HeadlessUserDiscoveryHandler Interface

Use this interface to create a headless user discovery handler that you implement during he login, passwordless login, and forgot password flows.

Namespace

Auth

Usage

Develop headless authorization flows where users log in to an off-platform app with an ider other than their username, such as an email address, phone number, or order number. Wh user enters the identifier in your headless app, your app sends the identifying information t Salesforce endpoint. Salesforce then passes the identifying information to your implementa the `Auth.HeadlessUserDiscoveryHandler` interface. The handler finds the user's account and i associated email address or phone number.

Headless user discovery supports these use cases.

- Headless login with any identifier and a password. For example, a user goes to your h app and enters their order number and password to log in.
- Headless login with any identifier and a one-time password (OTP). For example, a use to your app and enters just their order number. Your Apex handler finds the user's ac based on the order number. Salesforce sends an OTP to the verified email address th associated with the account. To log in, the user enters the OTP.
- Headless password reset with any identifier. For example a user goes to your app and their phone number. Your Apex handler finds the user account and Salesforce sends to the user's verified phone number. To verify their identity for password reset, the us enters the OTP and can then set a new password.

Headless user discovery is supported for Headless Identity API flows and OAuth 2.0 for First Applications flows. For more information about supported flows and implementation detai Headless Login Without a Username.

- **HeadlessUserDiscoveryHandler Methods**
- **HeadlessUserDiscoveryHandler Example Implementation**

HeadlessUserDiscoveryHandler Methods

The following are methods for `HeadlessUserDiscoveryHandler`.

- **discoverUserFromLoginHint(networkId, loginHint, verificationAction, customDataJson, requestAttributes)**
  Finds a user's Salesforce account based on user information, such as their email addr phone number, or other data, that's passed to a Salesforce endpoint during headless login, passwordless login, and forgot password flows.

discoverUserFromLoginHint(networkId, loginHint, verificationAction, customDataJson, requestAttributes)

```
public Auth.HeadlessUserDiscoveryResponse discoverUserFromLoginHint(Id networkId, Strin
loginHint, Auth.VerificationAction verificationAction, String customDataJson,
Map<String,String> requestAttributes)
```

Parameters

networkId

Type: Id

The ID of the Experience Cloud site where your headless app sends requests.

loginHint

Type: String

Information about the user that Salesforce can use to find their associated account, such email address or phone number.

verificationAction

Type: Auth.VerificationAction

The verification method that's used to log the user in, either email or SMS.

customDataJson

Type: String

Custom user data, such as first name, that you collect when the user logs in to your headl

requestAtttibutes

Type: Map<String,String>

Information about the login request that's based on the user's browser state when access login page. `requestAttributes` passes in the CommunityUrl, IpAddress, UserAgent, Platfor Application, City, Country, and Subdivision values. The City, Country, and Subdivision valu from IP geolocation.

Return Value

Type: Auth.HeadlessUserDiscoveryResponse

If the handler finds a user, it returns a user ID. If not, it returns an error message.

HeadlessUserDiscoveryHandler Example Implementation

Here's an example implementation of the `Auth.HeadlessUserDiscoveryHandler` interface. This example supports login with email and login with SMS.

The `discoverUserFromLoginHint` method uses custom logic to search for a user account with verified email address or phone number that matches the data passed in the login hint. As security best practice, Salesforce always recommends writing code to determine if the user' address or phone number is verified.

For users logging in with email, the custom logic first checks whether the email address pas the login hint is in a valid format. Then, to look for a verified Salesforce email address that n the email address passed in the login hint, it queries the TwoFactorMethodsInfo object. If successful, it returns an instance of `Auth.HeadlessUserDiscoveryResponse` with the user ID. If something goes wrong, it returns an instance of `Auth.HeadlessUserDiscoveryResponse` with a error message. In this example, it returns error messages when the email address format isn the email address isn't verified, there's no user with that email address, or there are multipl with that email address.

```
/*
 * Headless User Discovery Handler
 */
global class MyHeadlessUserDiscoveryHandler implements Auth.HeadlessUserDiscoveryH


  /*
  * This method handles the logic to determine the user account based on the logir
  */
  global Auth.HeadlessUserDiscoveryResponse discoverUserFromLoginHint(Id networkId
  Auth.VerificationAction verificationAction, String customDataJson, Map<String,St
    if (verificationAction == Auth.VerificationAction.EMAIL) {
      return doLookupByVerifiedEmail(loginHint, verificationAction);
    } else if (verificationAction == Auth.VerificationAction.SMS) {
      return doLookupByVerifiedMobile(loginHint, verificationAction);
    } else {
      return new Auth.HeadlessUserDiscoveryResponse(null, 'Unsupported Auth.Verifi
    }
  }

  private Auth.HeadlessUserDiscoveryResponse doLookupByVerifiedEmail(String loginH
    if (String.isBlank(loginHint) || !isValidEmail(loginHint)) {
      return new Auth.HeadlessUserDiscoveryResponse(null, 'Invalid email sent as l
    }
    // Search for an user account by email
    List<User> users = [SELECT Id FROM User WHERE Email = :loginHint AND IsActive
    if (!users.isEmpty() && users.size() == 1) {
      Id userId = users[0].Id;
      // Check if the user has a verified email
      List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedEmailAddres
      if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedEmailAddress =
        // Prepare and return HeadlessUserDiscoveryResponse with userId
        return new Auth.HeadlessUserDiscoveryResponse(new Set<Id>{userId}, null);
      } else {
        // Return HeadlessUserDiscoveryResponse with error message
        return new Auth.HeadlessUserDiscoveryResponse(null, 'Email ' + loginHint +
      }
    } else {
      if (users.isEmpty()) {
        return new Auth.HeadlessUserDiscoveryResponse(null, 'No user identified fo
      } else {
        return new Auth.HeadlessUserDiscoveryResponse(null, 'Multiple users identi
      }
    }
  }

  private Auth.HeadlessUserDiscoveryResponse doLookupByVerifiedMobile(String logir
    String formattedSms = !String.isBlank(loginHint) ? getFormattedSms(loginHint)
    if (String.isBlank(formattedSms)) {
      return new Auth.HeadlessUserDiscoveryResponse(null, 'Invalid phone number se
    }
    // Search for an user account by phone
    List<User> users = [SELECT Id FROM User WHERE MobilePhone = :loginHint AND IsA
    if (!users.isEmpty() && users.size() == 1) {
      Id userId = users[0].Id;
      // Check if the user has a verified phone
      List<TwoFactorMethodsInfo> verifiedInfo = [SELECT HasUserVerifiedMobileNumbe
      if (!verifiedInfo.isEmpty() && verifiedInfo[0].HasUserVerifiedMobileNumber =
        // Prepare and return HeadlessUserDiscoveryResponse with userId
        return new Auth.HeadlessUserDiscoveryResponse(new Set<Id>{userId}, null);
      } else {
        // Return HeadlessUserDiscoveryResponse with error message
        return new Auth.HeadlessUserDiscoveryResponse(null, ' ' + loginHint + ' no
      }
    } else {
      if (users.isEmpty()) {
        return new Auth.HeadlessUserDiscoveryResponse(null, 'No user identified fo
      } else {
        return new Auth.HeadlessUserDiscoveryResponse(null, 'Multiple users identi
      }
```

```
    Pattern EmailPattern = Pattern.compile(emailRegex);
    Matcher EmailMatcher = EmailPattern.matcher(identifier);
    if (EmailMatcher.matches()) { return true; }
    else { return false; }
  }

  private String getFormattedSms(String identifier) {
    // Accept SMS input formats with 1 or 2 digits country code, 3 digits area coc
    // You can customize the SMS regex to allow different formats
    String smsRegex = '^(\\+?\\d{1,2}?[\\s-])?(\\(?\\d{3}\\)?[\\s-]?\\d{3}[\\s-]?'
    Pattern smsPattern = Pattern.compile(smsRegex);
    Matcher smsMatcher = SmsPattern.matcher(identifier);
    if (smsMatcher.matches()) {
      try {
        // Format user input into the verified SMS format '+xx xxxxxxxxx' before
        // Append US country code +1 by default if no country code is provided
        String countryCode = smsMatcher.group(1) == null ? '+1' : smsMatcher.group
        return System.UserManagement.formatPhoneNumber(countryCode, smsMatcher.gro
      } catch(System.InvalidParameterValueException e) {
        return null;
      }
    } else { return null; }
  }
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your fee