreshToken": "SweqweewXgg3673737VV"
}
```

Process

1. **Fetch Pool Details**:
   Based on the `customer` value provided, the system retrieves the customer-specific Cognito User Pool details (`UserPoolId` and `ClientId`) from the connection pool.
2. **Validation**:
   If the `ClientId` is not valid or if any mandatory field is missing from the request, the API responds with a `500` status and an error message.
3. **Refresh Tokens**:
   A new token set (ID token, access token, refresh token) is generated using the `AdminInitiateAuthCommand` from AWS Cognito. The process is done using the `REFRESH_TOKEN_AUTH` flow, where the provided refresh token is validated and new tokens are issued.
4. **Setting Cookies**:
   After successful token refresh, the new tokens are stored in cookies (`idToken`, `accessToken`, and `refreshToken`) to maintain session state for the user.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The tokens were refreshed successfully.

- **Response Body**:

json
Copy code
```
{
  "message": "Tokens refreshed",
  "data": {
    "IdToken": "new-id-token",
    "AccessToken": "new-access-token",
    "RefreshToken": "new-refresh-token"
  }
}
```

Error Response

- **Status**: `401 Unauthorized`
- **Description**: The token refresh failed due to an invalid refresh token or other issues.
- **Response Body**:

json
Copy code
```
{
  "error": "Failed to refresh token",
  "details": {
    // Error details
  }
}
```

Example Request

bash
Copy code
```
curl -X POST https://yourapi.com/refreshToken \
-H "Content-Type: application/json" \
-d '{
  "userName": "user@email.com",
  "customer": "SME",
  "refreshToken": "SweqweewXgg3673737VV"
}'
```

Notes

- **Mandatory Fields**:
  The fields `userName`, `customer`, and `refreshToken` are all required for the token refresh process.
- **Token Storage**:
  The newly issued tokens are stored in cookies to facilitate session management for the user.

Change Password API Documentation

Overview

The `PUT /changePassword` endpoint allows users to change their password. The request requires the current password (previous password) and the new password (proposed password), along with a valid access token in the request headers for authentication.

Endpoint

- **URL**: `/changePassword`
- **Method**: `PUT`

Request Headers

Header	Type	Required	Description
`Authorization`	String	Yes	The bearer token (access token) for authentication.

The `Authorization` header must contain the Bearer token in the format:

makefile
Copy code
```
Authorization: Bearer <access_token>
```

Request Body

The request body must contain the following fields:

Field	Type	Required	Description
`previousPassword`	String	Yes	The user's current password.
`proposedPassword`	String	Yes	The new password the user wants to set.

Example Request

json
Copy code

```
{
  "previousPassword": "OldPassword123!",
  "proposedPassword": "NewPassword456!"
}
```

Process

1. **Authorization**:
   The access token is extracted from the `Authorization` header to verify the user's identity.
2. **Password Change**:
   The API uses AWS Cognito's `ChangePasswordCommand` to update the user's password. The command is executed using the `previousPassword` and `proposedPassword` values provided in the request body.
3. **Response Handling**:
   If the password change is successful, the API responds with a success message. If there is an error (e.g., invalid access token or password mismatch), appropriate error handling is triggered.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The password was changed successfully.
- **Response Body**:

json
Copy code

```
{
```

```
  "message": "Password changed successfully"
}
```

Error Response

- **Status**: Various (e.g., `401 Unauthorized`, `400 Bad Request`)
- **Description**: The password change failed due to invalid access token, password mismatch, or other errors.
- **Response Body**:

json
Copy code

```
{
  "error": "Failed to change password",
  "details": {
    // Error details
  }
}
```

Example Request

bash
Copy code

```
curl -X PUT https://yourapi.com/changePassword \
-H "Content-Type: application/json" \
-H "Authorization: Bearer yourAccessToken" \
-d '{
  "previousPassword": "OldPassword123!",
  "proposedPassword": "NewPassword456!"
}'
```

Notes

- **Authorization**:
  The `Authorization` header is mandatory and should include a valid access token (`Bearer <access_token>`).
- **Password Validation**:
  Ensure that the `proposedPassword` follows the required password policy (e.g., length, special characters, etc.).

Overview

The `POST /refreshToken` endpoint is used to refresh the user's authentication tokens (ID token, access token, and refresh token) using AWS Cognito. The request payload includes the user's email, customer name, and refresh token.

Endpoint

- **URL**: `/refreshToken`
- **Method**: `POST`

Request Body

The request body must contain the following fields:

Field	Type	Required	Description
`userName`	String	Yes	The email address of the user (used as the username).
`customer`	String	Yes	The customer or organization the user belongs to.
`refreshToken`	String	Yes	The refresh token used to obtain new authentication tokens.

Example Request
json
Copy code

```
{
  "userName": "user@email.com",
  "customer": "SME",
  "refreshToken": "SweqweewXgg3673737VV"
}
```

Process

1. **Fetch Pool Details**:
   Based on the `customer` value provided, the system retrieves the customer-specific Cognito User Pool details (`UserPoolId` and `ClientId`) from the connection pool.

2. **Validation**:
   If the `ClientId` is not valid or if any mandatory field is missing from the request, the API responds with a `500` status and an error message.
3. **Refresh Tokens**:
   A new token set (ID token, access token, refresh token) is generated using the `AdminInitiateAuthCommand` from AWS Cognito. The process is done using the `REFRESH_TOKEN_AUTH` flow, where the provided refresh token is validated and new tokens are issued.
4. **Setting Cookies**:
   After successful token refresh, the new tokens are stored in cookies (`idToken`, `accessToken`, and `refreshToken`) to maintain session state for the user.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The tokens were refreshed successfully.
- **Response Body**:

json
Copy code

```
{
  "message": "Tokens refreshed",
  "data": {
    "IdToken": "new-id-token",
    "AccessToken": "new-access-token",
    "RefreshToken": "new-refresh-token"
  }
}
```

Error Response

- **Status**: `401 Unauthorized`
- **Description**: The token refresh failed due to an invalid refresh token or other issues.
- **Response Body**:

json
Copy code

```
{
  "error": "Failed to refresh token",
```

```
  "details": {
    // Error details
  }
}
```

Example Request

bash
Copy code

```
curl -X POST https://yourapi.com/refreshToken \
-H "Content-Type: application/json" \
-d '{
  "userName": "user@email.com",
  "customer": "SME",
  "refreshToken": "SweqweewXgg3673737VV"
}'
```

Notes

- **Mandatory Fields**:
  The fields `userName`, `customer`, and `refreshToken` are all required for the token refresh process.
- **Token Storage**:
  The newly issued tokens are stored in cookies to facilitate session management for the user.

Overview

The `POST /register` endpoint registers a new user in AWS Cognito. The request payload includes the user's email, customer name, and callback URL. A temporary password is sent to the user's email, and they are required to change it upon their first login. Once the password is changed, the user will be redirected to the specified callback URL.

Endpoint

- **URL**: `/register`
- **Method**: `POST`

Request Body

The request body must contain the following fields:

Field	Type	Required	Description
`userName`	String	Yes	The email address of the user (used as the username).
`customer`	String	Yes	The customer or organization the user belongs to.
`callback`	URL	Yes	The URL to redirect the user after successful login.

Example Request

json

Copy code

```
{
  "userName": "user@example.com",
  "customer": "novartis",
  "callback": "https://www.clinion.com"
}
```

Process

1. **Fetch Pool Details**:
   Based on the `customer` value provided, the system retrieves the customer-specific Cognito User Pool details (`UserPoolId` and `ClientId`) from the connection pool. These details are used for the registration process.
2. **Validation**:
   If the `UserPoolId` or `ClientId` is not valid or if any mandatory field is missing from the request, the API responds with a `500` status and an error message.
3. **Temporary Password and User Registration**:
   A new user is created in the Cognito User Pool with a temporary password that is sent to their email. The command includes:
   - **UserPoolId**: The ID of the Cognito User Pool.
   - **ClientId**: The ID of the Cognito client application.
   - **Username**: The email address provided in `userName`.
   - **UserAttributes**: The user's email and a flag indicating the email is verified (`email_verified: true`).
4. **Force Password Change**:
   After logging in with the temporary password, the user is required to change their

password. Upon successfully changing the password, they will be redirected to the **callback URL** provided in the request.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The user registration was successful, and a temporary password has been sent to their email.
- **Response Body**:

json
Copy code

```
{
  "message": "User registration success",
  "data": {
    // Cognito response data
  }
}
```

Error Response

- **Status**: `500 Internal Server Error`
- **Description**: The registration failed due to invalid pool details or missing mandatory fields.
- **Response Body**:

json
Copy code

```
{
  "error": "Failed to register user",
  "details": {
    // Error details
  }
}
```

Example Request

bash
Copy code

```
curl -X POST https://yourapi.com/register \
-H "Content-Type: application/json" \
-d '{
  "userName": "user@example.com",
  "customer": "novartis",
  "callback": "https://www.clinion.com"
}'
```

Temporary Password Flow

1. **User Registration**:
   The user is registered, and a temporary password is sent to their email.
2. **User Login**:
   The user logs in with the temporary password.
3. **Force Password Change**:
   Cognito forces the user to change the temporary password upon first login.
4. **Redirection to Callback URL**:
   After the password is successfully changed, the user is redirected to the `callback` URL provided during registration (e.g., `https://www.clinion.com`).

Notes

- **Mandatory Fields**:
  The fields `userName`, `customer`, and `callback` are all required for the registration process.
- **Email Verification**:
  The email is marked as verified during registration by default.
- **Callback URL**:
  The user is redirected to this URL after they successfully change their password.

Sign Out API Documentation

Overview

The `POST /signout` endpoint is used to sign out the currently authenticated user globally. This invalidates the access token and clears all session cookies. The user must provide a valid access token in the request headers. After a successful sign-out, the user can choose where to navigate next.

Endpoint

- **URL**: `/signout`
- **Method**: `POST`

Request Headers

Header	Type	Required	Description
`Authorization`	String	Yes	The bearer token (access token) of the user to sign out.

The `Authorization` header must contain the Bearer token in the format:

makefile

Copy code

```
Authorization: Bearer <access_token>
```

Process

1. **Authorization**:
   The API extracts the access token from the `Authorization` header to identify and sign out the user globally using AWS Cognito's `GlobalSignOutCommand`. This invalidates the user's session across all devices.
2. **Clear Cookies**:
   After the global sign-out is successful, the API clears all relevant session cookies (`isAuthenticated`, `user`, `idToken`, `accessToken`, and `refreshToken`) to log the user out locally.
3. **Redirect Options**:
   After signing out and clearing cookies, users can choose where they want to navigate next. The API does not dictate a specific URL for redirection.
4. **Response Handling**:
   If the sign-out is successful, the API returns a success message. If an error occurs (e.g., invalid access token), the API handles the error and returns an appropriate response.

Response

Success Response

- **Status**: `200 OK`
- **Description**: The user was signed out successfully, and cookies were cleared.
- **Response Body**:

json

Copy code

```
{
  "message": "User signed out successfully"
}
```

Error Response

- **Status**: `500 Internal Server Error`
- **Description**: The sign-out failed due to an invalid access token or other issues.
- **Response Body**:

json

Copy code

```
{
  "error": "Failed to sign out",
  "details": {
    // Error details
  }
}
```

Example Request

bash

Copy code

```
curl -X POST https://yourapi.com/signout \

-H "Content-Type: application/json" \

-H "Authorization: Bearer yourAccessToken"
```

Notes

- **Global Sign Out**:
  The `GlobalSignOutCommand` ensures that the user is logged out globally, invalidating the session across all devices.
- **Cookies Cleared**:
  The API clears session-related cookies after a successful sign-out to ensure the user is logged out from the current session.
- **User Navigation**:
  After a successful sign-out, users can decide where to navigate next, giving them flexibility in their application experience.