Contents

[**I.** **Introduction** 1](#_Toc169797656)

[**1.** **Purpose of the document** 1](#_Toc169797657)

[**2.** **Scope of the document** 1](#_Toc169797658)

[**3.** **Target audience of the document** 2](#_Toc169797659)

[**II.** **Overview of the SSO System** 2](#_Toc169797660)

[**1.** **Overview of the SSO Model Workflow** 2](#_Toc169797661)

[**2.** **Overview of the Main Components of the Centralized Authentication Model** 3](#_Toc169797662)

[**3.** **Main Functions (within the scope of the demo)** 3](#_Toc169797663)

[**4.** **Tổ chức thông tin user tổng quan** 4](#_Toc169797664)

[**III.** **Flow Diagrams** 5](#_Toc169797665)

[**1.** **Login function** 5](#_Toc169797666)

[**2.** **Logout function** 6](#_Toc169797667)

[**3.** **Associate social account function** 7](#_Toc169797668)

[**IV.** **API/Endpoint Integration Specification** 8](#_Toc169797669)

[**1.** **Login endpoint** 8](#_Toc169797670)

[**2.** **Logout enpoint** 9](#_Toc169797671)

[**3.** **API get access-token** 9](#_Toc169797672)

[**4.** **API get user-info** 11](#_Toc169797673)

[**V.** **Conclusion** 12](#_Toc169797674)

[**1.** **Missing points in the document:** 12](#_Toc169797675)

[**2.** **Upcoming Content (if the project is approved)** 12](#_Toc169797676)

**SSO Design Document**

1. **Introduction**
2. **Purpose of the document**

This design document aims to provide a comprehensive and detailed overview of the Single Sign-On (SSO) system design. The document will present key technical aspects, including authentication flow, session management, database structure, and related APIs

1. **Scope of the document**

This design document focuses on the technical aspects for the demo of Dextrends' Single Sign-On (SSO) system, including:

* Login: Detailed description of the processing flow when users log in using different methods (user/email, Facebook, Google).
* Logout: Explanation of the session termination process when a user logs out.
* Database: Presentation of a general user/account database structure diagram.
* API/Endpoint: Listing of APIs/Endpoints provided according to the OAuth standard to integrate a subsystem into the SSO system.

Within the scope of the demo, this document assumes and does not focus heavily on the following issues:

* Use of default user interface.
* Does not focus on describing the technical implementation in subsystems (due to programming factors).
* Due to the writer's limitations in mobile technologies, there will be no demo for integration on mobile apps.

1. **Target audience of the document**

The target audience of this document includes:

* Developers: to understand the technical requirements, processing flows, data structures, and necessary APIs.
* Software architects: to assess the feasibility, effectiveness, and scalability of the SSO design model, and make decisions related to technology selection and integration with other systems.

1. **Overview of the SSO System**
2. **Overview of the SSO Model Workflow**

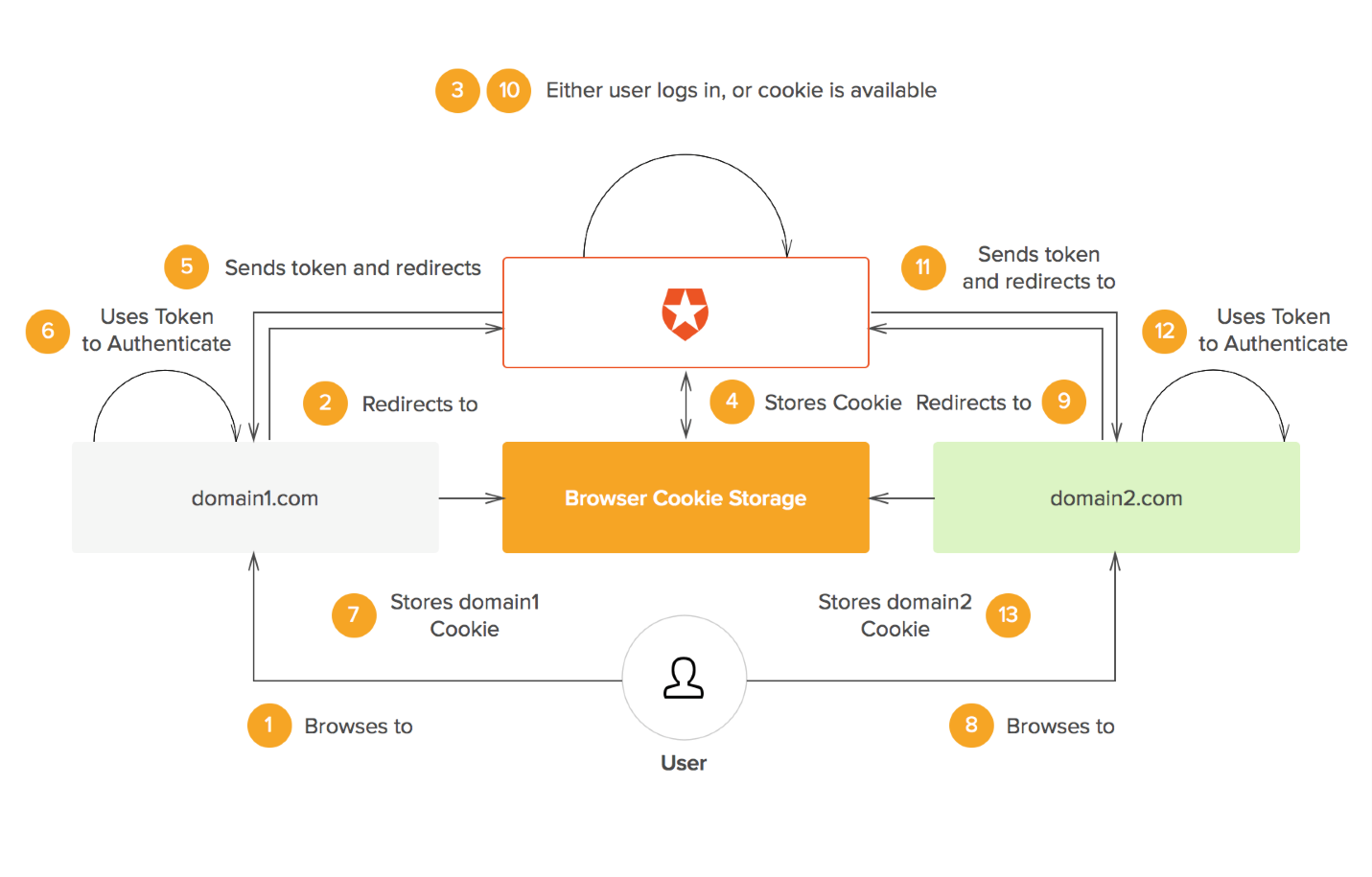


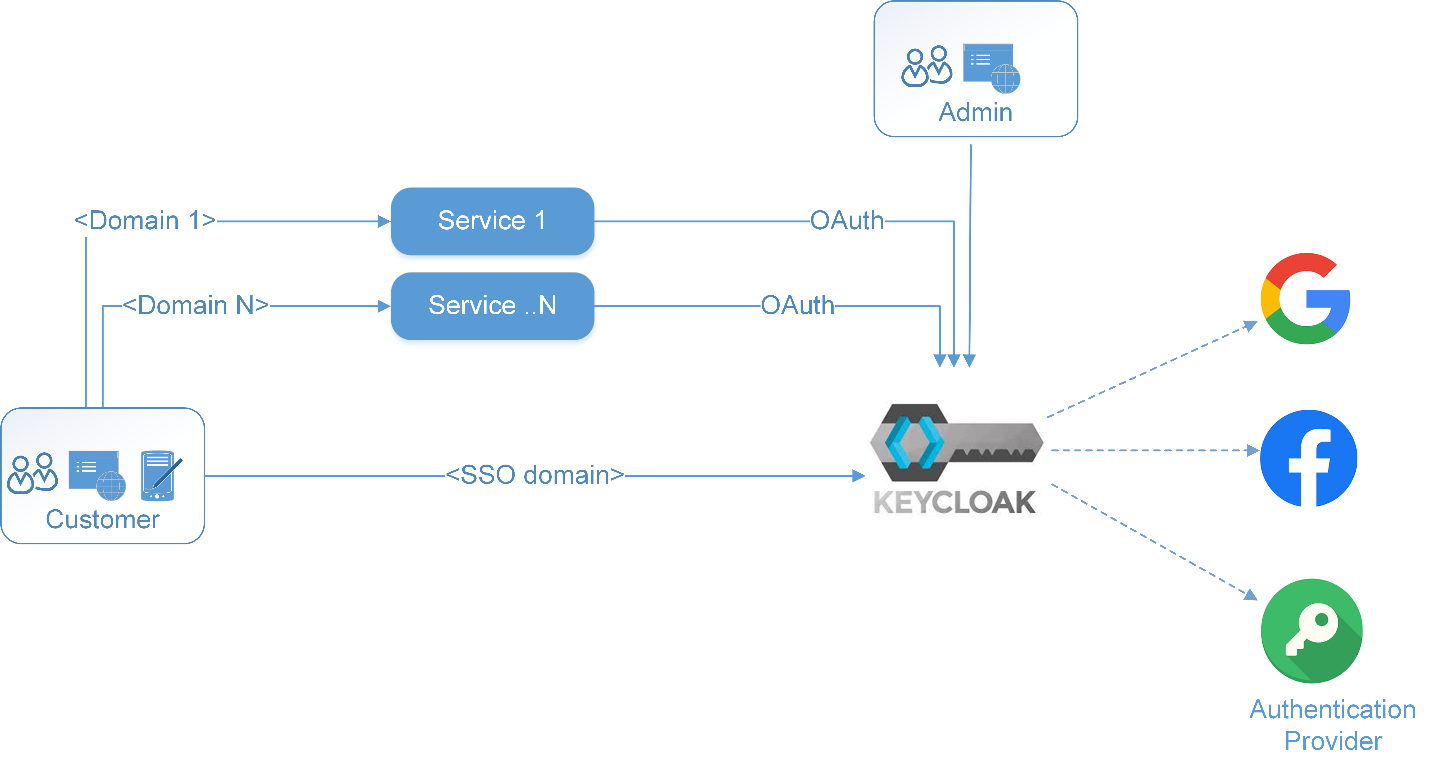
Figure 1: General Workflow of the SSO Model

Key points in this model:

* Utilize a centralized authentication system for subsystems, including the following functions:
  + Centralized identity & access management
  + Session management
  + Support single sign-on (SSO) within any domain, and automatic authentication when logging in on other domains
* To integrate, subsystems need to:
  + Redirect to the centralized login interface (instead of the login interface at each subsystem)
  + Implement authentication based on the token returned from the centralized authentication system (instead of using the traditional authentication system)

1. **Overview of the Main Components of the Centralized Authentication Model**

In this project, Keycloak is chosen as the framework to develop the centralized SSO authentication system:



1. **Main Functions (within the scope of the demo)**

* Login according to the SSO model
* Logout according to the SSO model
* Link accounts with other popular social media accounts (within the scope of this topic, support demo linking with Google accounts)

1. **User Information Models**

In the given problem, besides supporting traditional user (or email) and password authentication, the proposed SSO system requires support for authentication through popular social platforms like Google, Facebook, etc.

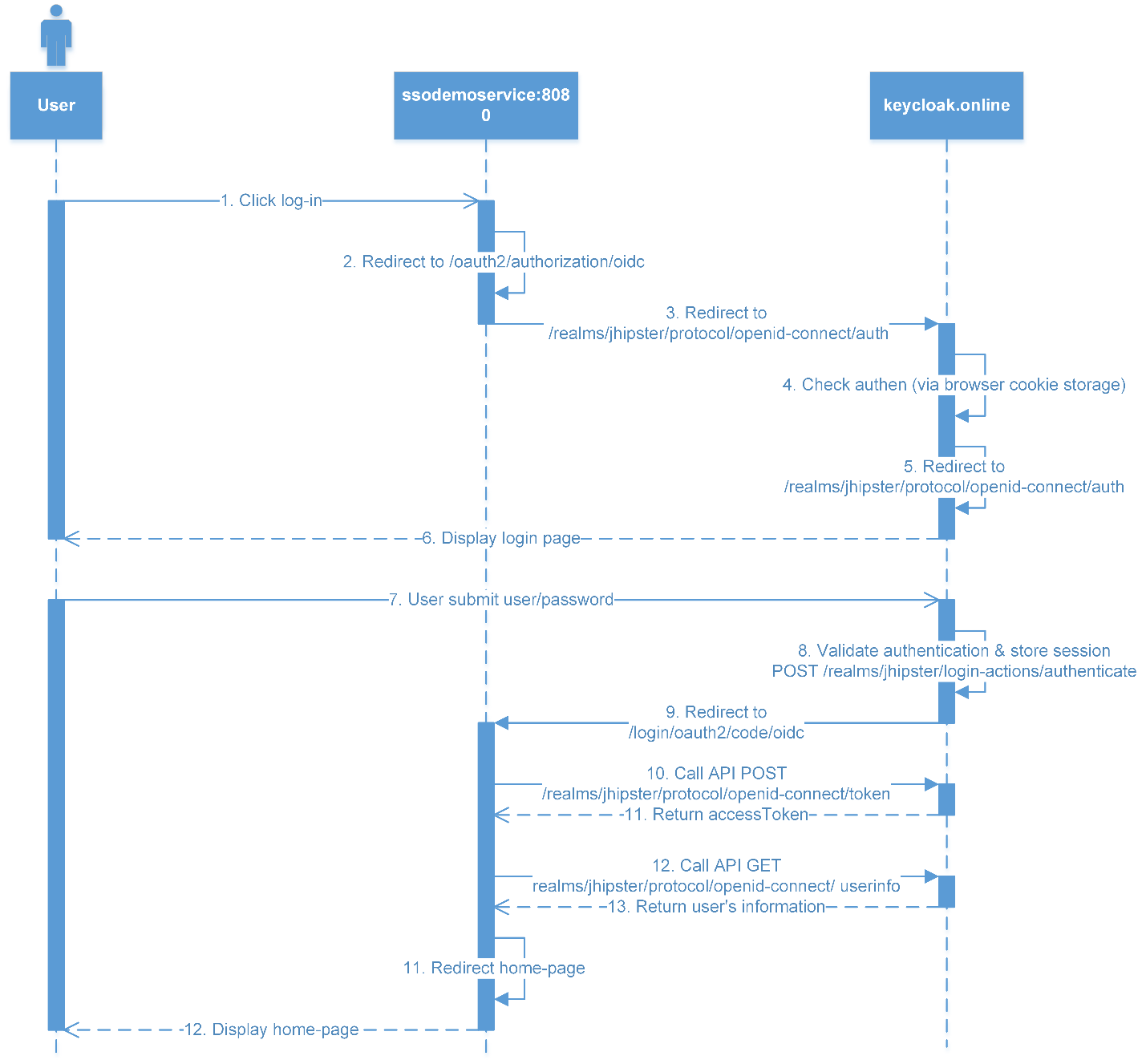
Therefore, the proposed user organization model is as follows:



In which:

* **User**: The table storing the main account information in the company's system
* **Federated\_identity**: The table storing account information from social media platforms (or other systems) that support linking
* One User can have multiple linked account information

1. **Flow Diagrams**
2. **Login function**



**Scenario:**

User accesses the service:

* If not logged in, display the login screen with options: email, Facebook, Google.
* If already logged in, redirect to the service's homepage.

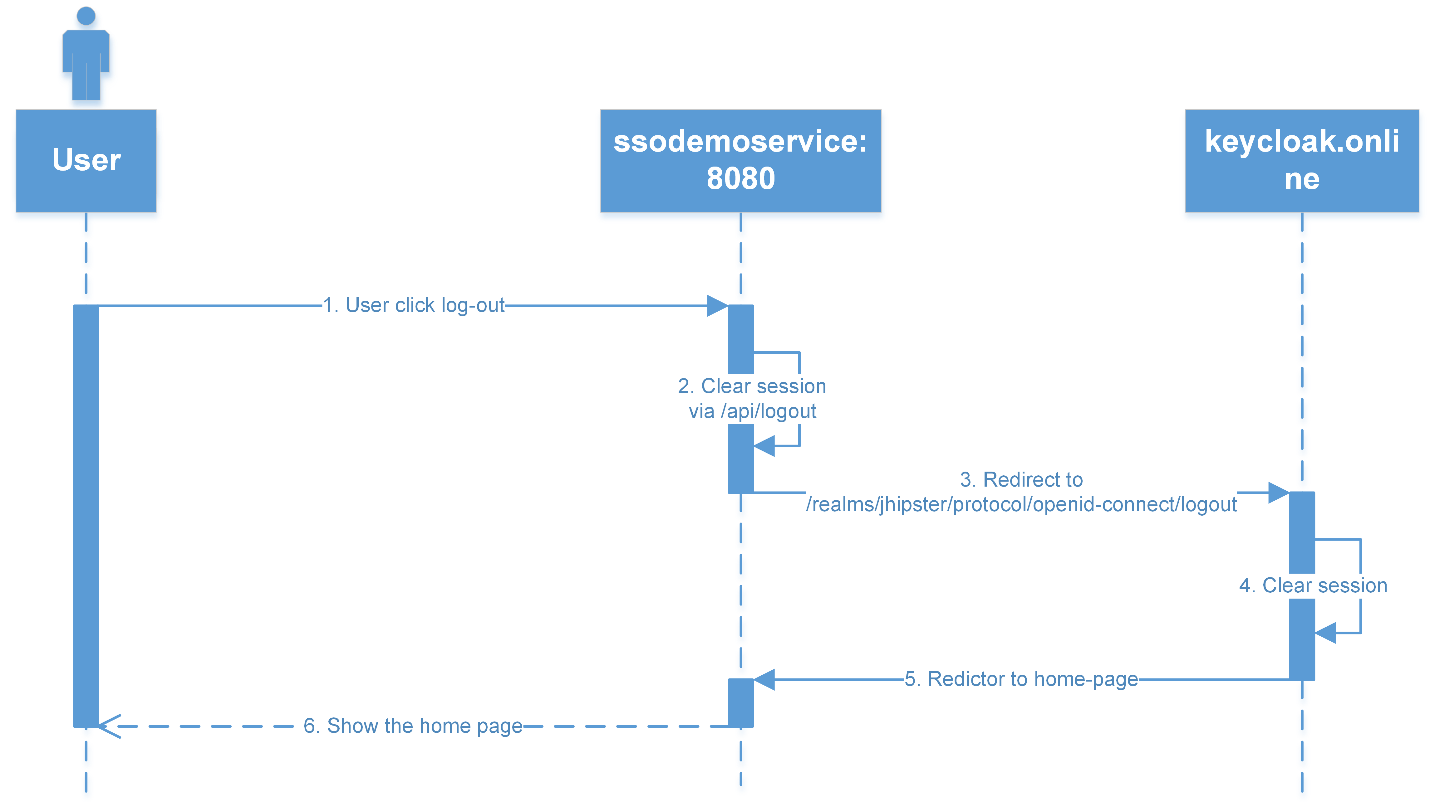
User logs in:

* Authenticate login information.
* Create a login session and store user information.

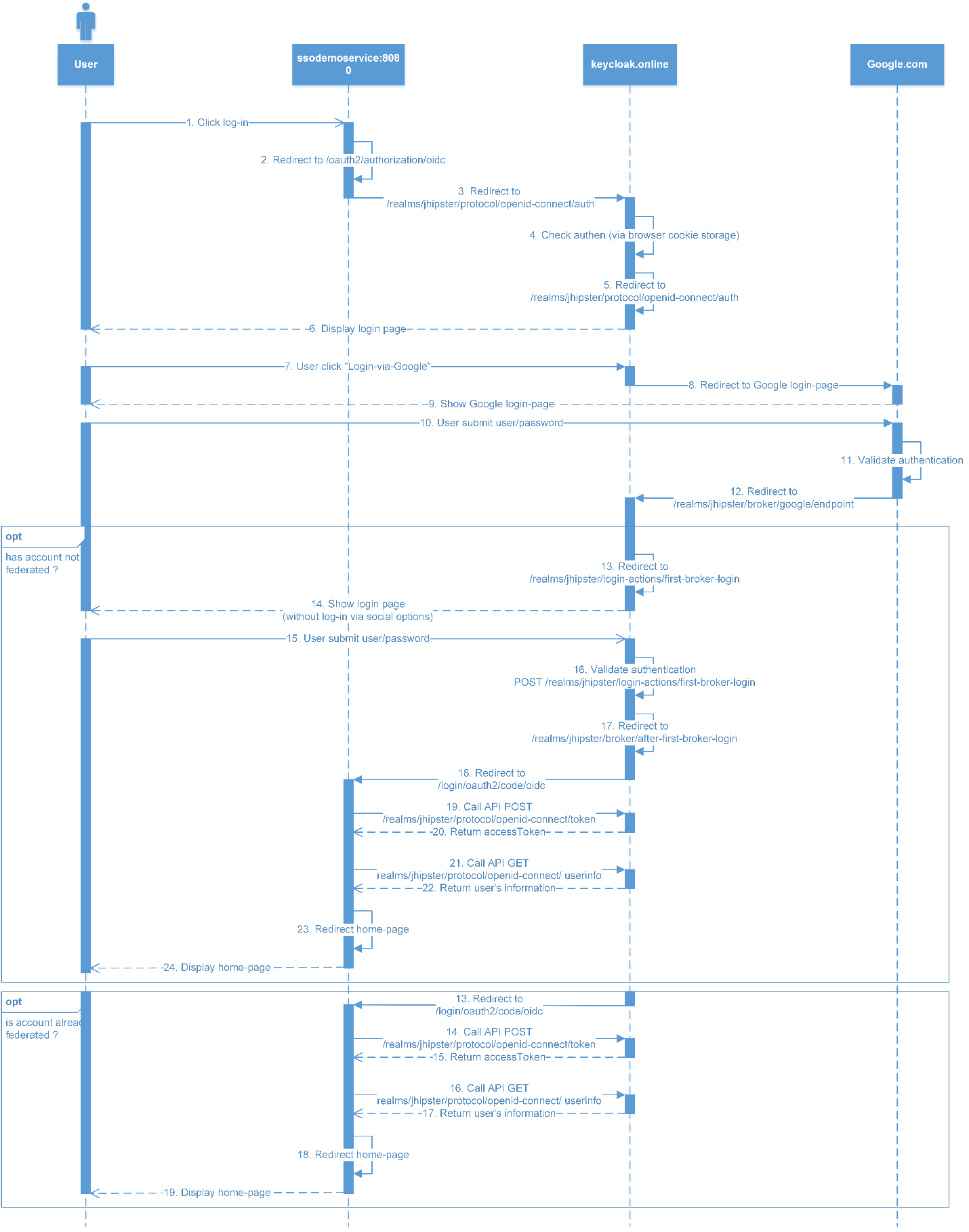
User accesses other services:

* Check login session.
* If the session is valid, allow access.
* If the session is invalid, require re-login.

1. **Logout function**



1. **Associate social account function**



**Scenario:**

User logs in using a different service (in this case, Google-authen-provider is used):

* The system redirects to Google's login page.
* The user logs in according to Google's instructions.
* The system redirects back to the SSO login page (if Google login is successful).
* The user logs in with their original account.
* The system automatically links the original account with the Google account.
* Create a login session and redirect to the service's homepage.

1. **API/Endpoint Integration Specification**
2. **Login endpoint**

**Request Details:**

* *URL: /realms/{realm}/protocol/{protocal}/auth*
* *HTTP Version: HTTP/1.1*
* *Full URL: http://keycloak.online:9080/realms/jhipster/protocol/openid-connect/auth?response\_type=code&client\_id=ssodemoservice01&scope=openid profile email offline\_access&state=arGmrK8G2lmxQ6bdt9A2N0\_V2In8wB5MOYUjZhUMgNM=&redirect\_uri=http://ssodemoservice01:8080/login/oauth2/code/oidc&nonce=-rhKs5Dn8yFjpMsjfKeXRV4t3-AOw2hhXPY4RerfCF0*

**Example endpoint:**

* *GET /realms/jhipster/protocol/openid-connect/auth?response\_type=code&client\_id=ssodemoservice01&scope=openid%20profile%20email%20offline\_access&state=arGmrK8G2lmxQ6bdt9A2N0\_V2In8wB5MOYUjZhUMgNM=&redirect\_uri=http://ssodemoservice01:8080/login/oauth2/code/oidc&nonce=-rhKs5Dn8yFjpMsjfKeXRV4t3-AOw2hhXPY4RerfCF0 HTTP/1.1*

*Host: keycloak.online:9080*

**Explanation:**

* *URL Parameters:*
  + *response\_type=code: Indicates that the authorization server should return an authorization code.*
  + *client\_id=ssodemoservice01: The client identifier.*
  + *scope=openid profile email offline\_access: The scope of the access request.*
  + *state=arGmrK8G2lmxQ6bdt9A2N0\_V2In8wB5MOYUjZhUMgNM=: An opaque value used to maintain state between the request and the callback.*
  + *redirect\_uri=http://ssodemoservice01:8080/login/oauth2/code/oidc: The redirect URI where the authorization server will send the user once the access request is granted or denied.*
  + *nonce=-rhKs5Dn8yFjpMsjfKeXRV4t3-AOw2hhXPY4RerfCF0: A string value used to associate a client session with an ID token.*

1. **Logout enpoint**

**Request Details:**

* *URL: /realms/{realm}/protocol/{protocol}/logout*
* *HTTP Version: HTTP/1.1*

**Example endpoint:**

* *GET /realms/jhipster/protocol/openid-connect/logout?id\_token\_hint=eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI4cGJRSnljZHZJc1FrRzJybElNUFlLNFFNeXdYeW02Wk5meGJkQlpwUGp3In0..GBDUkPcnwav9hl317h5\_ay3gNT1FANKejHG8hULnOS6GHU7SWHN-2doUYotic5kg6LBBexAlpdlIxBz4cQOtAD2iaYSMu2gK-CuxvRdm3Z0xKMKKAWBxpU3qKxuKSr7AYD63LaYOrFo4uyWxK1NnTHGglc\_aJxCVjM9dZmih8g0PNSLWSVGLxYpFcjSauHDqpJwCJxqBytSmKBxwdxyc\_mU5WnUbAO4-jg0F8b5A8f\_MNhZP064fkeRgMEYlCyGndEhy5AD5Yk35LXMhRUAapR2CYFUQN3WirHL12EPuTyhYsF2Wzq4UB5jNGt77redSr3lzLXc0TCIq2QgAGYB9CQ&post\_logout\_redirect\_uri=http://ssodemoservice01:8080 HTTP/1.1*

*Host: keycloak.online:9080*

**Explanation:**

URL Parameters:

* *id\_token\_hint: The JWT token previously issued by the authorization server, representing the current session to be logged out.*
* *post\_logout\_redirect\_uri: The URL to which the user is redirected after logout.*

1. **API get access-token**

**Request Details:**

* *Method: POST*
* *URL: /realms/{realm}/protocol/{protocol}/token*
* *HTTP Version: HTTP/1.1*
* *Headers:*
  + *Accept: application/json;charset=UTF-8*
  + *Content-Type: application/x-www-form-urlencoded;charset=UTF-8*
  + *Authorization: Basic <encode of clientKey:clientSecretKey>*
  + *User-Agent: Java/11.0.22*
  + *Host: localhost:9080*
  + *Connection: keep-alive*
  + *Content-Length: 219*
* *Body*
  + *code: The authorization code received from the authorization server.*
  + *client\_id: The client identifier issued to the client during the registration process.*
  + *client\_secret: The client secret issued to the client during the registration process.*
  + *grant\_type: The type of grant being used, which in this case is authorization\_code.*
  + *redirect\_uri: The registered callback-uri when redirect to login-endpoint*

**Example Request:**

* *POST /realms/jhipster/protocol/openid-connect/token HTTP/1.1*
* *Accept: application/json;charset=UTF-8*
* *Content-Type: application/x-www-form-urlencoded;charset=UTF-8*
* *Authorization: Basic d2ViX2FwcF9kZXY6d2ViX2FwcF9kZXY=*
* *User-Agent: Java/11.0.22*
* *Host: localhost:9080*
* *Connection: keep-alive*
* *Content-Length: 219*
* *code=b3b495a9-665b-439c-9c28-7e523e8e35bd.cefe12dc-75ce-4058-881e-9657250e1278.6e8deddb-b4d6-4e2e-b389-b397d3f74fcd&client\_id=ssodemoservice01 &client\_secret=ssodemoservice01&grant\_type=authorization\_code*

**Response:**

*{*

*"access\_token": <jwt from authorize service>,*

*"expires\_in": <expired in minute>,*

*"refresh\_expires\_in": 0,*

*"refresh\_token": <refresh token for request a new access token when the old one is expired>,*

*"token\_type": "Bearer",*

*"id\_token": <id-token, using for logout endpoint>,*

*"not-before-policy": 1718801705,*

*"session\_state": < An opaque value used to maintain state between the request and the callback.>,*

*"scope": "openid email profile offline\_access"*

*}*

1. **API get user-info**

**Request Details:**

* *Method: GET*
* *URL: /realms/{realm}/protocol/{protocol}/*userinfo
* *HTTP Version: HTTP/1.1*
* *Headers:*
  + *Accept: application/json;charset=UTF-8*
  + *Content-Type: application/x-www-form-urlencoded;charset=UTF-8*
  + *Authorization: Bearer <access\_token>*
  + *User-Agent: Java/11.0.22*
  + *Host: localhost:9080*
  + *Connection: keep-alive*

**Example Request:**

* *GET /realms/jhipster/protocol/openid-connect/* *userinfo HTTP/1.1*
* *Accept: application/json;charset=UTF-8*
* *Content-Type: application/x-www-form-urlencoded;charset=UTF-8*
* *Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI4cGJRSnljZHZJc1FrRzJybElNUFlLNFFNeXdYeW02Wk5meGJkQlpwUGp3In0..TxmmSILWGCnoWDRN5bKohjQTIJEkf8ITNThXbeidTEBHLgvBxVhRlA67lSqEH2bvYYaSLEI5DkhSK7tERjzHJk2aMLTqzQakeDC1jks8DQqFLgIKH5\_3ucEcbNZMrlnRZBDPsx1IHNvUJdKWkKWh9BjAy89VXsCfOFOMzQmsn5olKR5T5tvb6fymOkn3jffX21F48bashNQj62UsH5hylh4CmbZbJ0t3Uu-syQQJs568XK6nq7AJZ8o4fgHEHrPzGRYIAeEJ1o0wPZXqKwbMb8J3onWFv1GXpRiZ8T8ZSUKla\_O2dRCLFHYZCy8bz2\_4eJ2ipaIx9pjpnOQqRGc1xg*
* *User-Agent: Java/11.0.22*
* *Host: localhost:9080*
* *Connection: keep-alive*

**Response:** (the information of loging-in user from authorize-server)

*{*

*"sub": "4c973896-5761-41fc-8217-07c5d13a004b",*

*"email\_verified": true,*

*"roles": [*

*"ROLE\_USER",*

*"offline\_access",*

*"ROLE\_ADMIN",*

*"uma\_authorization"*

*],*

*"name": "Admin Administrator",*

*"preferred\_username": "admin",*

*"given\_name": "Admin",*

*"family\_name": "Administrator",*

*"email": "admin@localhost"*

*}*

1. **Conclusion**
2. **Missing points in the document:**

* Does not clearly specify the standards applied in the project such as: OAuth 2.0, OpenID Connect, SAML
* Lacks workflows for SSO login on mobile devices
* Lacks comparison with other centralized authentication frameworks
* The child service application is currently using the default stateful mechanism (using session-id)

1. **Upcoming Content (if the project is approved)**

* Support authentication according to the stateless mechanism
* Detail the mechanisms that Keycloak supports