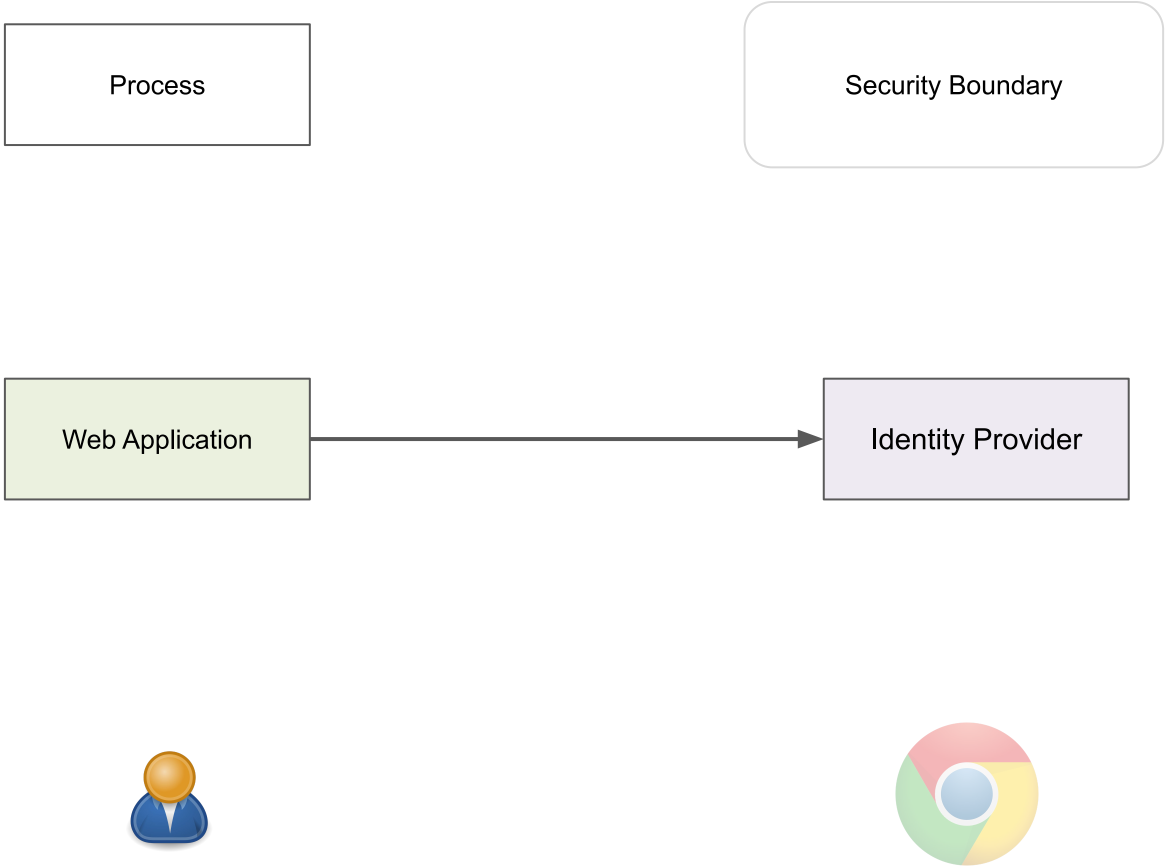
5G
5G

Architecture Diagrams

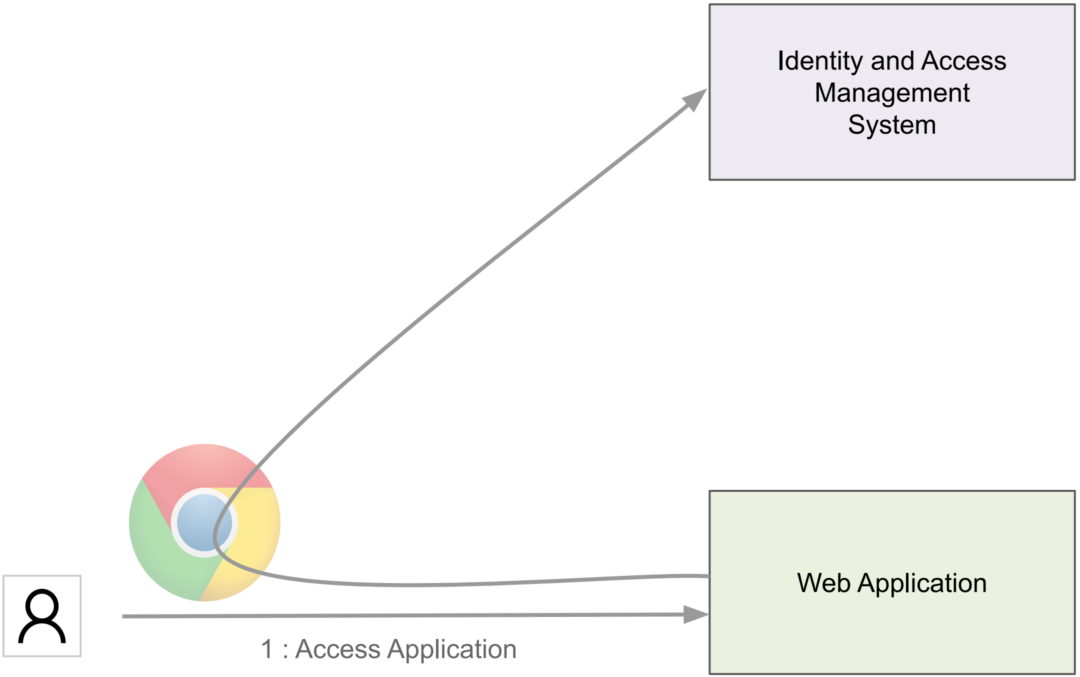


Process

Security Boundary

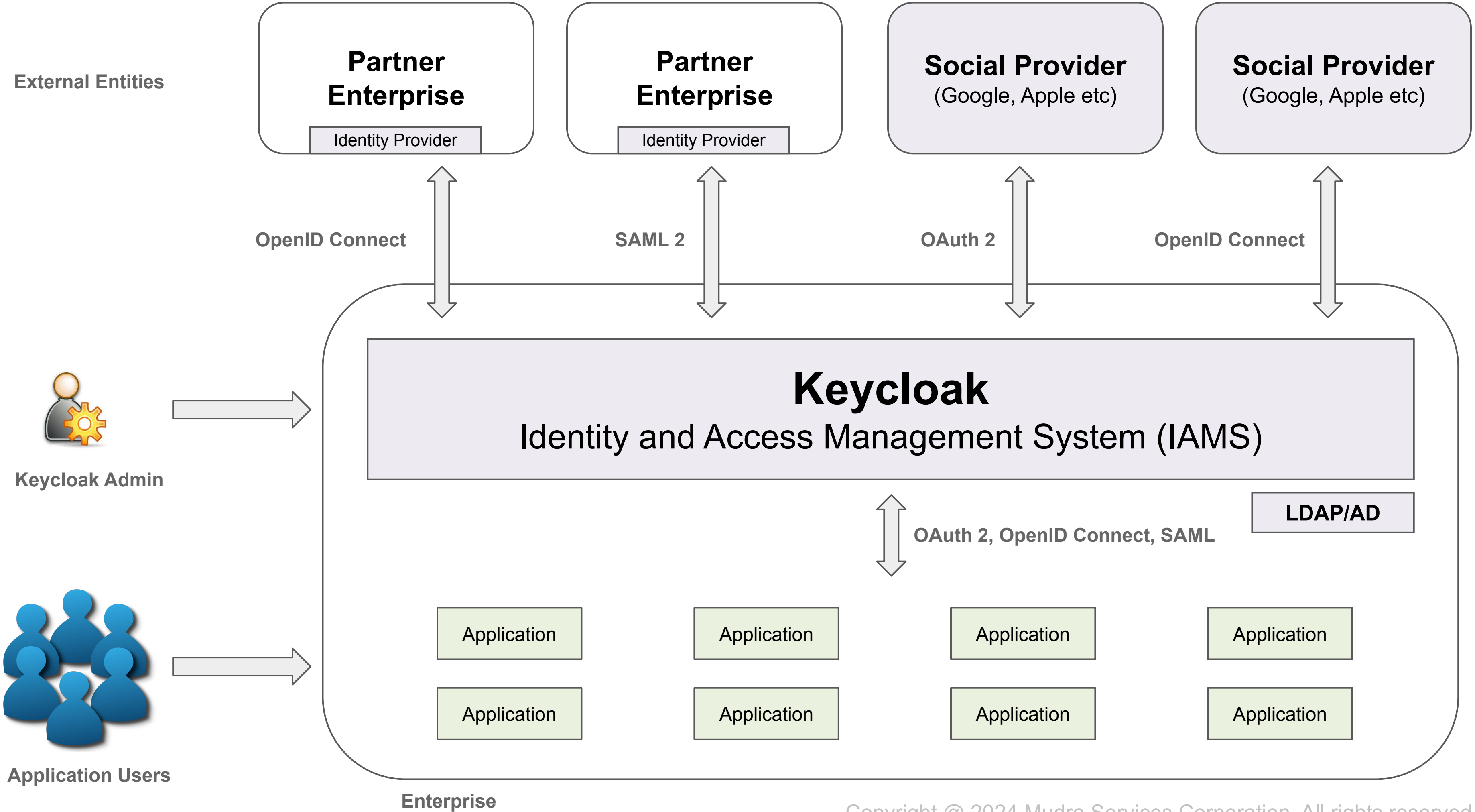
Web Application

Identity Provider



Identity and Access Management System
Web Application
1 : Access Application

Keycloak Architecture



External Entities

Partner Enterprise

Identity Provider

Partner Enterprise

Identity Provider

Social Provider
(Google, Apple etc)

Social Provider
(Google, Apple etc)

OpenID Connect

SAML 2

OAuth 2

OpenID Connect

Keycloak Admin

Application Users

Keycloak

Identity and Access Management System (IAMS)

LDAP/AD

OAuth 2, OpenID Connect, SAML

Application

Application

Application

Application

Application

Application

Application

Application

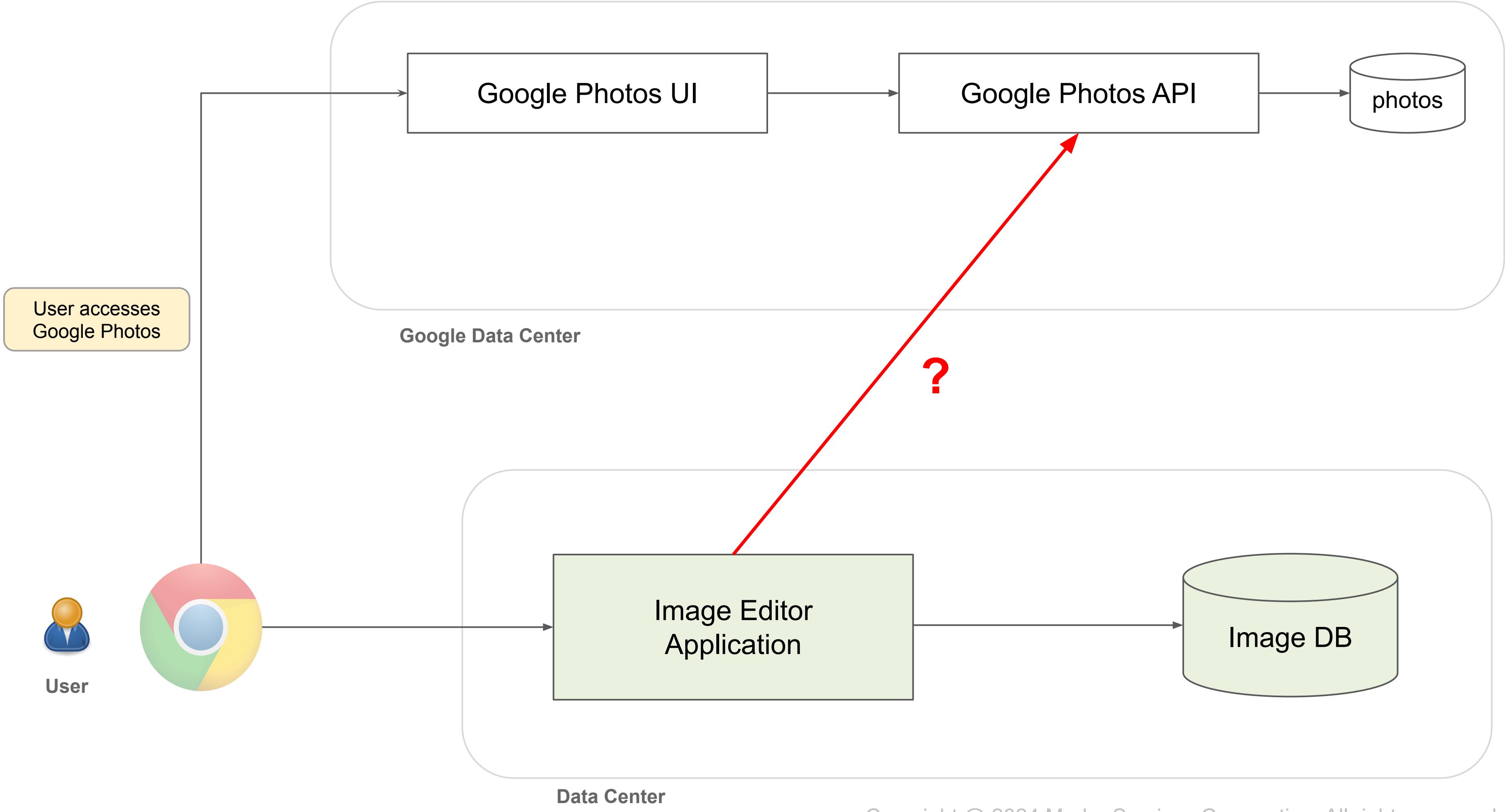
Enterprise

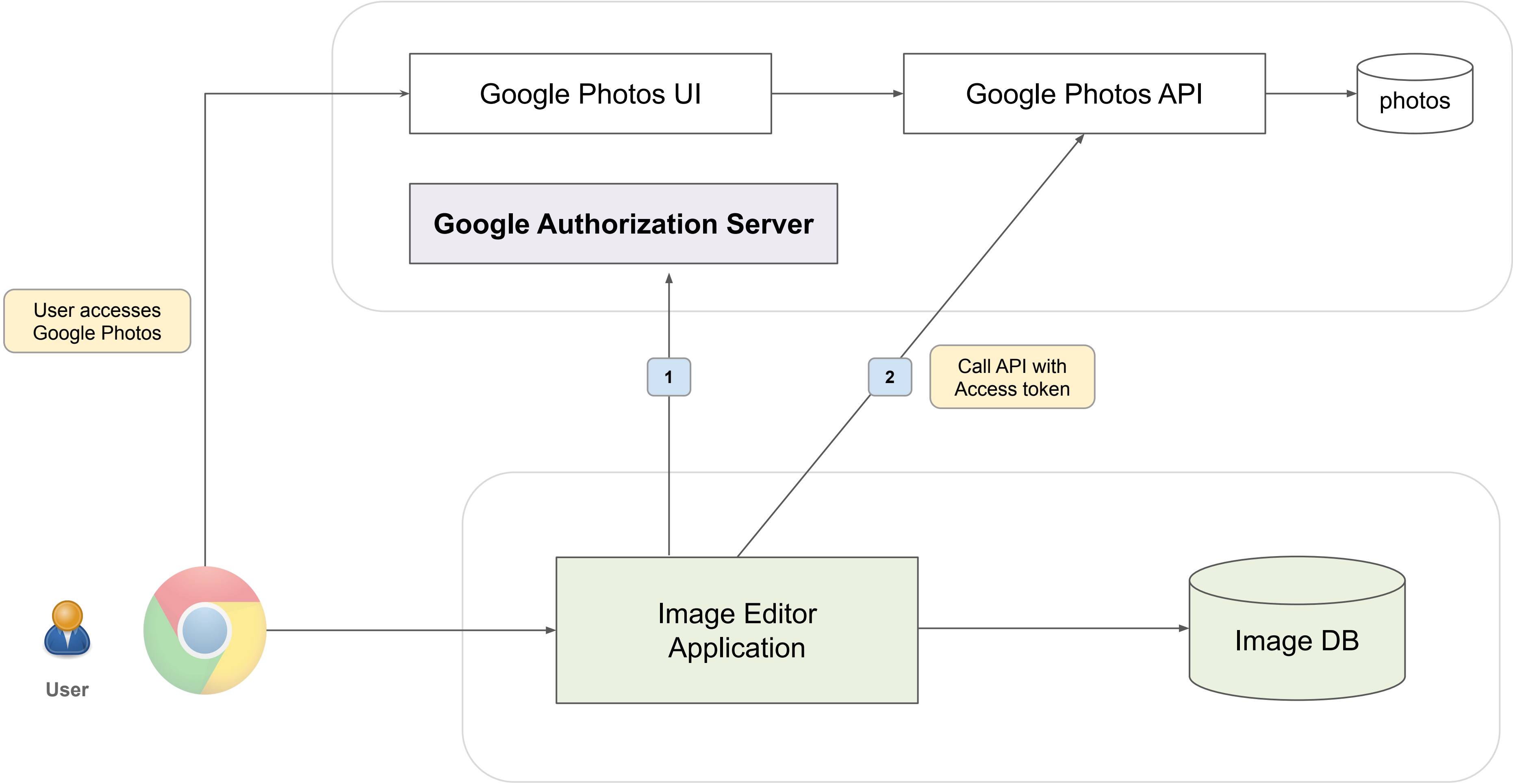
Keycloak Features

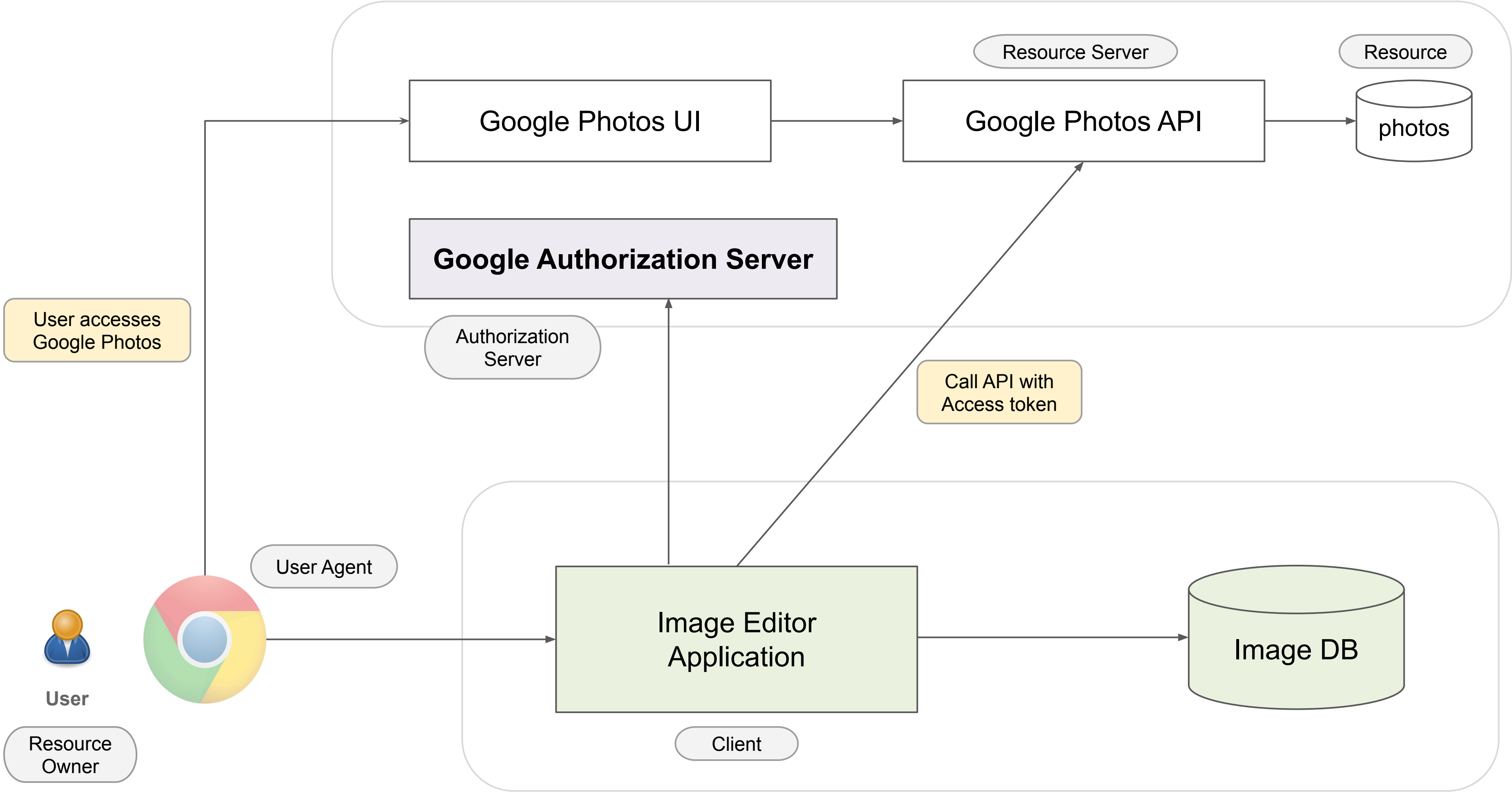
- Single Sign-On
- Identity Brokering
- Social Login - **Facebook, Google etc**
- Centralized Administration Console
- Standard Protocols - **SAML 2.0, OAuth 2.0, OpenID Connect**
- LDAP and Active Directory Integration
- MFA
- Fine grained Authorization Services

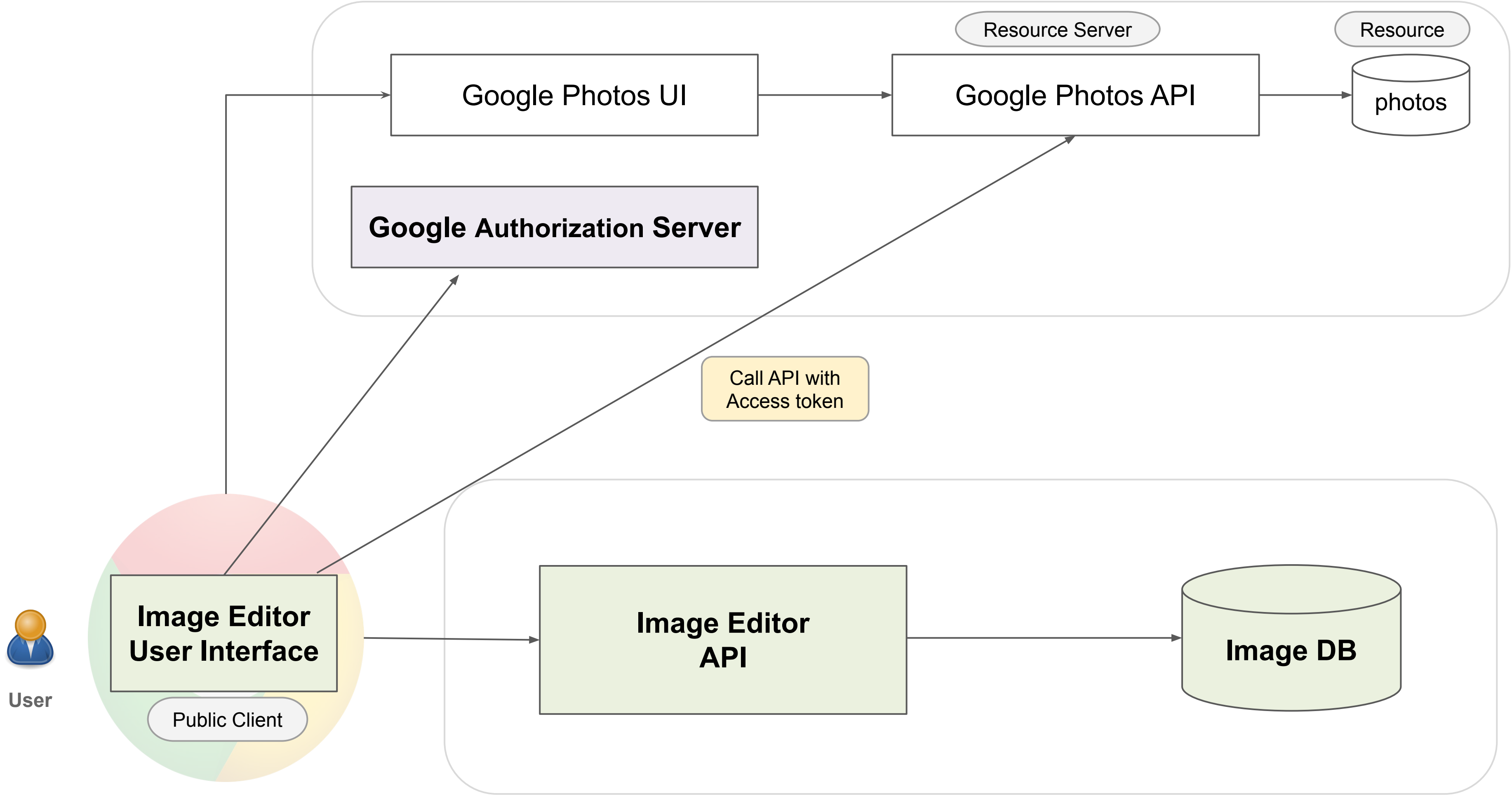
OAuth and OpenID Connect

OAuth 2.0 Actors

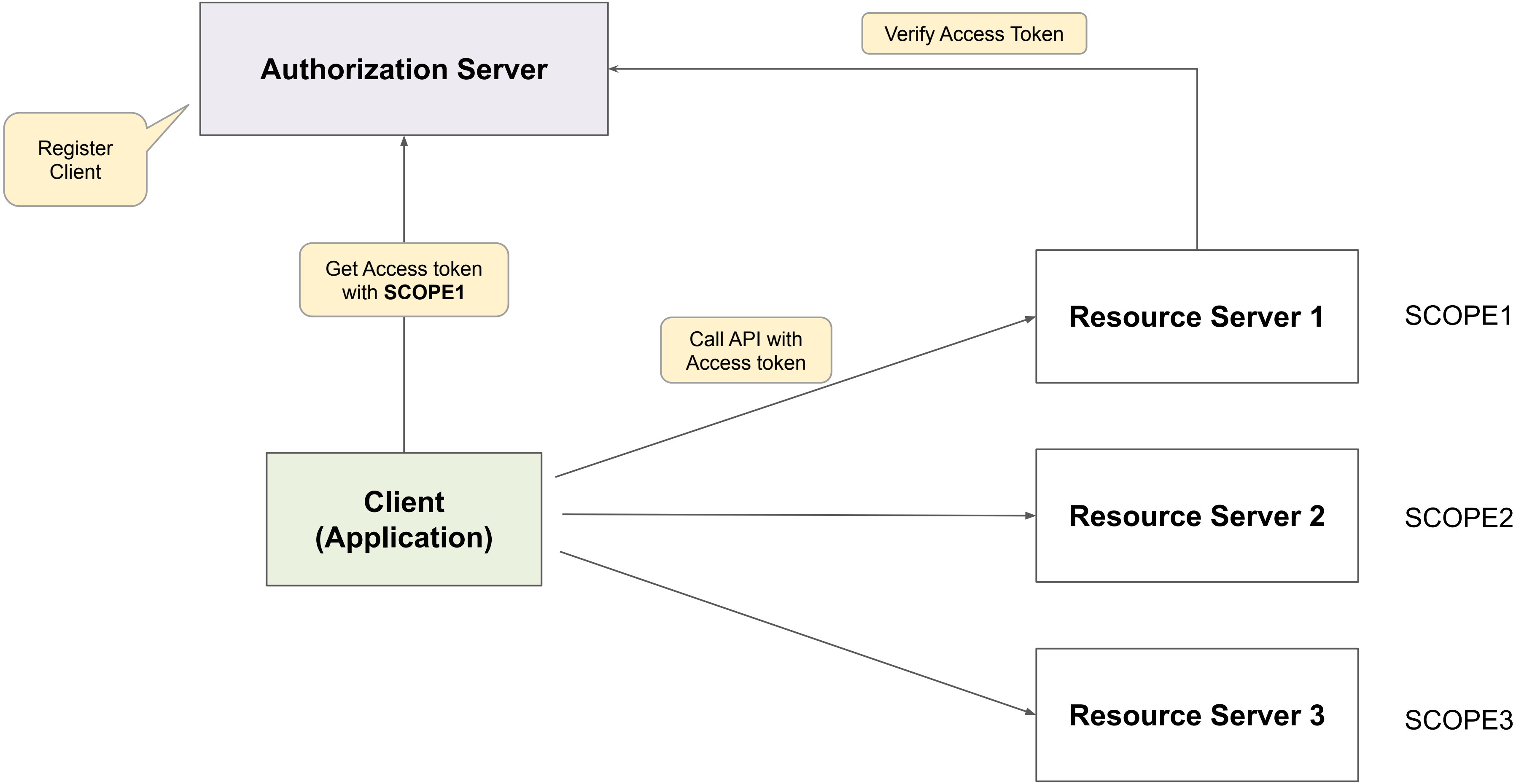

Google Photos UI
Google Photos API
photos
Google Data Center
User accesses Google Photos
?
User
Image Editor Application
Image DB
Data Center

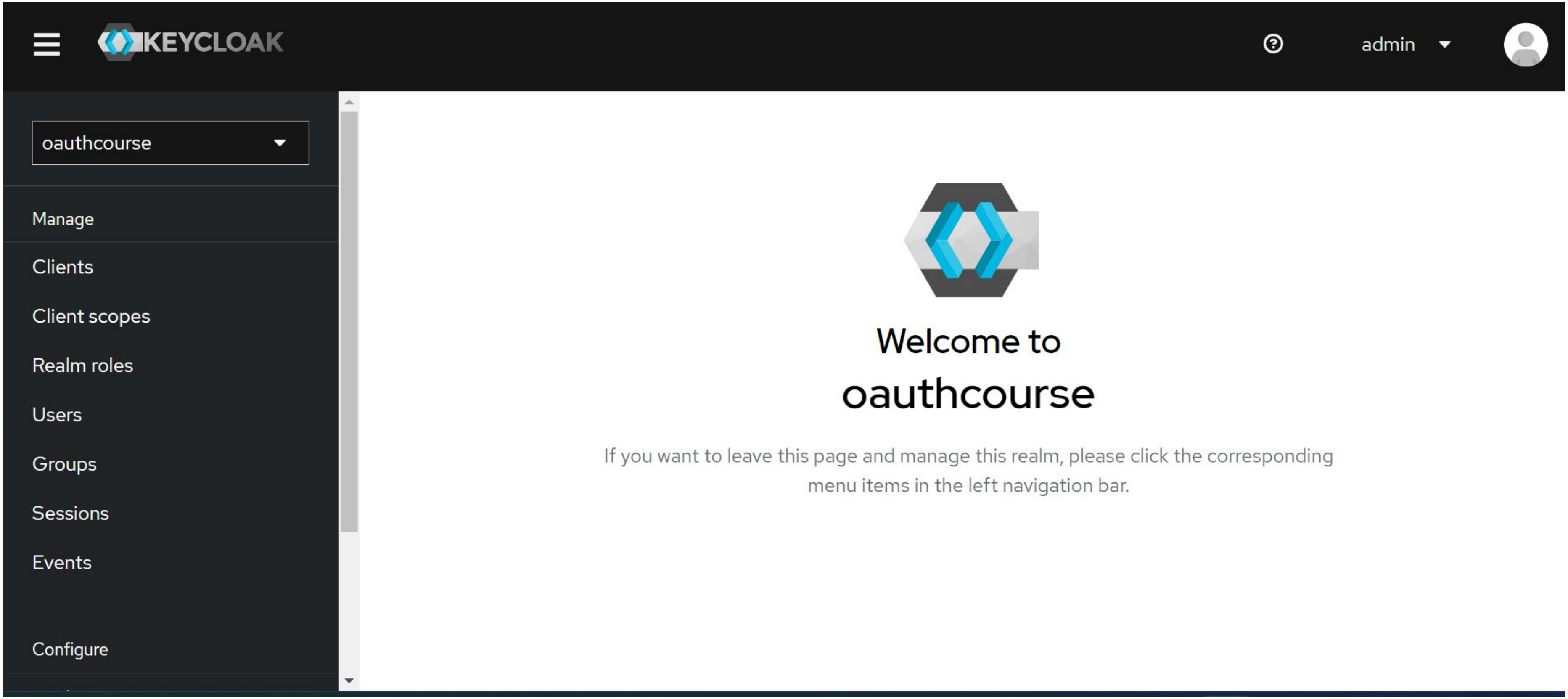

Google Photos UI
Google Photos API
photos
Google Authorization Server
User accesses Google Photos
1
2
Call API with Access token
Image Editor Application
Image DB
User


Resource Server
Resource
Google Photos UI
Google Photos API
photos
Google Authorization Server
User accesses Google Photos
Authorization Server
Call API with Access token
User Agent
Image Editor Application
Image DB
User
Resource Owner
Client


Resource Server
Resource
Google Photos UI
Google Photos API
photos
Google Authorization Server
Call API with
Access token
Image Editor
User Interface
Public Client
User
Image Editor
API
Image DB

Clients and Scopes


Authorization Server
Register Client
Verify Access Token
Get Access token with **SCOPE1**
Call API with Access token
Client (Application)
Resource Server 1
SCOPE1
Resource Server 2
SCOPE2
Resource Server 3
SCOPE3

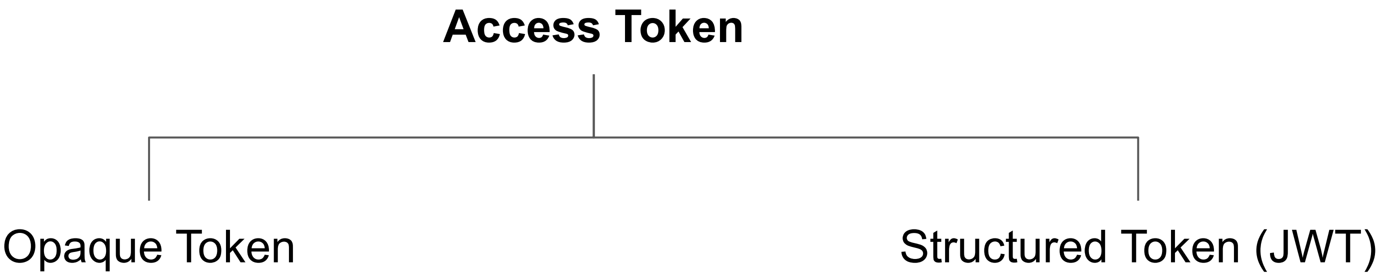


Welcome to

oauthcourse

If you want to leave this page and manage this realm, please click the corresponding menu items in the left navigation bar.

Access Tokens



Access Token

Opaque Token

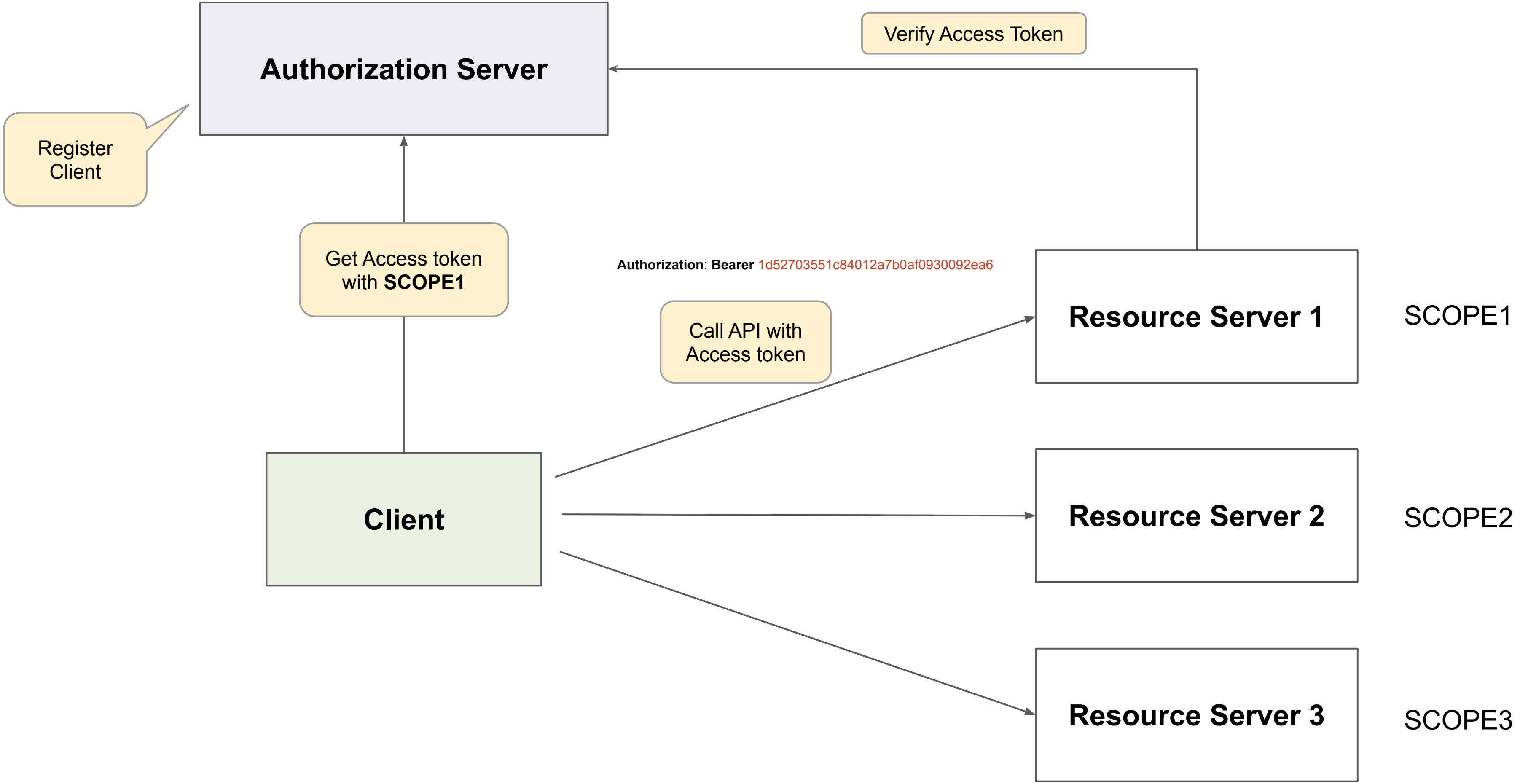
Structured Token (JWT)

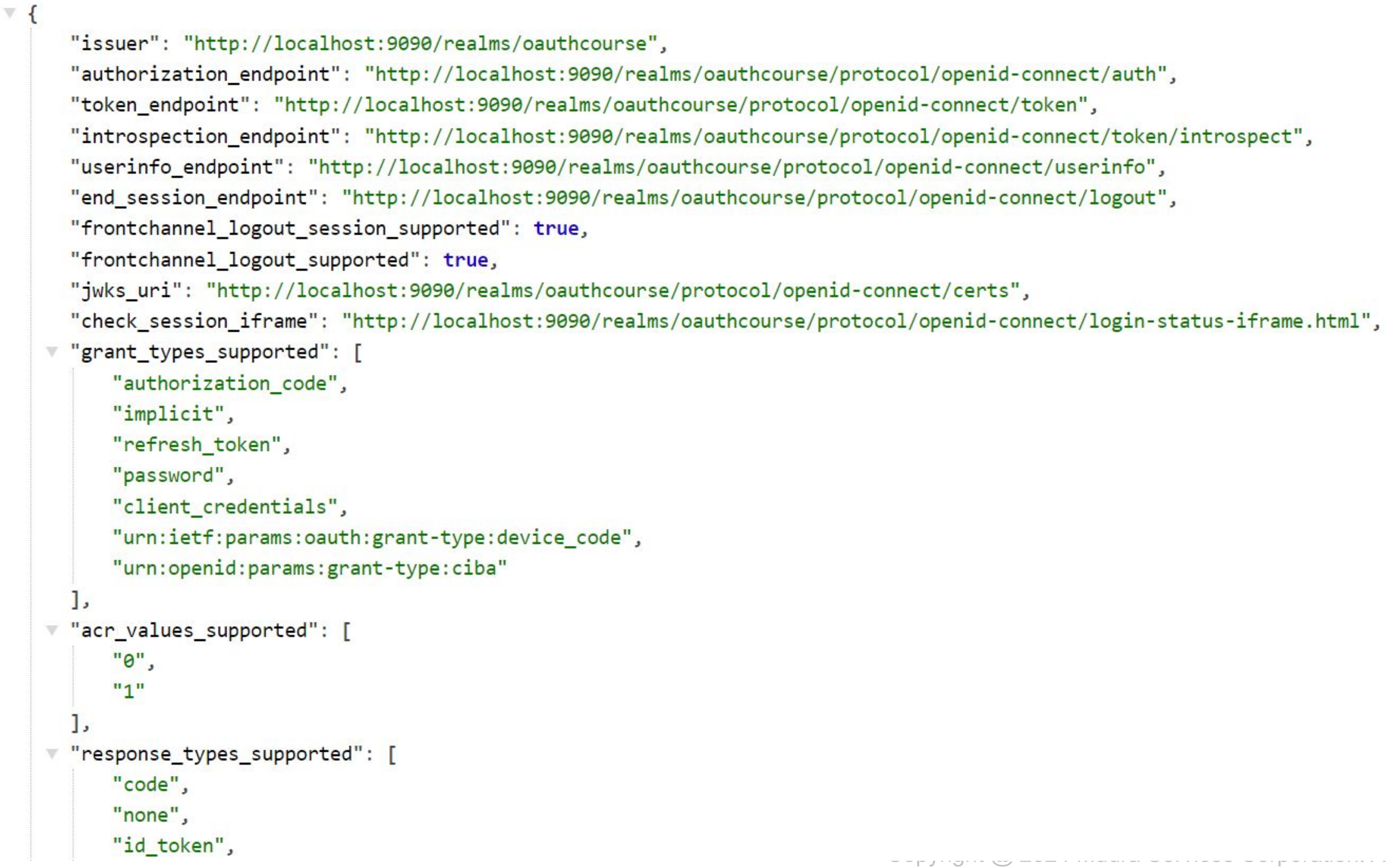
```
1d52703551c84012a7b0af0930092ea6
```

```
// THIS IS THE BODY OF JWT WITHOUT HEADER AND SIGNATURE
{
    "exp" : 1700941502,
    "iat" : 1700941202,
    "auth_time" : 1700941202,
    "jti" : "47f9ecbc-f221-4228-acd9-df15f604cb5a",
    "iss" : "http://127.0.0.1:9090/realms/oauthcourse",
    "sub" : "2fc9115f-75c8-4eea-b1d3-3edd0f3598c2",
    "typ" : "Bearer",
    "azp" : "bugtracker",
    "nonce" : "jvDZ6MnJ7NHJNY2LnuwEXoExnFnppd29ggGa50G0a3c",
    "acr" : "1",
    "scope" : "openid profile bugtracker.admin email",
    "sid" : "15597759-7c5e-4941-945d-629ddf607403",
    "email_verified" : false,
    "name" : "John Doe",
    "preferred_username" : "johndoe",
    "given_name" : "John",
    "family_name" : "Doe"
}
```

Sending Access Token in HTTP

```
GET /api/v4/projects?owned=true HTTP/1.1
Authorization Bearer e4f27418e13792f6a652432....
Host gitlab.com
User-Agent HTTPie
```

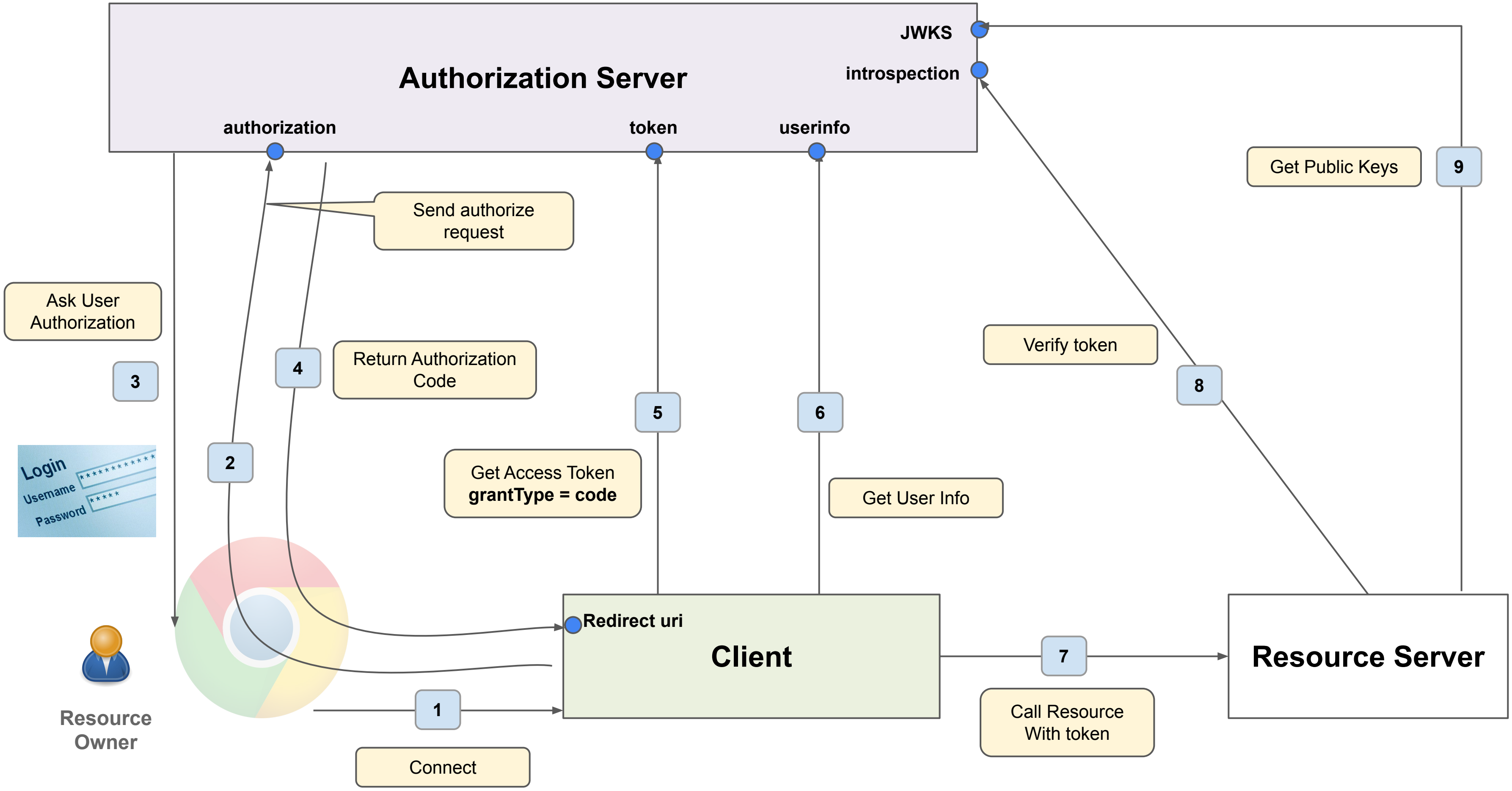

Authorization Server
Verify Access Token
Register Client
Get Access token with **SCOPE1**
Authorization: Bearer 1d52703551c84012a7b0af0930092ea6
Call API with Access token
Resource Server 1
SCOPE1
Client
Resource Server 2
SCOPE2
Resource Server 3
SCOPE3



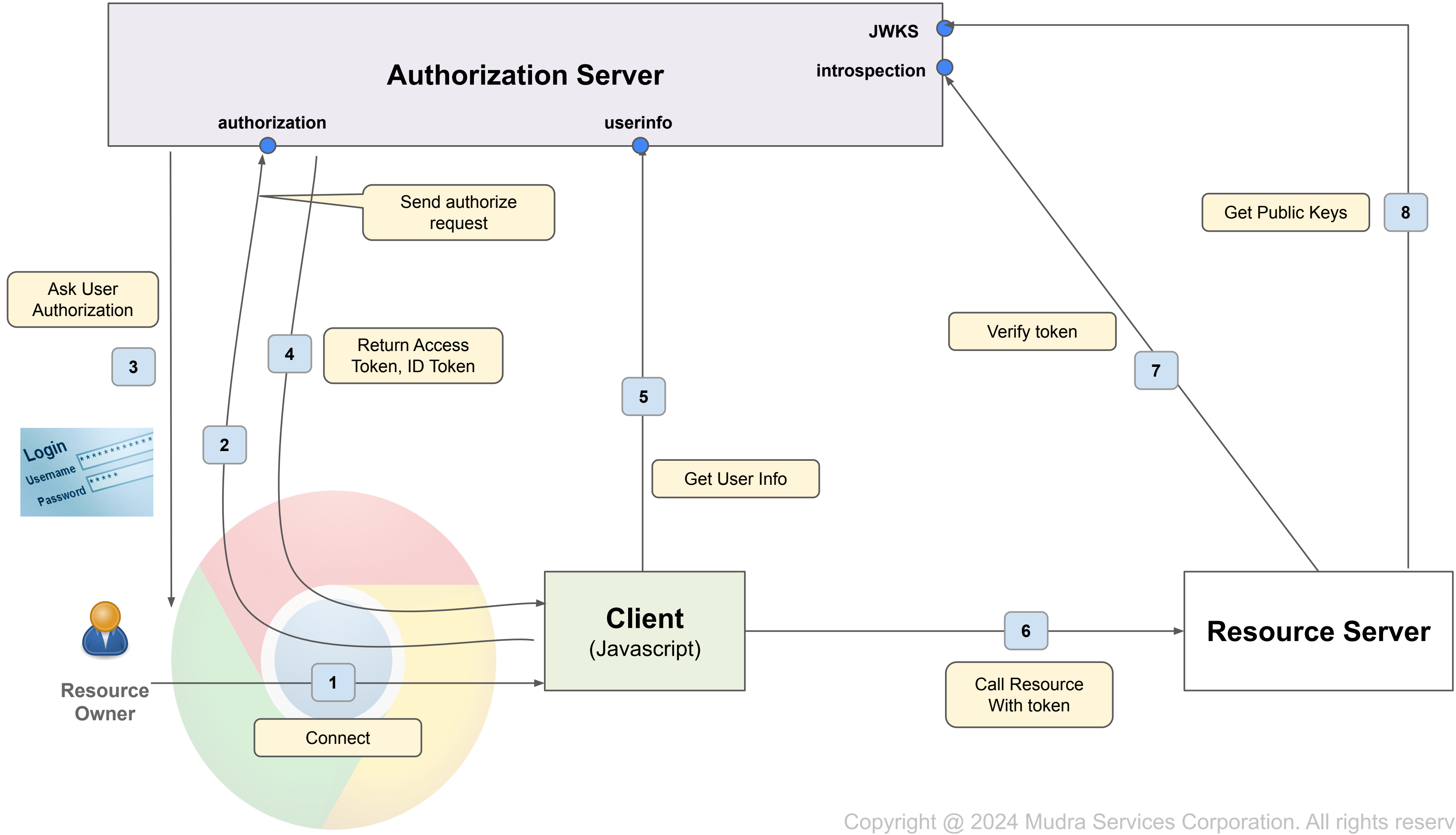
```
{
  "issuer": "http://localhost:9090/realms/oauthcourse",
  "authorization_endpoint": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/auth",
  "token_endpoint": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/token",
  "introspection_endpoint": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/token/introspect",
  "userinfo_endpoint": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/userinfo",
  "end_session_endpoint": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/logout",
  "frontchannel_logout_session_supported": true,
  "frontchannel_logout_supported": true,
  "jwks_uri": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/certs",
  "check_session_iframe": "http://localhost:9090/realms/oauthcourse/protocol/openid-connect/login-status-iframe.html",
  "grant_types_supported": [
    "authorization_code",
    "implicit",
    "refresh_token",
    "password",
    "client_credentials",
    "urn:ietf:params:oauth:grant-type:device_code",
    "urn:openid:params:grant-type:ciba"
  ],
  "acr_values_supported": [
    "0",
    "1"
  ],
  "response_types_supported": [
    "code",
    "none",
    "id_token",
```

OAuth Grant Types

- Authorization Code
- Authorization Code with PKCE
- ~~Implicit (Deprecated)~~
- Refresh Token
- Client Credentials
- Device Code
- ~~Password (Deprecated)~~


JWKS
Authorization Server
introspection
authorization
token
userinfo
Send authorize request
Get Public Keys
9
Ask User Authorization
3
4
Return Authorization Code
Verify token
8
5
6
Login
Username
Password
2
Get Access Token
grantType = code
Get User Info
Redirect uri
Client
7
Resource Server
Resource Owner
1
Call Resource With token
Connect

~~Implicit Grant Type (Deprecated)~~



Authorization Server
JWKS
introspection
authorization
userinfo
Send authorize request
Ask User Authorization
3
4
Return Access Token, ID Token
5
Get User Info
Get Public Keys
8
Verify token
7
2
Login
Username
Password
Resource Owner
1
Connect
Client
(Javascript)
6
Call Resource With token
Resource Server

OAuth 2.0 - PKCE Extension

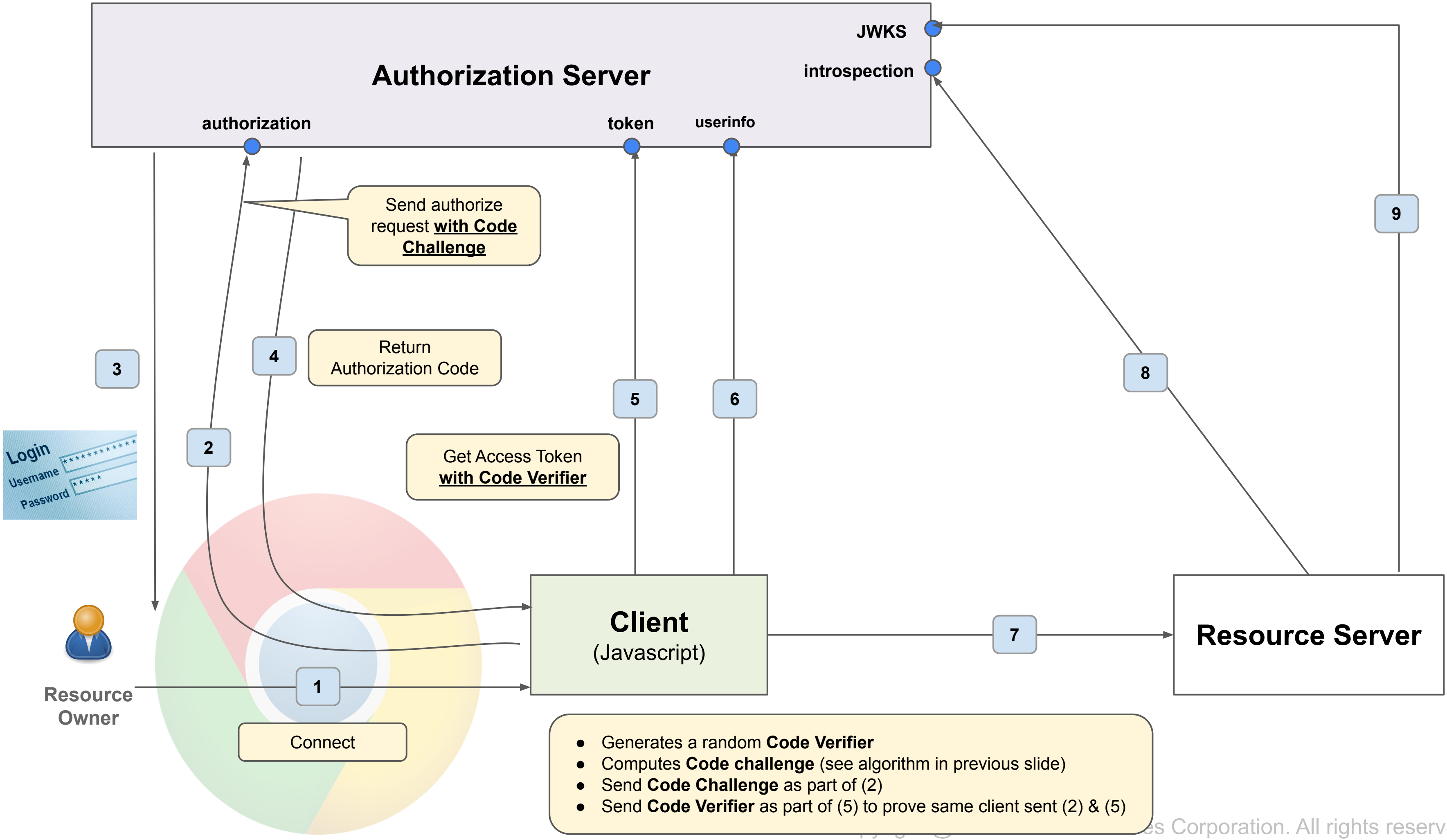
- Proof Key For Code Exchange
- Extension of the Authorization Code grant type
- Usually used by public clients

- Generate a **code_verifier**
  - Min = 43 chars and Max = 128 chars
  - Random and impractical to guess

- Send **code_challenge** with authorize request

```
code_challenge = BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))
```

- Send **code_verifier** with token request for grant type = code
  - That’s the proof



Authorization Server

JWKS

introspection

authorization

token

userinfo

1 Connect

2 Send authorize request **with Code Challenge**

3

4 Return Authorization Code

5 Get Access Token **with Code Verifier**

6

7

8

9

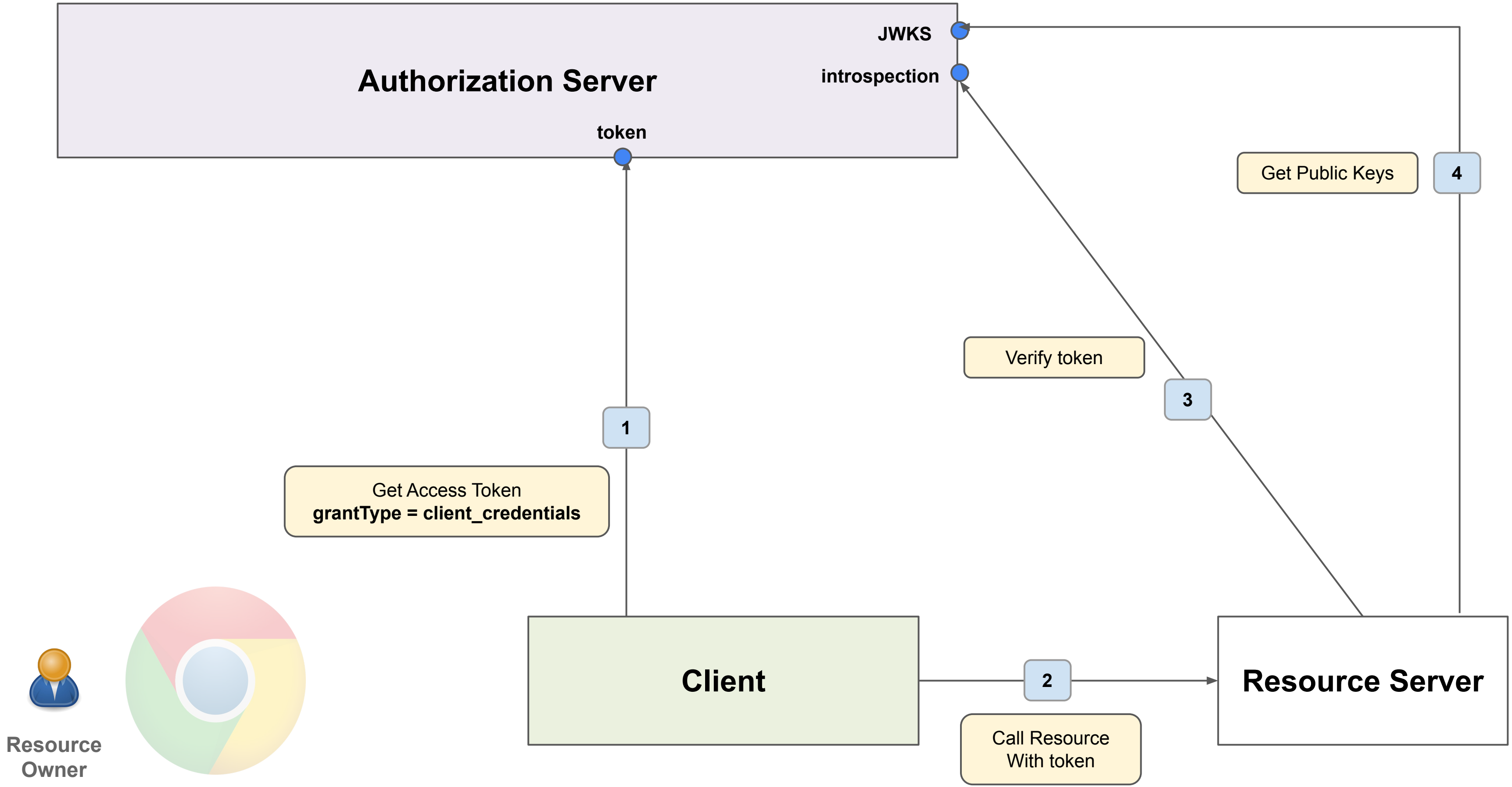
Resource Owner

Client (Javascript)

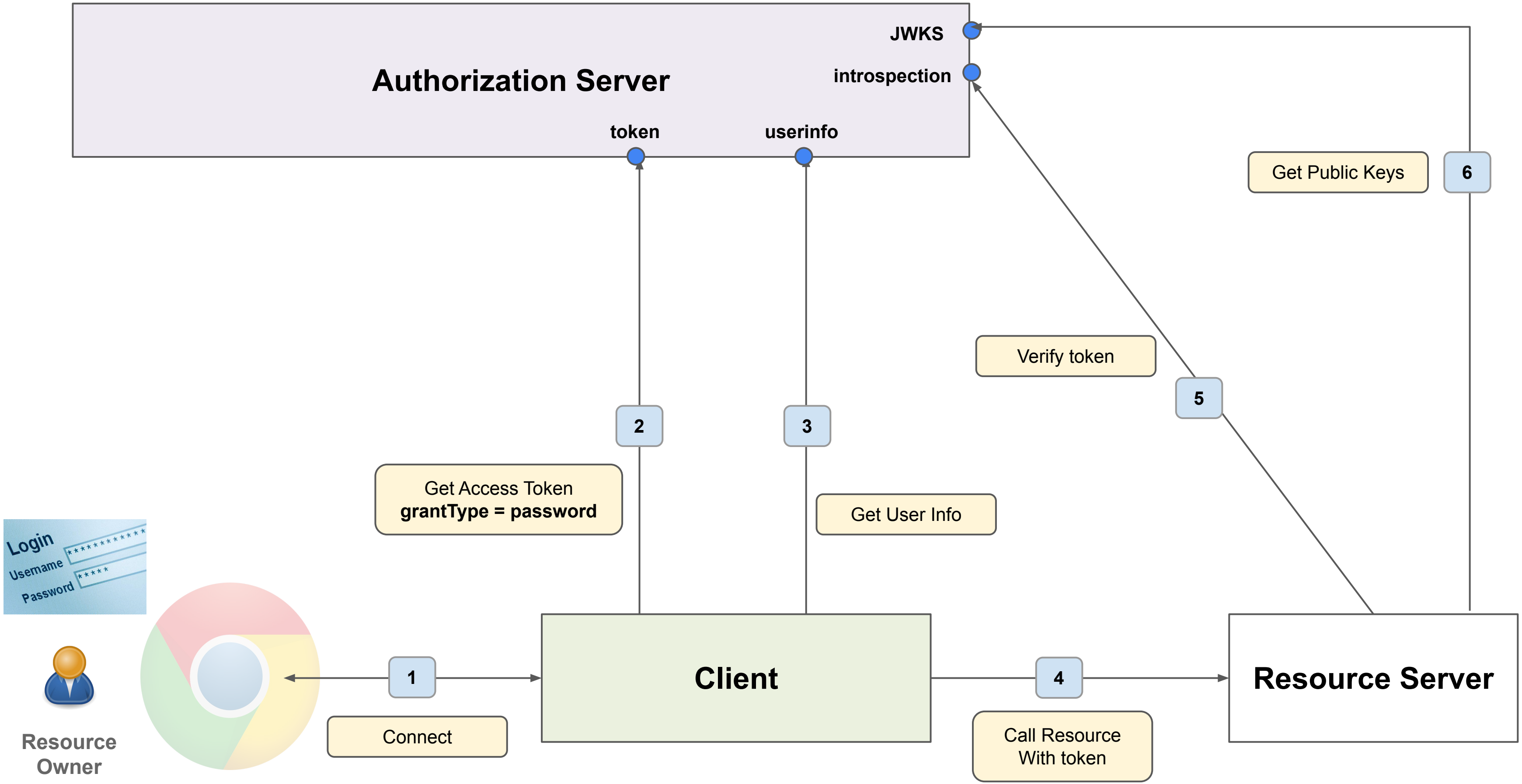
Resource Server

- Generates a random **Code Verifier**
- Computes **Code challenge** (see algorithm in previous slide)
- Send **Code Challenge** as part of (2)
- Send **Code Verifier** as part of (5) to prove same client sent (2) & (5)

Client Credentials Grant Type

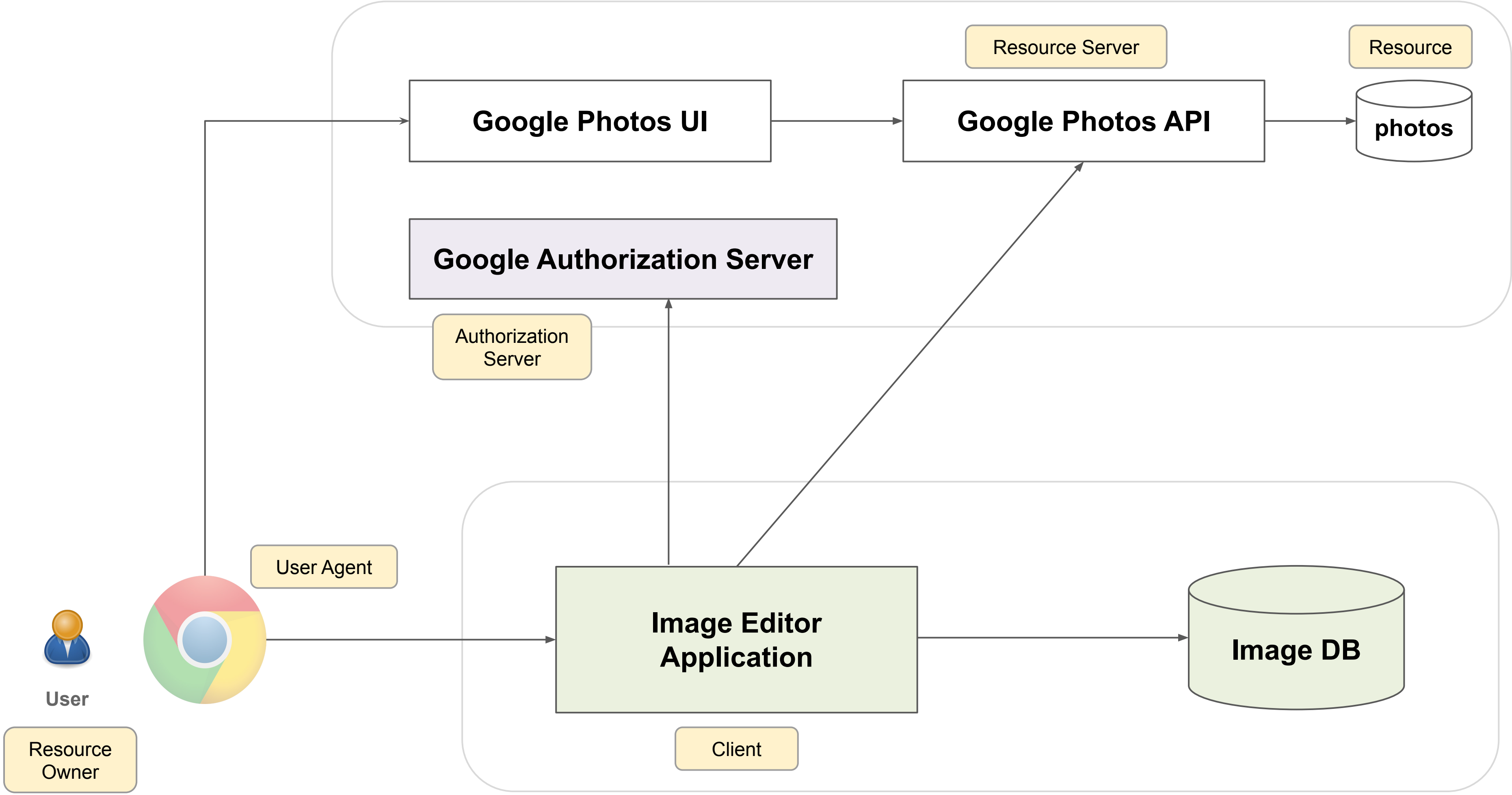

Authorization Server
JWKS
introspection
token
Get Public Keys
4
Verify token
3
1
Get Access Token
grantType = client_credentials
Resource Owner
Client
2
Resource Server
Call Resource
With token

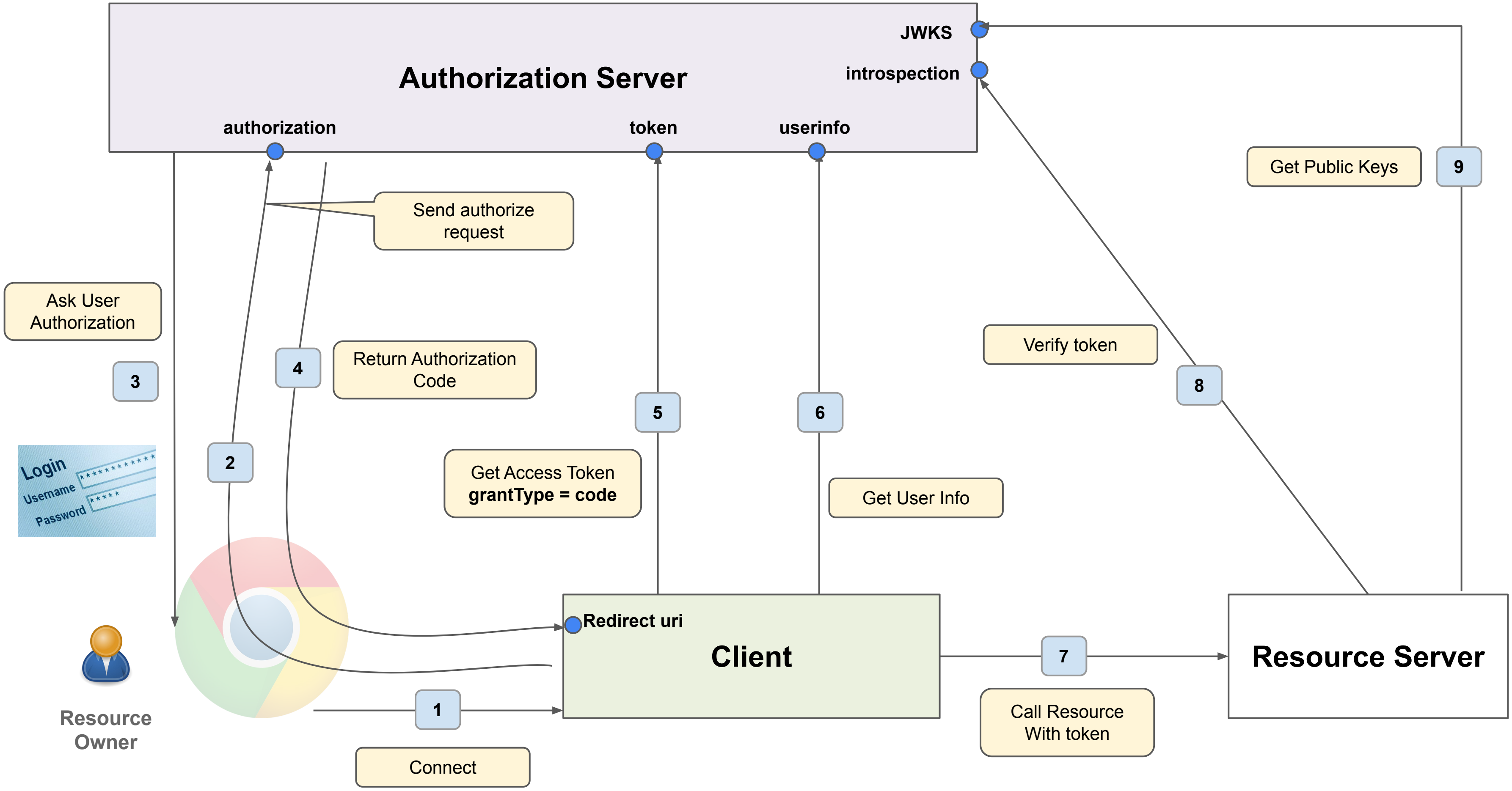
~~Password Grant Type~~



Authorization Server
JWKS
introspection
token
userinfo
Get Public Keys
6
Verify token
5
2
3
Get Access Token
grantType = password
Get User Info
Login
Username
Password
Resource Owner
1
Connect
Client
4
Call Resource
With token
Resource Server

OpenID Connect


Resource Server
Resource
Google Photos UI
Google Photos API
photos
Google Authorization Server
Authorization Server
User Agent
Image Editor Application
Image DB
User
Resource Owner
Client



Authorization Server

JWKS

introspection

authorization

token

userinfo

Send authorize request

Get Public Keys

9

Ask User Authorization

3

4

Return Authorization Code

5

6

Verify token

8

2

Get Access Token
grantType = code

Get User Info

Redirect uri

Client

Resource Server

7

Resource Owner

1

Call Resource With token

Connect

OpenID Scopes and Token

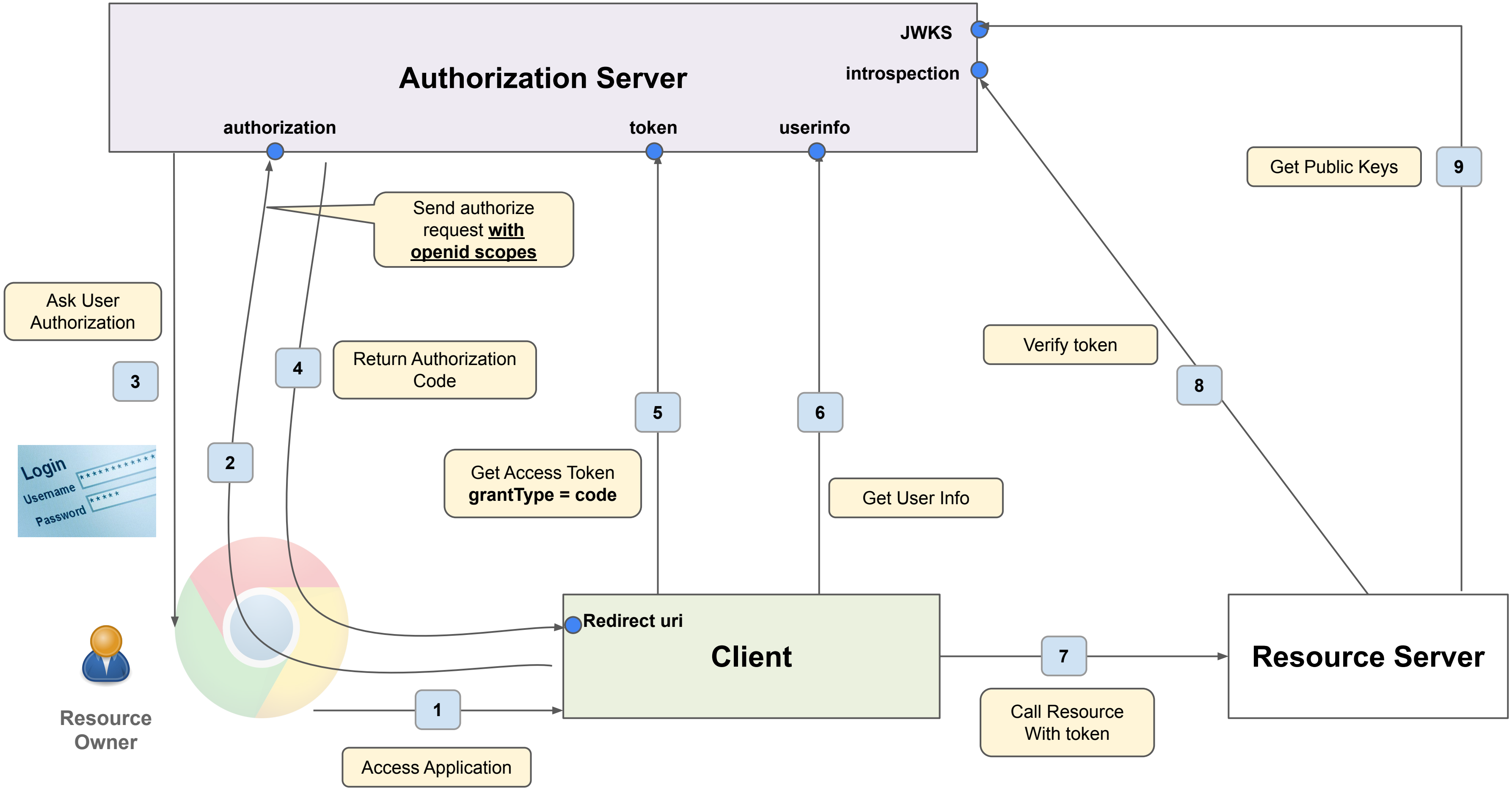
- **openid profile email address phone**
- ID Token contains User information
- **/userinfo** endpoint access

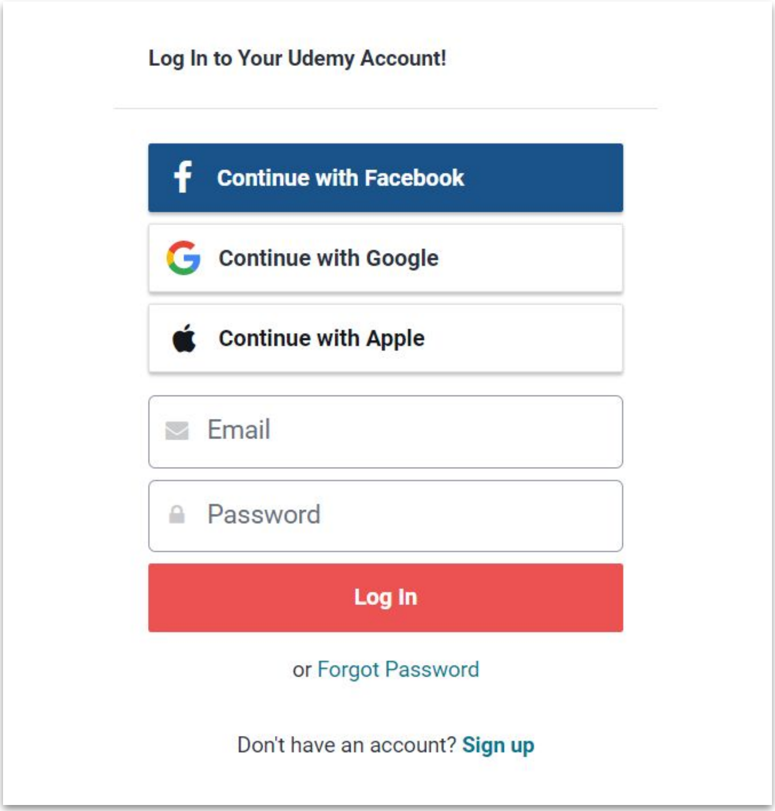
/userinfo Response (scopes : openid profile email)

```
{
    "sub": "00u36oy8fCxxxPWhf5d6",
    "locale": "en-US",
    "zoneinfo": "America/Los_Angeles",
    "name": "Viraj Shetty",
    "preferred_username": "shetty.viraj@gmail.com",
    "given_name": "Viraj",
    "family_name": "Shetty",
    "updated_at": 1627238752,
    "email": "shetty.viraj@gmail.com",
    "email_verified": true
}
```

profile

email


Authorization Server
JWKS
introspection
authorization
token
userinfo
Send authorize request **with openid scopes**
Ask User Authorization
3
4
Return Authorization Code
5
6
8
9
Get Public Keys
Verify token
Login
Username
Password
2
Get Access Token **grantType = code**
Get User Info
Redirect uri
Client
7
Resource Server
Resource Owner
1
Access Application
Call Resource With token

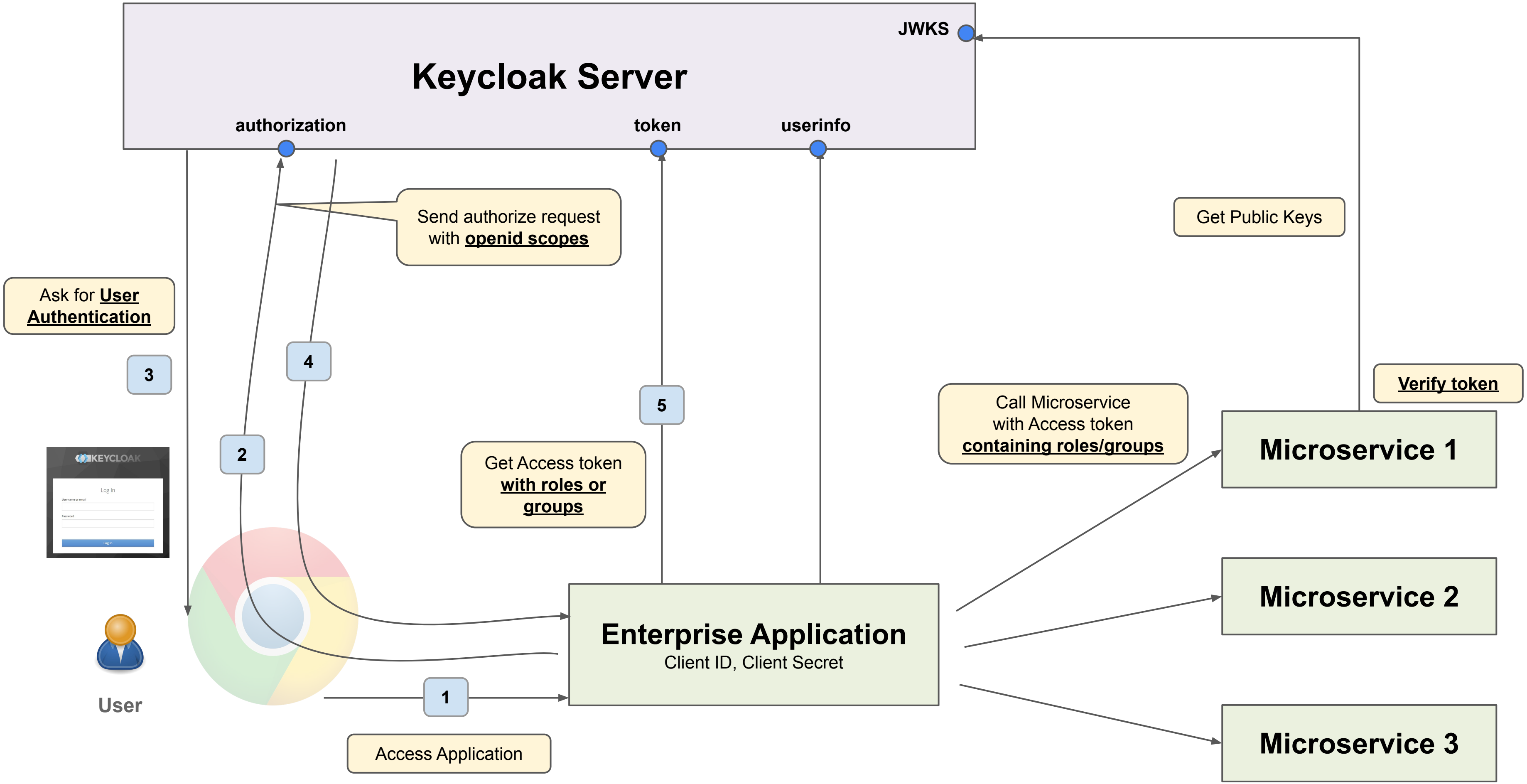


Log In to Your Udemy Account!

or Forgot Password

Don't have an account? **Sign up**

OpenID Connect in an Enterprise



Keycloak Server

JWKS

authorization

token

userinfo

Send authorize request with **openid scopes**

Ask for **User Authentication**

Get Public Keys

Verify token

1 Access Application

2

3

4

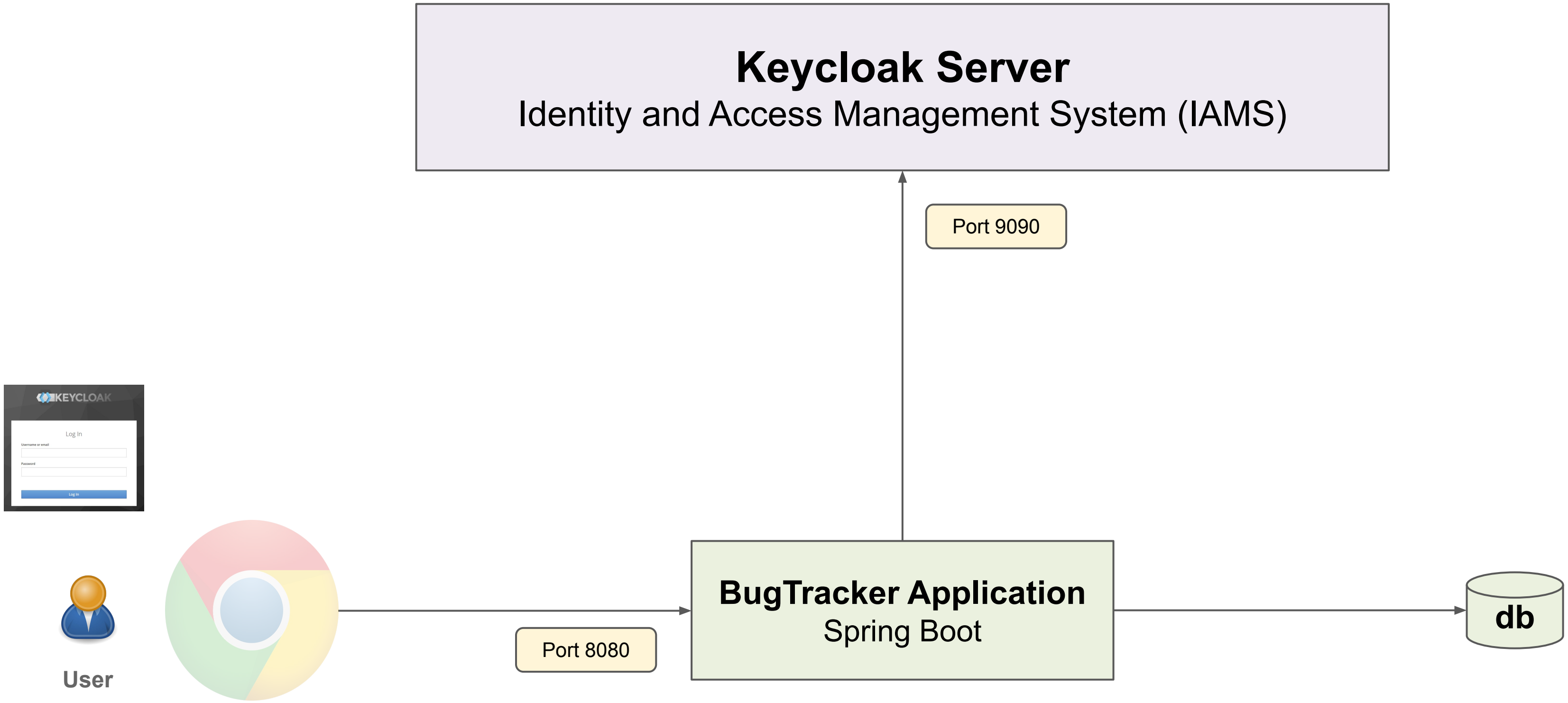
5

Get Access token **with roles or groups**

Call Microservice with Access token **containing roles/groups**

OpenID Connect with Scopes

Desktop



Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

User

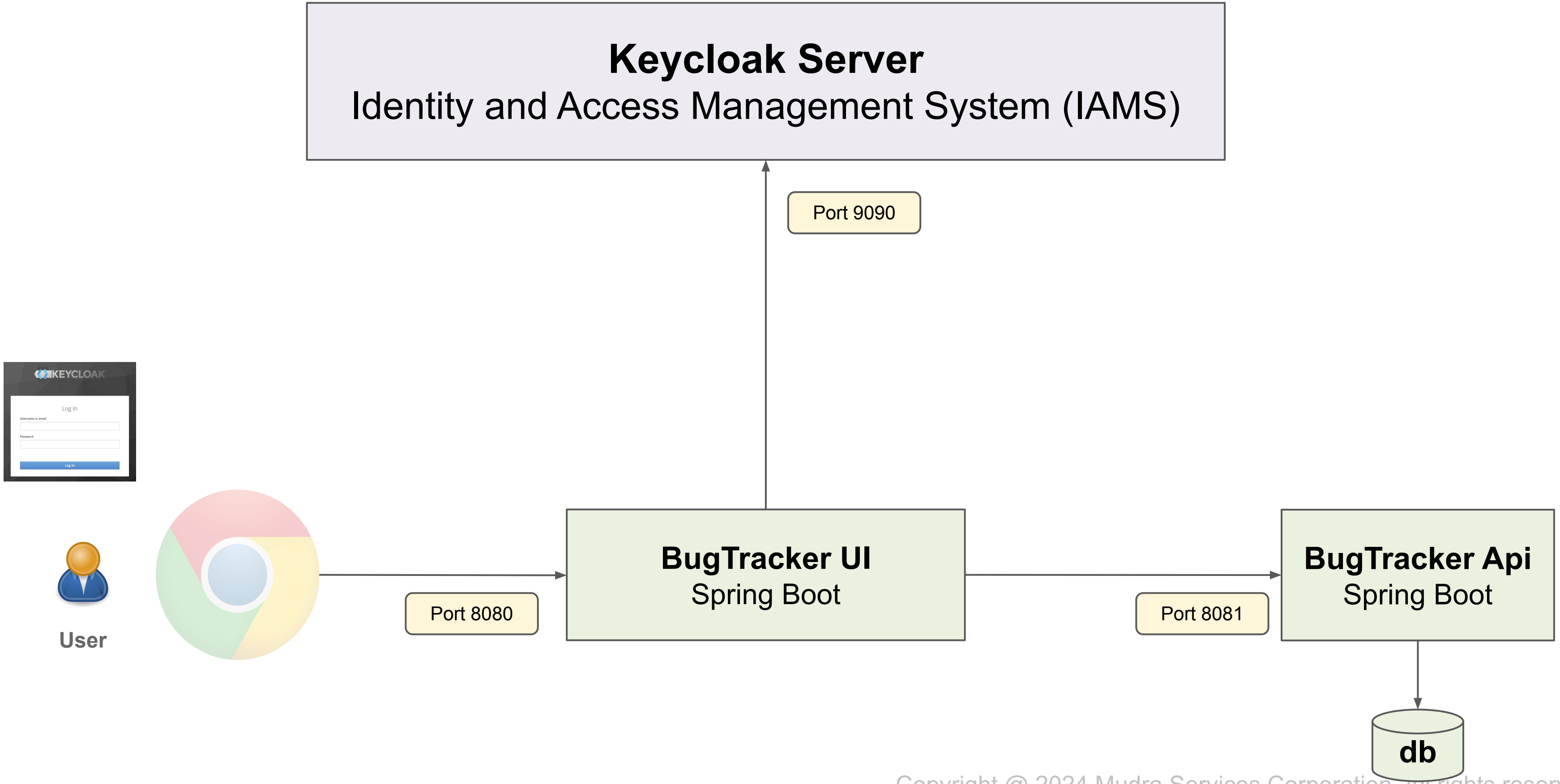
Port 8080

BugTracker Application

Spring Boot

db

Desktop



Keycloak Server
Identity and Access Management System (IAMS)

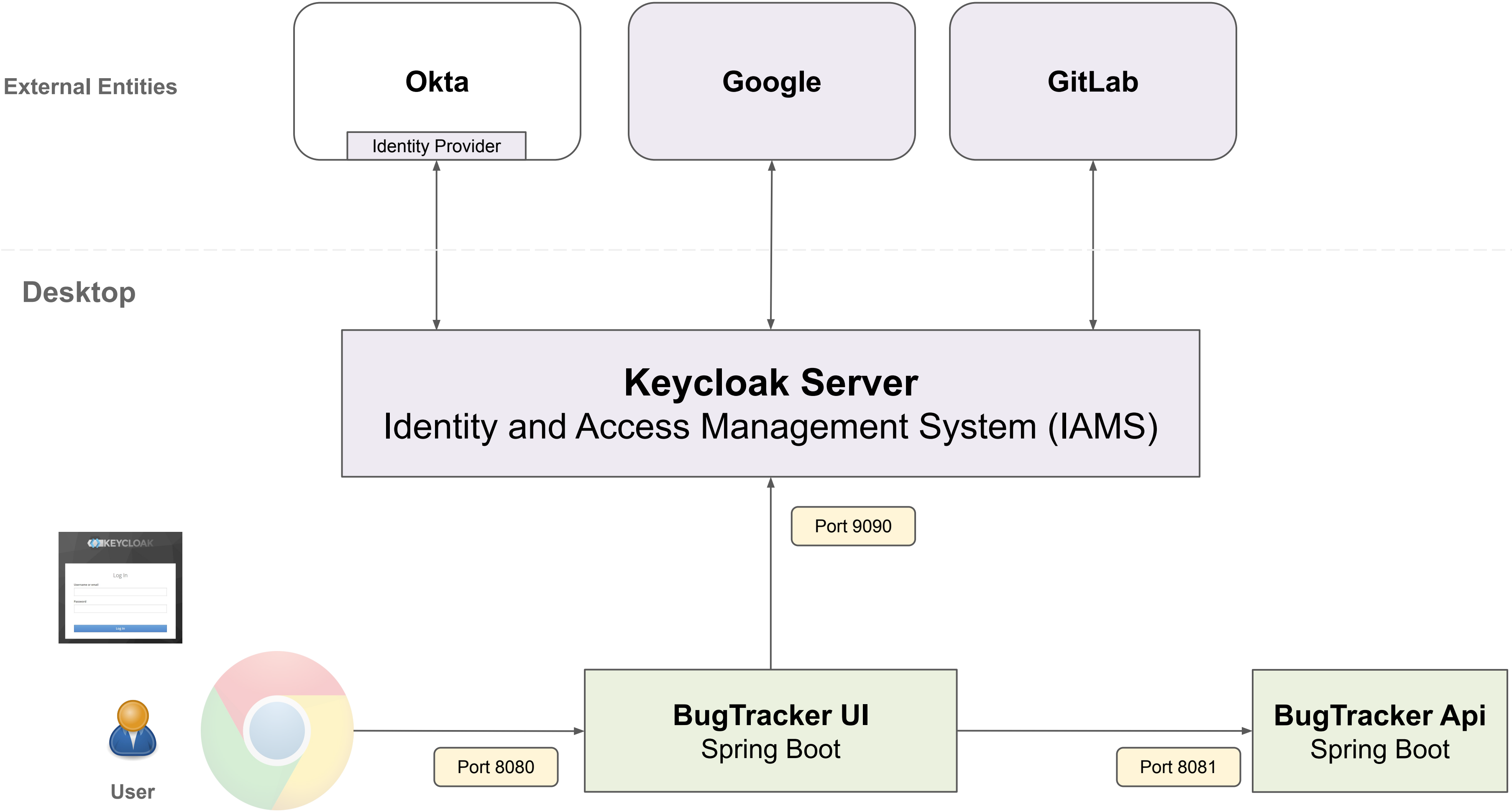
Port 9090

Port 8080

BugTracker UI
Spring Boot

Port 8081

BugTracker Api
Spring Boot



External Entities

Okta

Identity Provider

Google

GitLab

Desktop

Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

User

Port 8080

BugTracker UI

Spring Boot

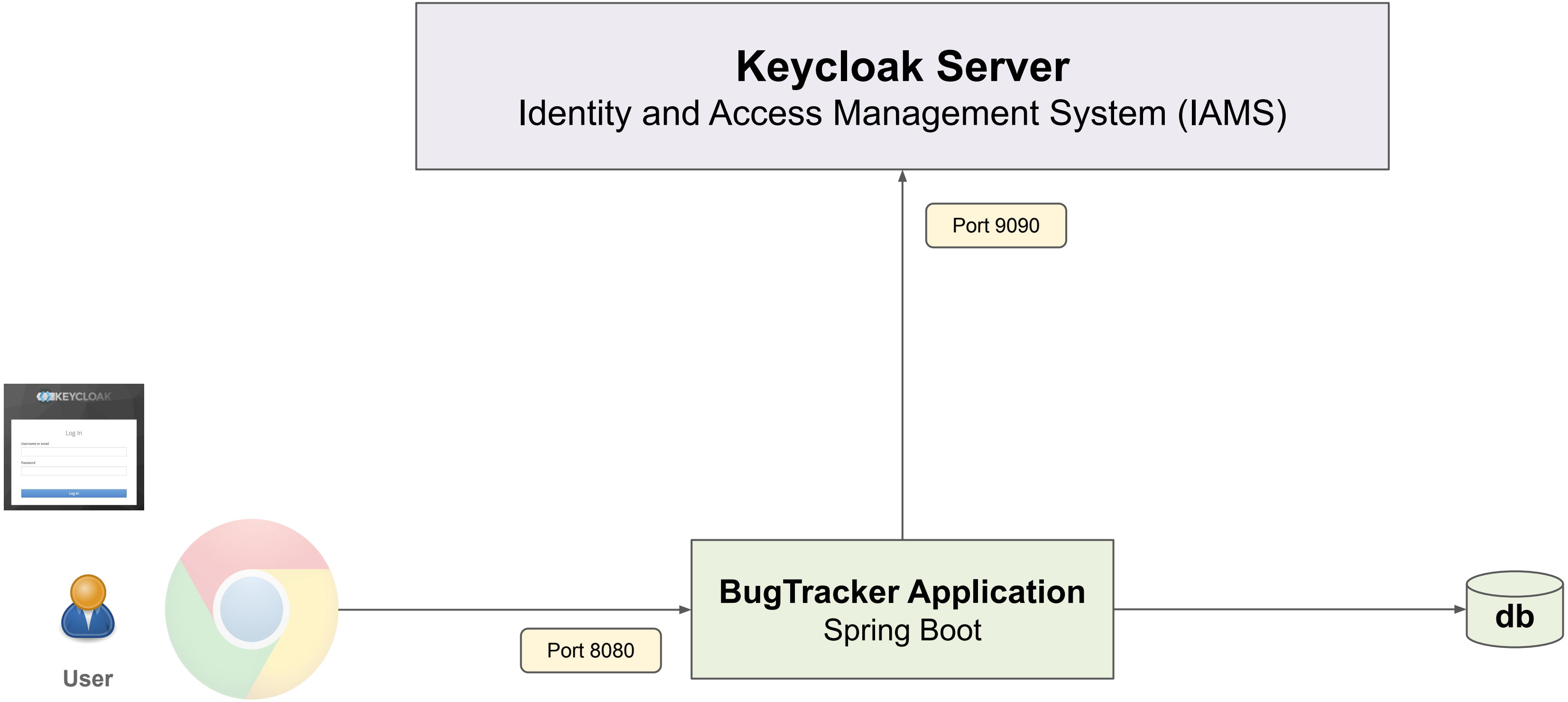
Port 8081

BugTracker Api

Spring Boot

BugTracker Demonstration

Desktop



Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

User

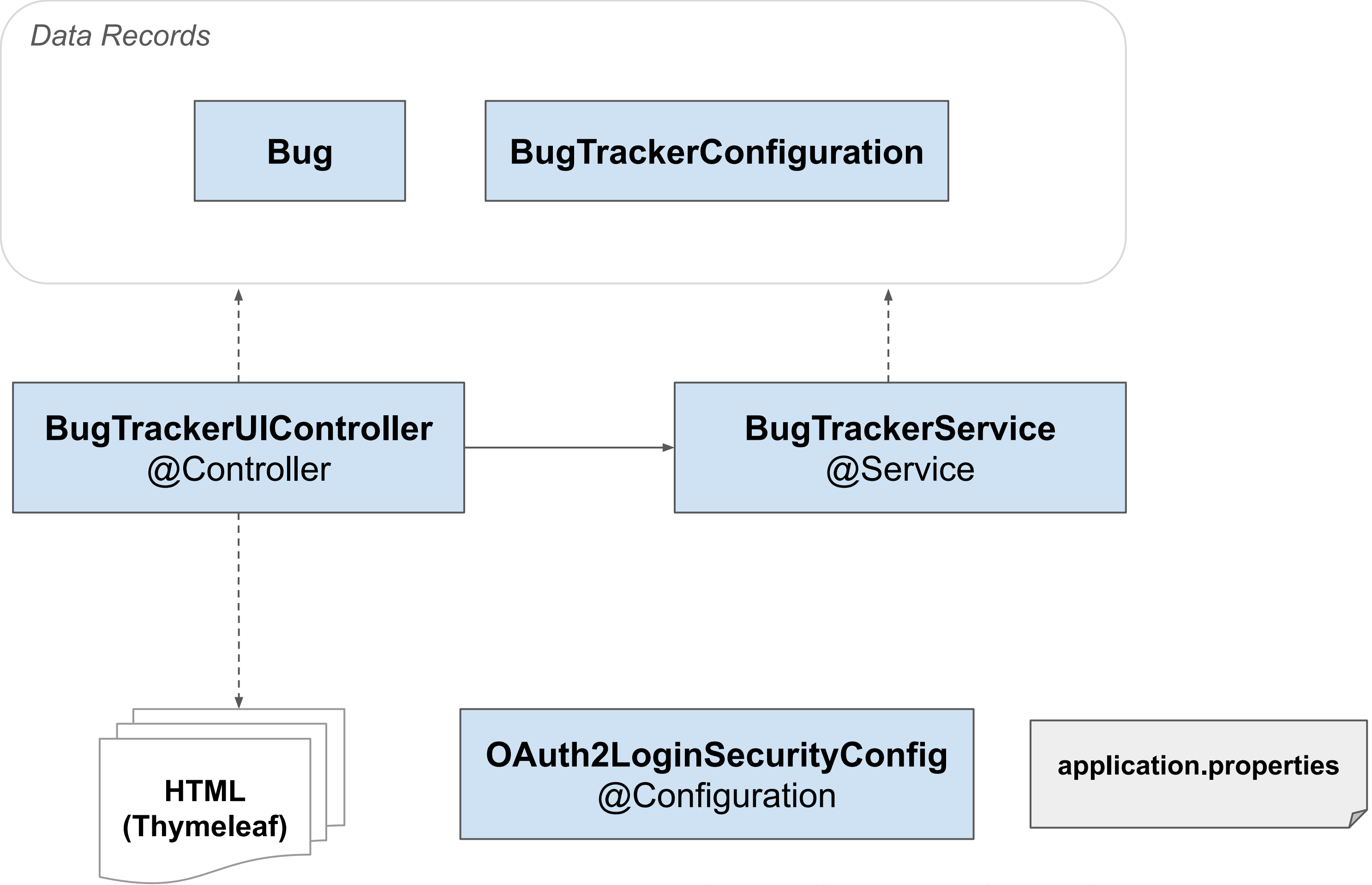
Port 8080

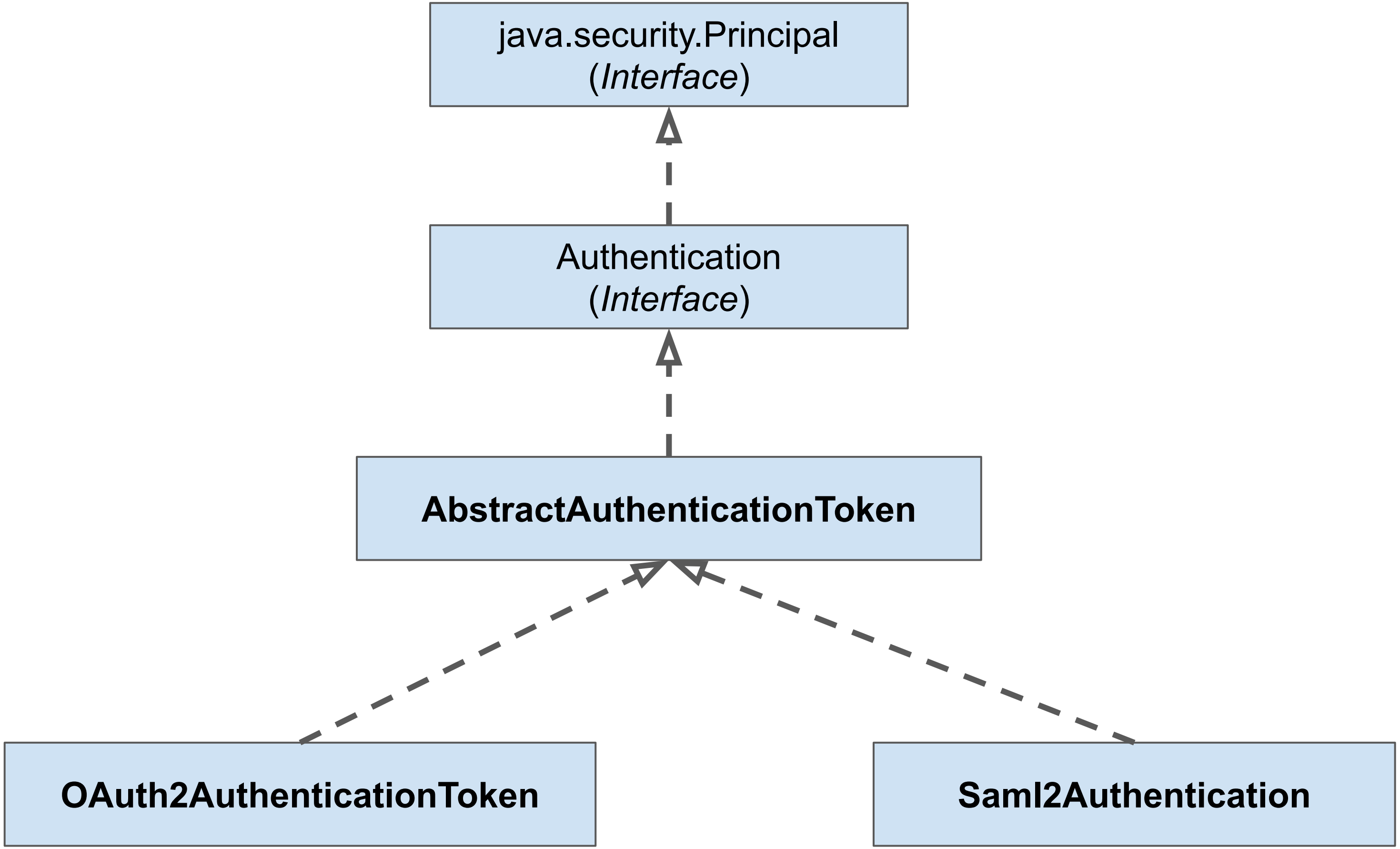
BugTracker Application

Spring Boot

db

BugTracker Application




java.security.Principal
(*Interface*)
Authentication
(*Interface*)
AbstractAuthenticationToken
OAuth2AuthenticationToken
Saml2Authentication

BugTracker Code Walkthrough

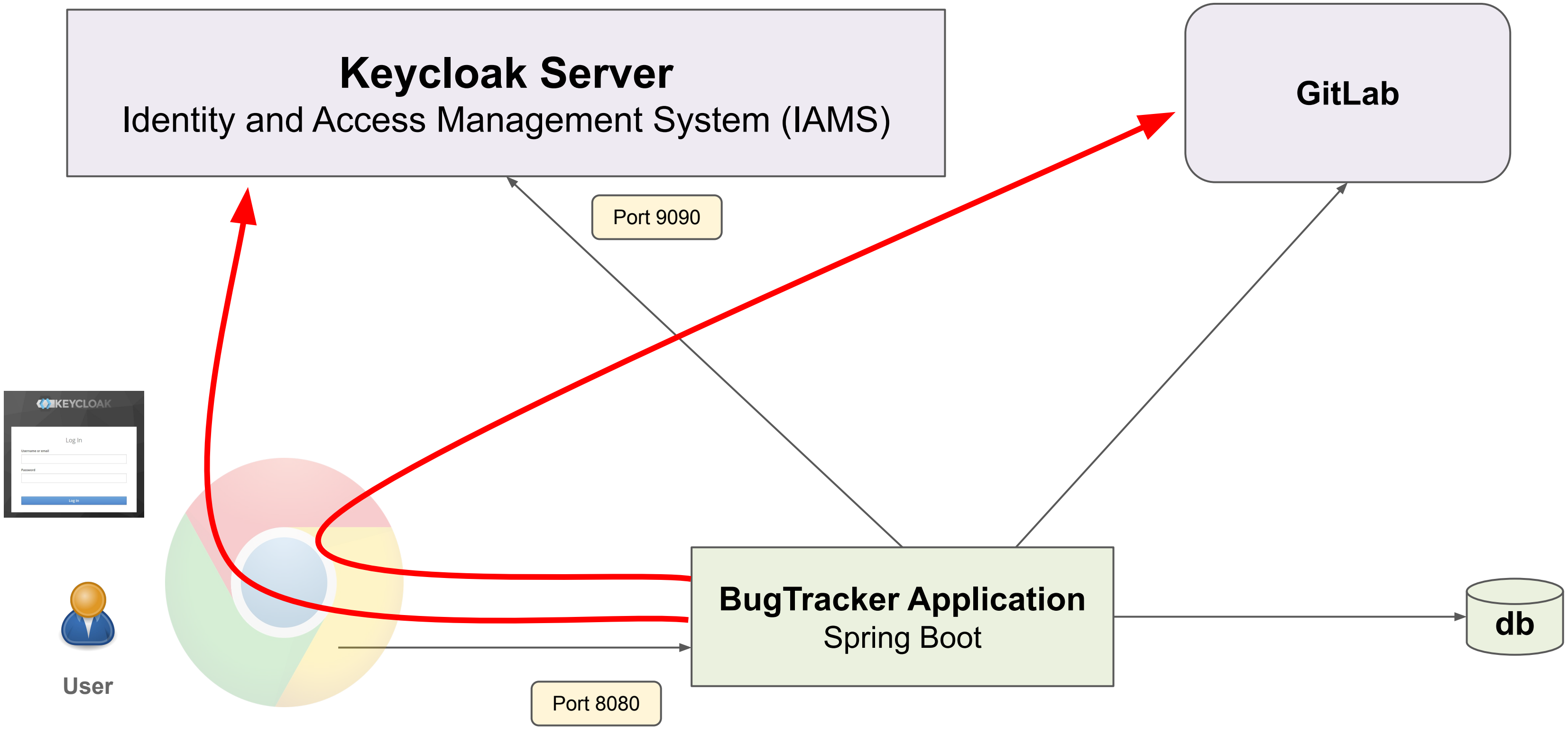
Course Project

Course Project

- JAUBS - Just Another Used Bookstore
- User Types
  - Regular User, Admin User
- JAUBS Users - Keycloak Identity Management System
- Social Users - GitLab, Google
- Partners - Okta & SAML (Wonder Bookstore)
- Pure OAuth
  - Use GitLab to pull data

OpenID Connect with Roles

Handling Multiple Providers



Keycloak Server

Identity and Access Management System (IAMS)

GitLab

Port 9090

BugTracker Application

Spring Boot

db

Port 8080

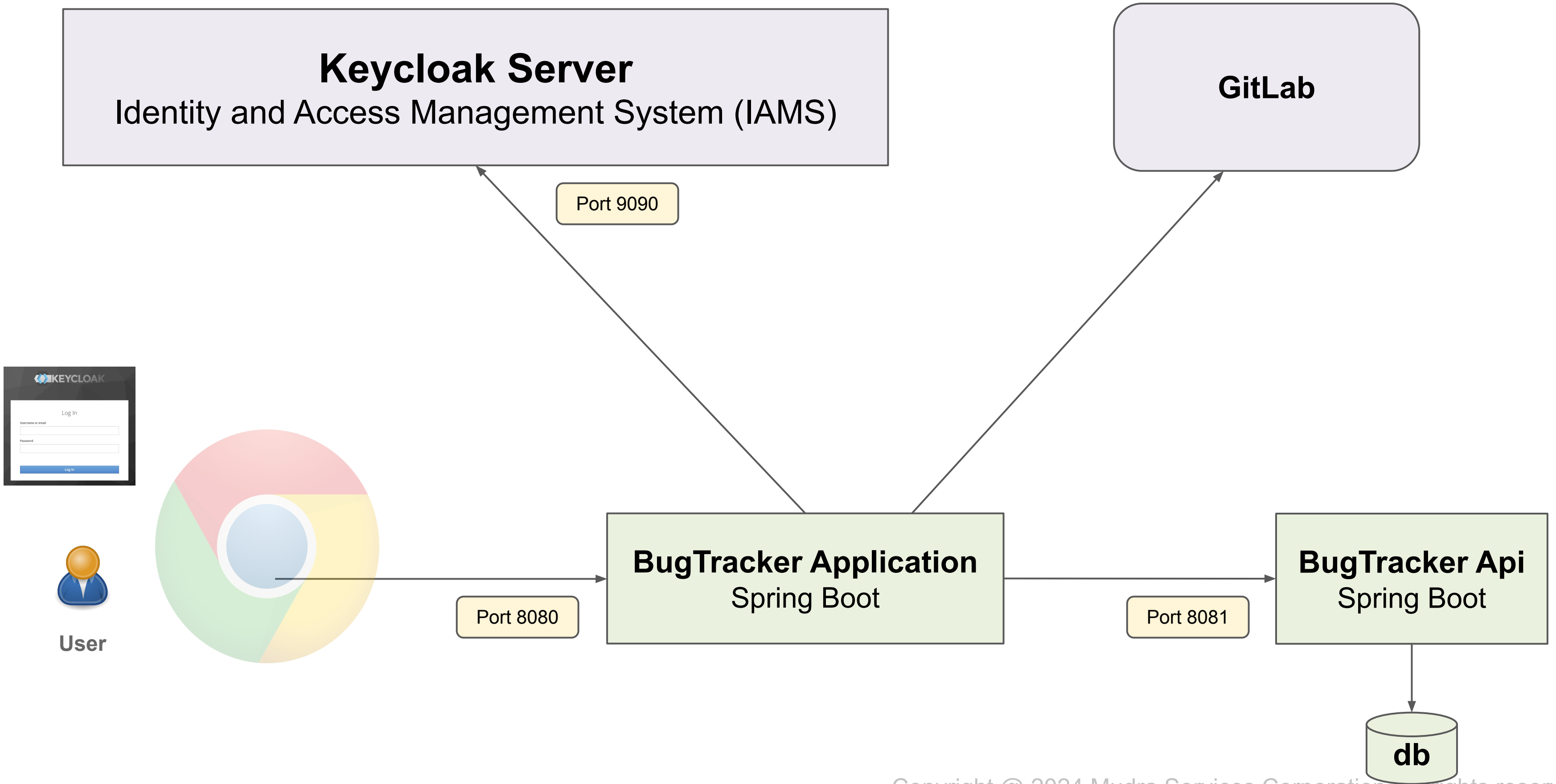
User

```
# KeyCloak specific OAuth 2 related properties
spring.security.oauth2.client.registration.keycloak-oidc.provider=keycloak
spring.security.oauth2.client.registration.keycloak-oidc.client-name=Keycloak
spring.security.oauth2.client.registration.keycloak-oidc.client-id=bugtracker
spring.security.oauth2.client.registration.keycloak-oidc.client-secret=**********************
spring.security.oauth2.client.registration.keycloak-oidc.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.keycloak-oidc.scope=openid,profile,email

# You need to set the issuer correctly
# Openid configuration - http://localhost:9090/realms/oauthcourse/.well-known/openid-configuration
spring.security.oauth2.client.provider.keycloak.issuer-uri=http://127.0.0.1:9090/realms/oauthcourse
```

```
# KeyCloak specific OAuth 2 related properties
spring.security.oauth2.client.registration.keycloak-oidc.provider=keycloak
spring.security.oauth2.client.registration.keycloak-oidc.client-name=Keycloak
spring.security.oauth2.client.registration.keycloak-oidc.client-id=bugtracker
spring.security.oauth2.client.registration.keycloak-oidc.client-secret=************************
spring.security.oauth2.client.registration.keycloak-oidc.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.keycloak-oidc.scope=openid,profile,email

# You need to set the issuer correctly
# Openid configuration - http://localhost:9090/realms/oauthcourse/.well-known/openid-configuration
spring.security.oauth2.client.provider.keycloak.issuer-uri=http://127.0.0.1:9090/realms/oauthcourse

# GitLab specific OAuth 2 related properties
spring.security.oauth2.client.registration.gitlab-oidc.provider=gitlab
spring.security.oauth2.client.registration.gitlab-oidc.client-name=GitLab
spring.security.oauth2.client.registration.gitlab-oidc.client-id=*************
spring.security.oauth2.client.registration.gitlab-oidc.client-secret=**********
spring.security.oauth2.client.registration.gitlab-oidc.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.gitlab-oidc.scope=openid,profile,email,read_user,read_api

# You need to set the issuer correctly (Not Strictly Required because its already known to Spring Boot)
# Openid configuration - https://gitlab.com/.well-known/openid-configuration
spring.security.oauth2.client.provider.gitlab.issuer-uri=https://gitlab.com
```

Architecture Downsides

- ❖ Need to handle multiple Identity Providers
- ❖ Need to translate claims in code
- ❖ Users are not consolidated into one place
- ❖ Architecture Problem
  - ➢ Single Page Applications
  - ➢ Microservices problem



Keycloak Server
Identity and Access Management System (IAMS)

GitLab

Port 9090

User

Port 8080

BugTracker Application
Spring Boot

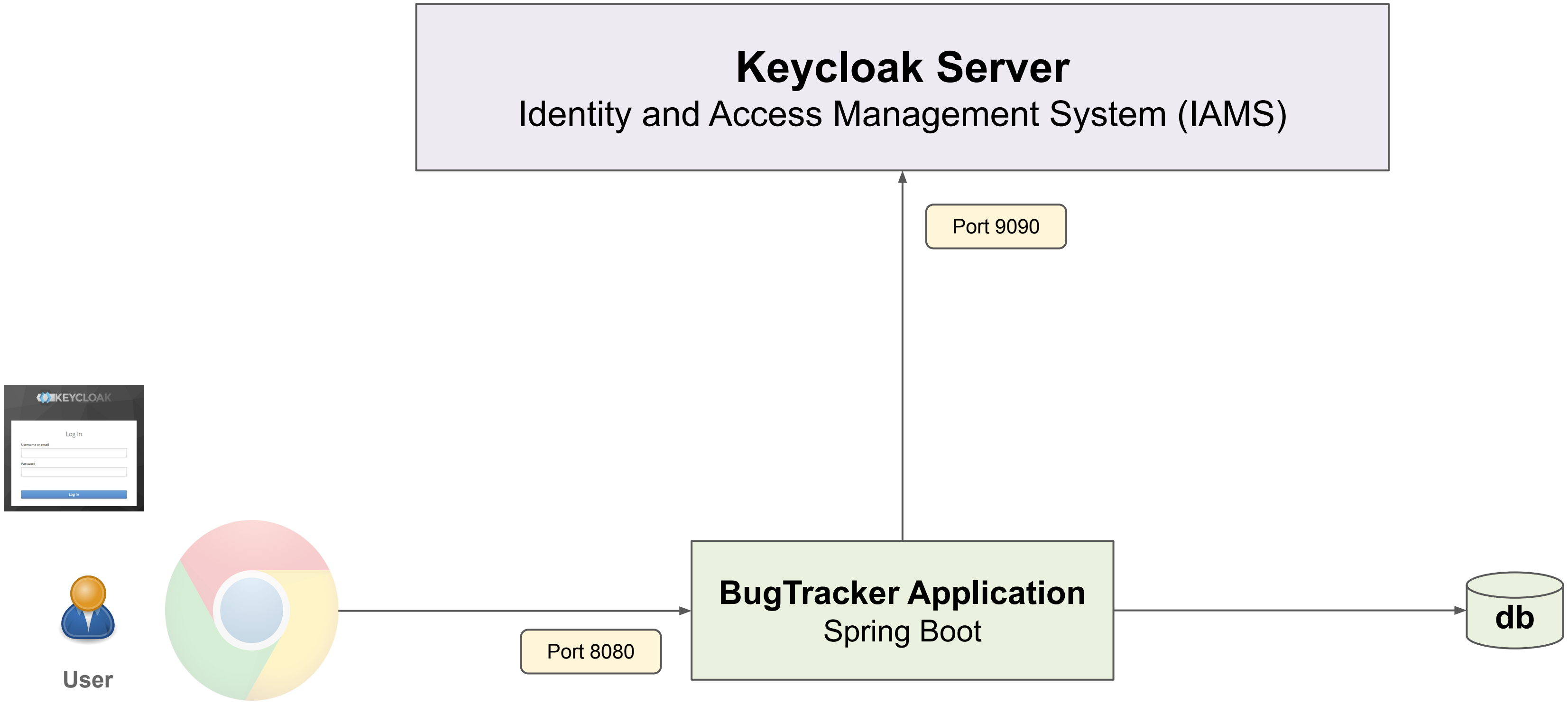
Port 8081

BugTracker Api
Spring Boot

db

OpenID Connect and Microservices

Desktop



Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

BugTracker Application

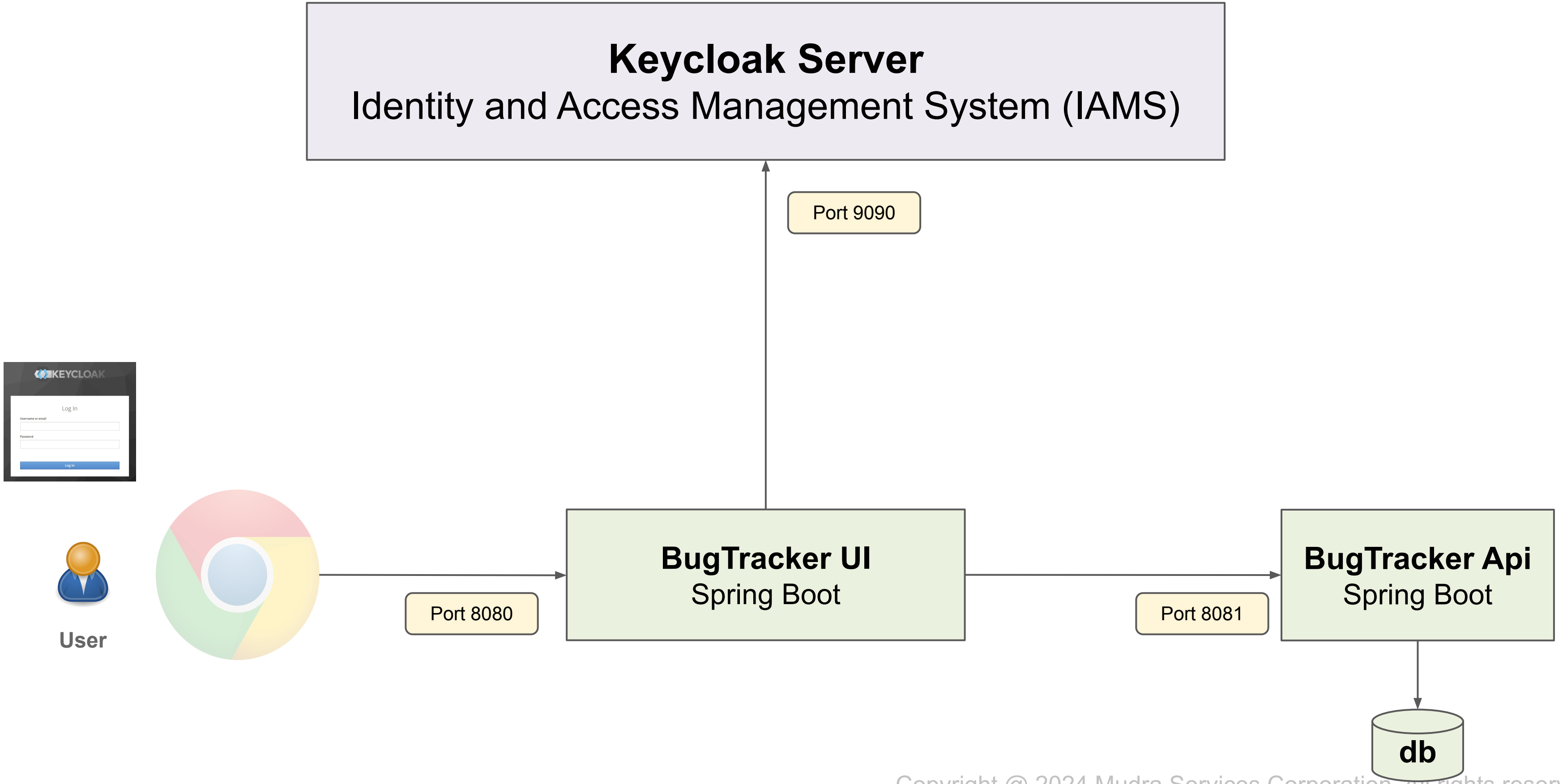
Spring Boot

Port 8080

db

User

Desktop



Keycloak Server
Identity and Access Management System (IAMS)

Port 9090

BugTracker UI
Spring Boot

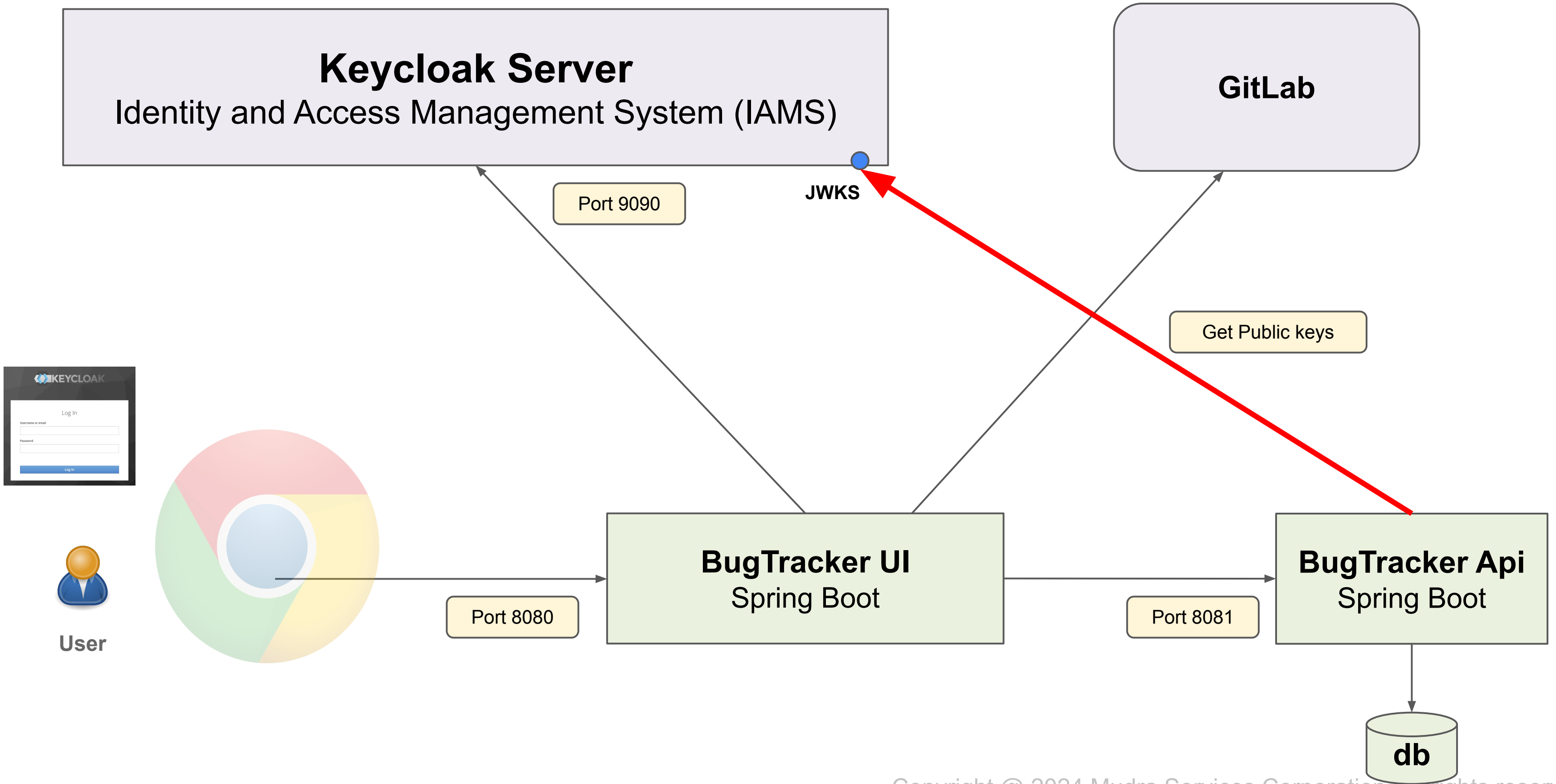
Port 8080

BugTracker Api
Spring Boot

Port 8081

db

User



Keycloak Server

Identity and Access Management System (IAMS)

JWKS

Port 9090

GitLab

Get Public keys

User

Port 8080

BugTracker UI

Spring Boot

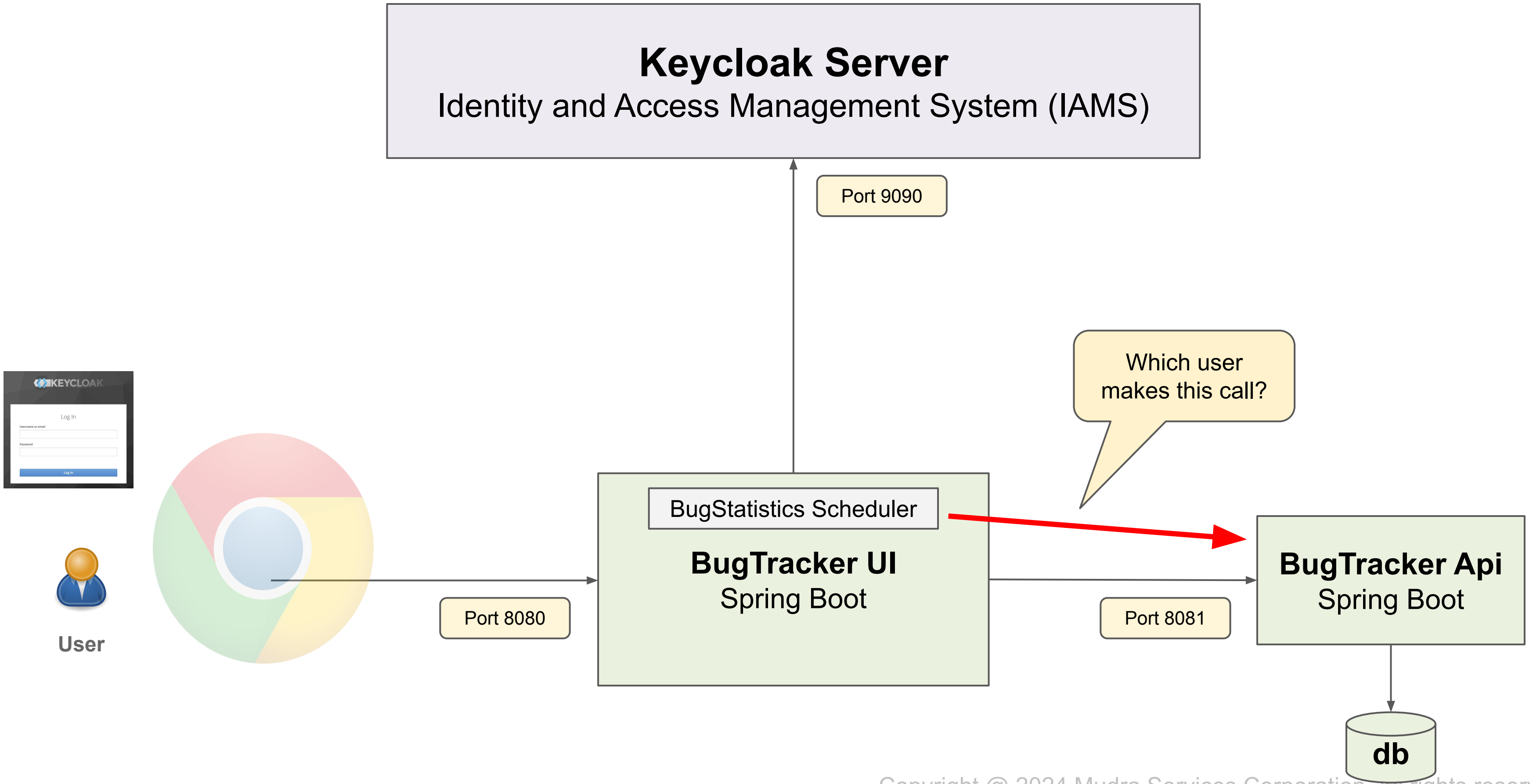
Port 8081

BugTracker Api

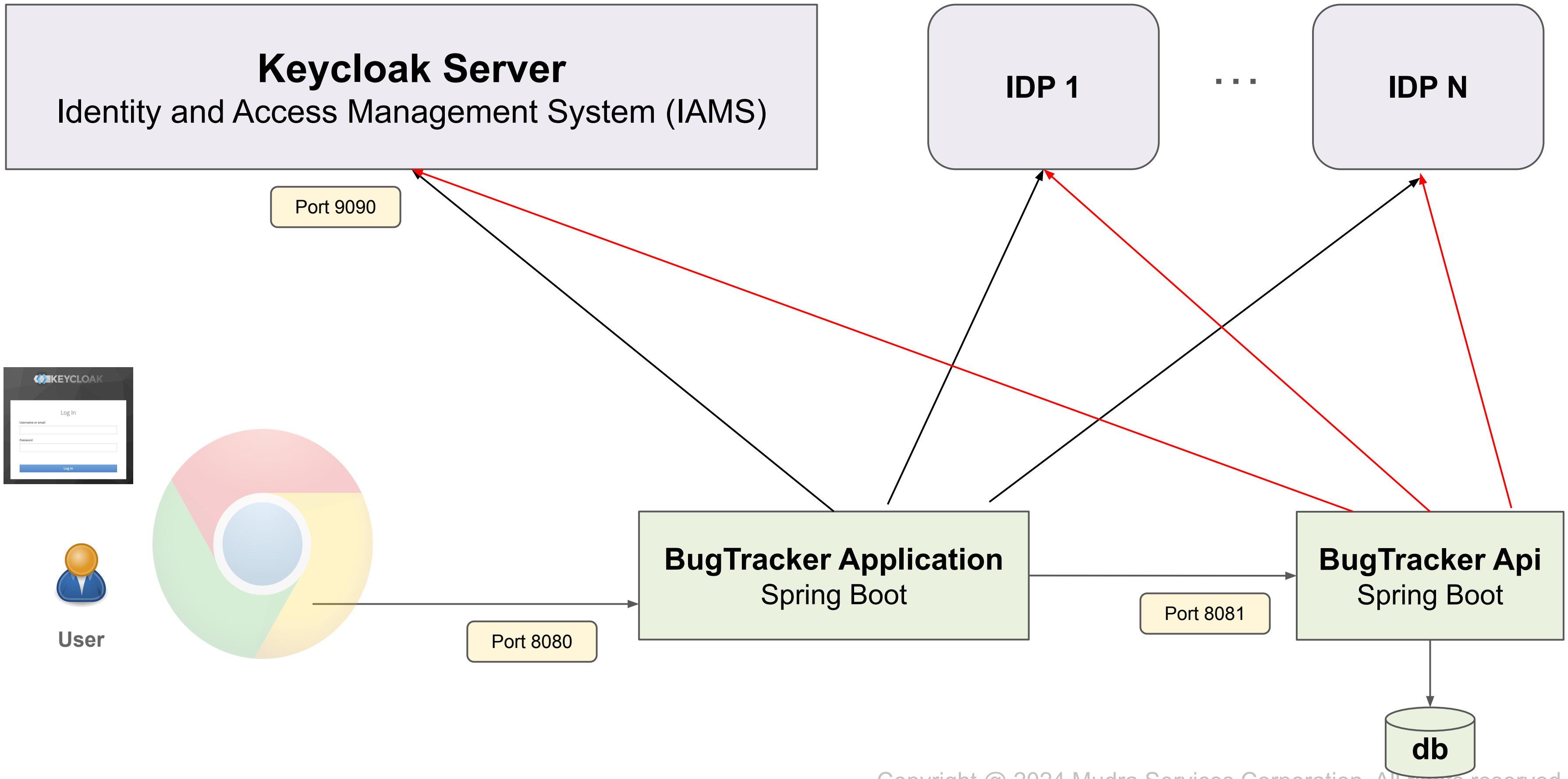
Spring Boot

db

Schedulers in an Enterprise


Keycloak Server
Identity and Access Management System (IAMS)
Port 9090
KEYCLOAK
Log In
Username or email
Password
Log In
User
Port 8080
BugStatistics Scheduler
BugTracker UI
Spring Boot
Which user makes this call?
BugTracker Api
Spring Boot
Port 8081
db

Keycloak and Identity Brokers



Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

IDP 1

. . .

IDP N

User

BugTracker Application

Spring Boot

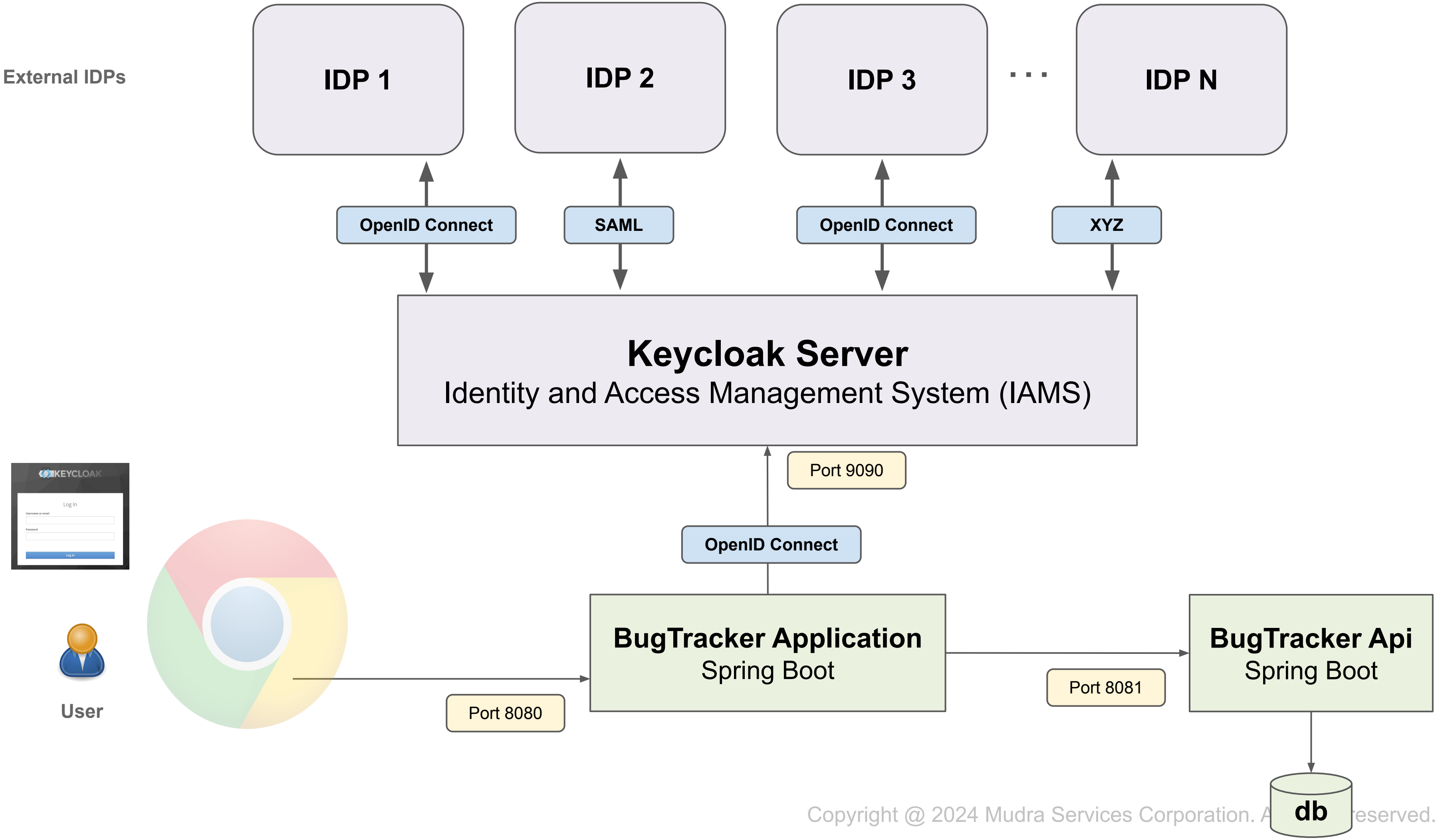
Port 8080

BugTracker Api

Spring Boot

Port 8081

db



External IDPs

IDP 1

IDP 2

IDP 3

. . .

IDP N

OpenID Connect

SAML

OpenID Connect

XYZ

Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

OpenID Connect

User

BugTracker Application

Spring Boot

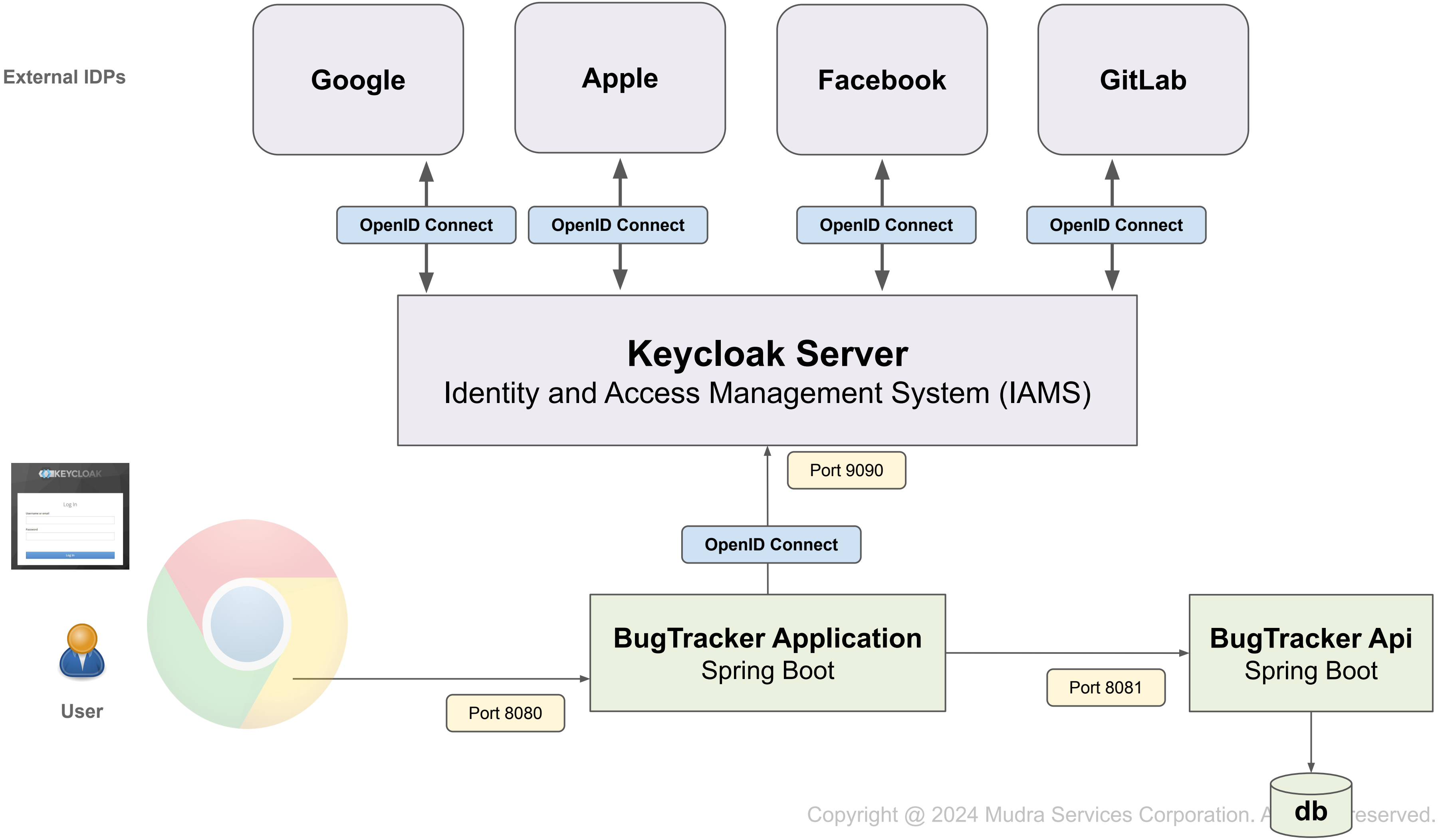
Port 8080

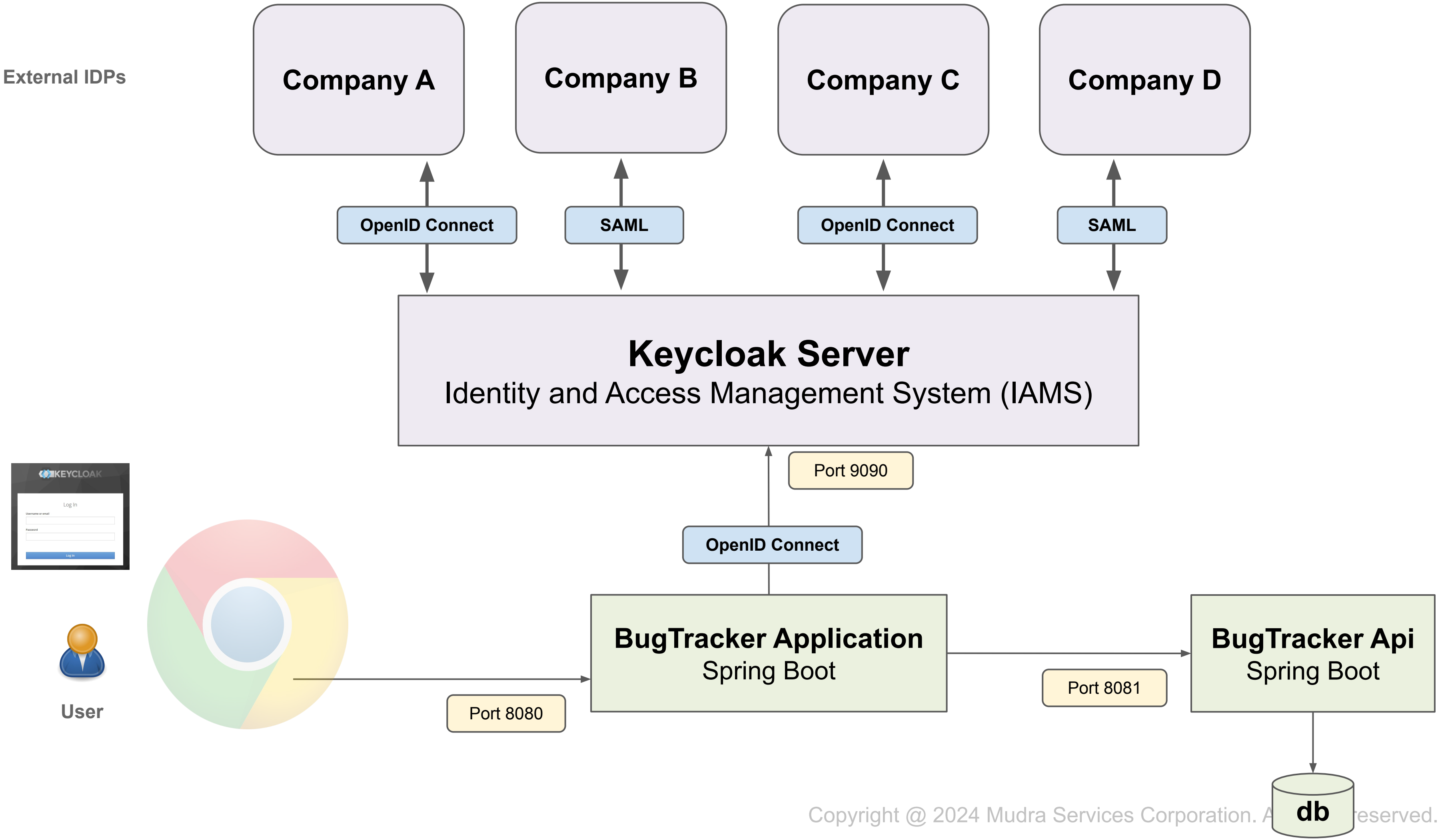
BugTracker Api

Spring Boot

Port 8081

db


External IDPs
Google
Apple
Facebook
GitLab
OpenID Connect
OpenID Connect
OpenID Connect
OpenID Connect
Keycloak Server
Identity and Access Management System (IAMS)
KEYCLOAK
Log In
Username or email
Password
Log In
Port 9090
OpenID Connect
User
Port 8080
BugTracker Application
Spring Boot
Port 8081
BugTracker Api
Spring Boot
db



External IDPs

Company A

Company B

Company C

Company D

OpenID Connect

SAML

OpenID Connect

SAML

Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

OpenID Connect

User

Port 8080

BugTracker Application

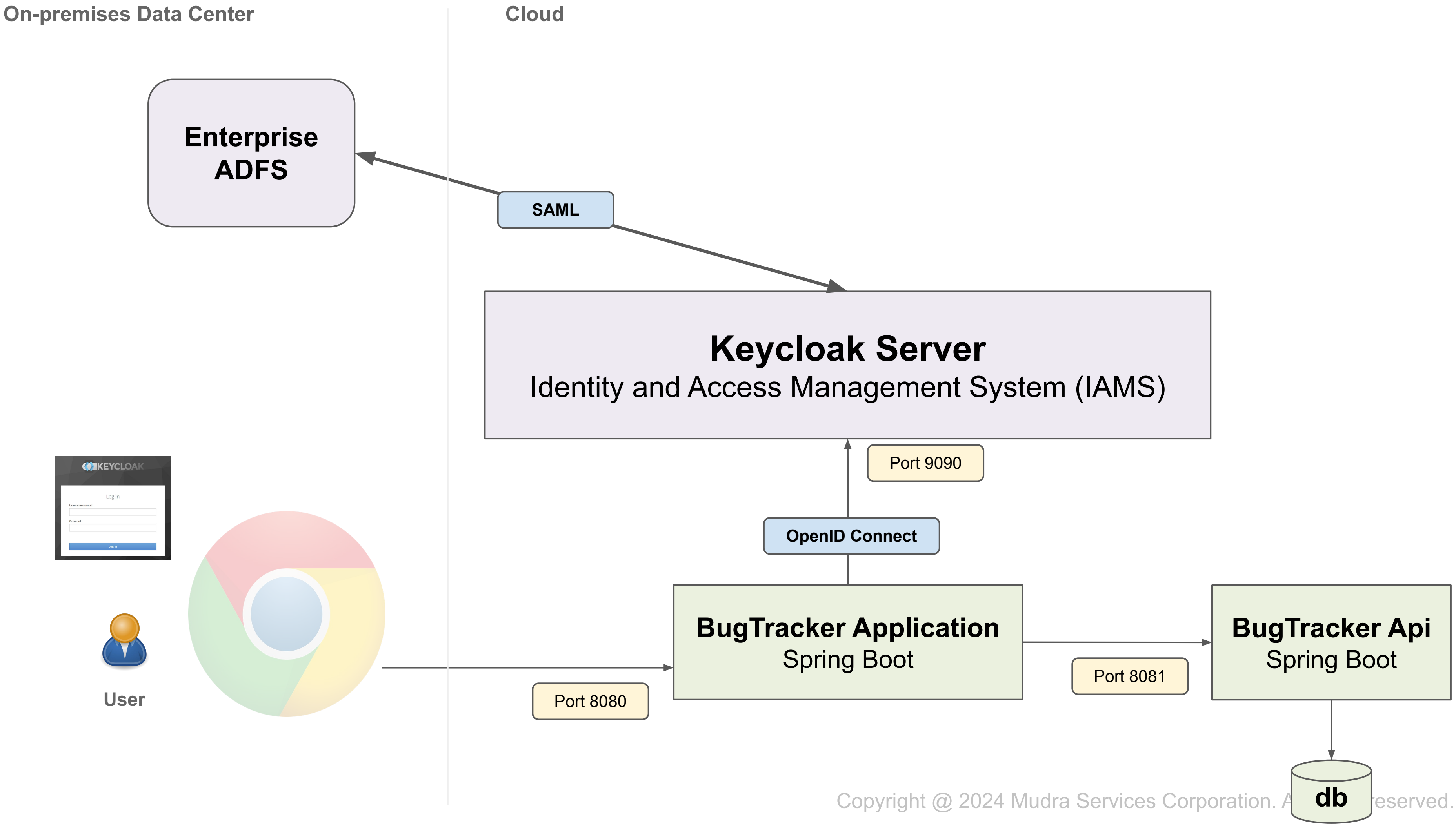
Spring Boot

Port 8081

BugTracker Api

Spring Boot

db



On-premises Data Center

Cloud

Enterprise ADFS

SAML

Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

OpenID Connect

User

Port 8080

BugTracker Application

Spring Boot

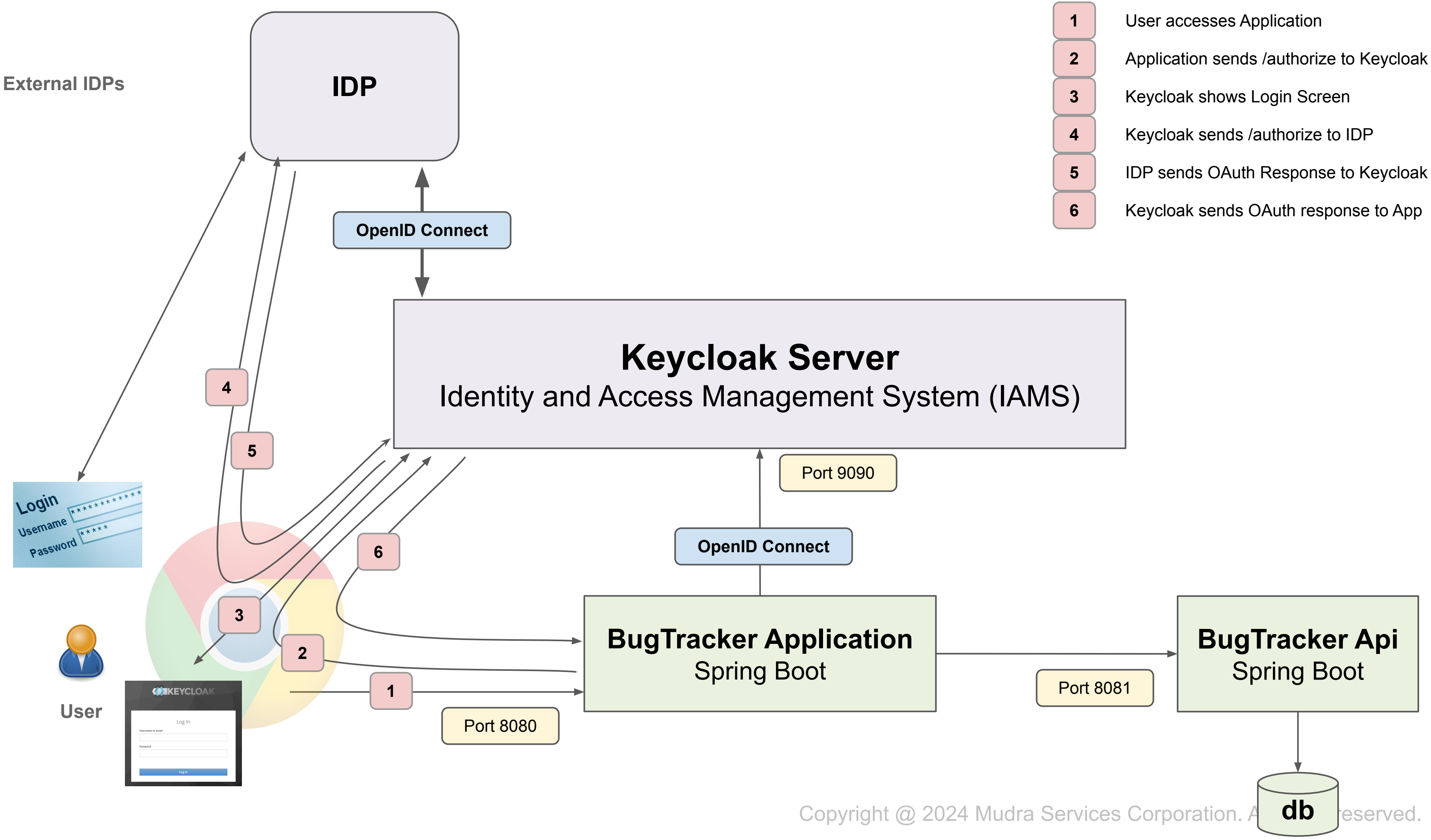
Port 8081

BugTracker Api

Spring Boot

db

Identity Broker Integration



1 User accesses Application

2 Application sends /authorize to Keycloak

3 Keycloak shows Login Screen

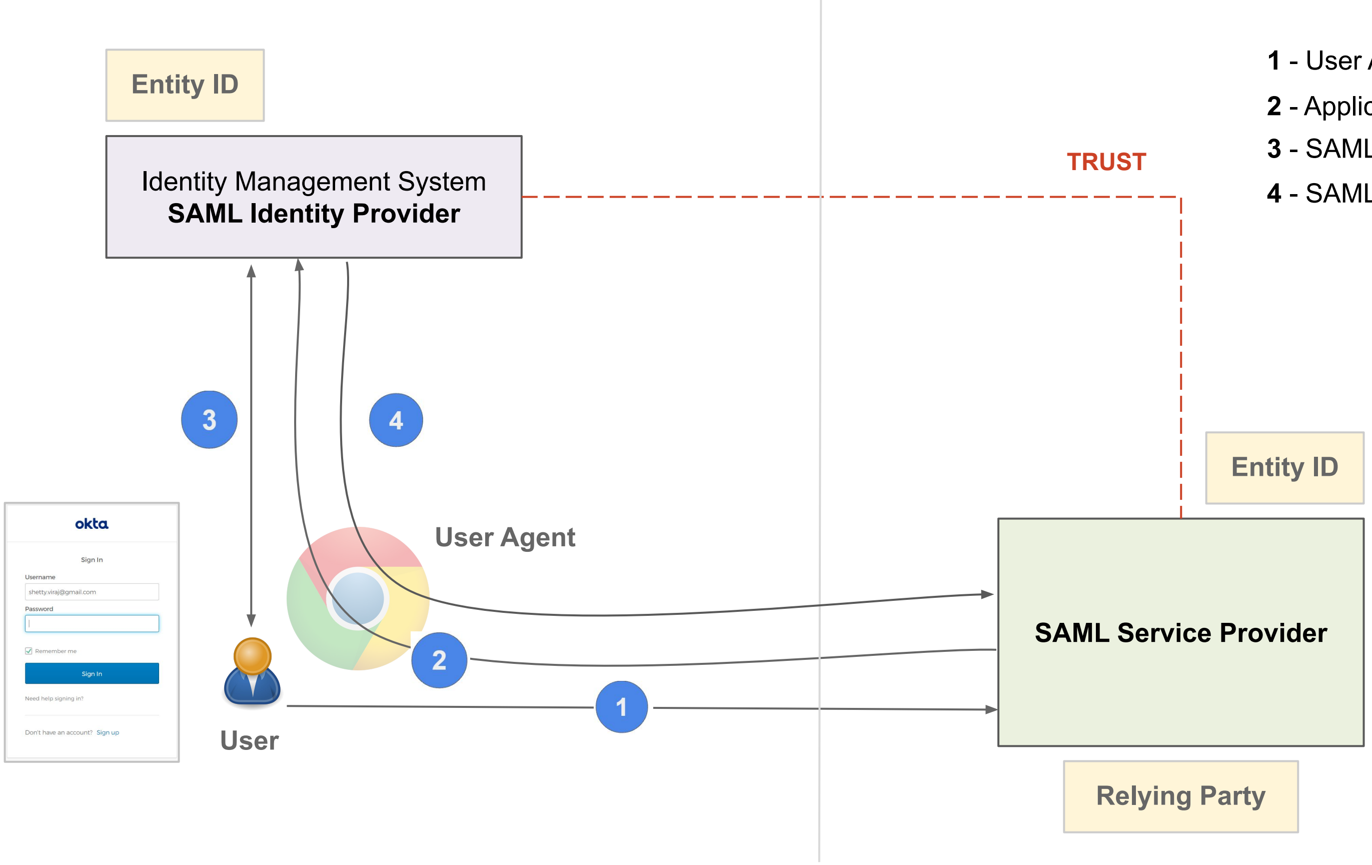
4 Keycloak sends /authorize to IDP

5 IDP sends OAuth Response to Keycloak

6 Keycloak sends OAuth response to App

SAML
Security Assertion Markup Language

SP INITIATED SAML SSO



1 - User Accesses SP

2 - Application forwards Request to IDP

3 - SAML IDP displays Login Screen

4 - SAML IDP forwards Response to SP

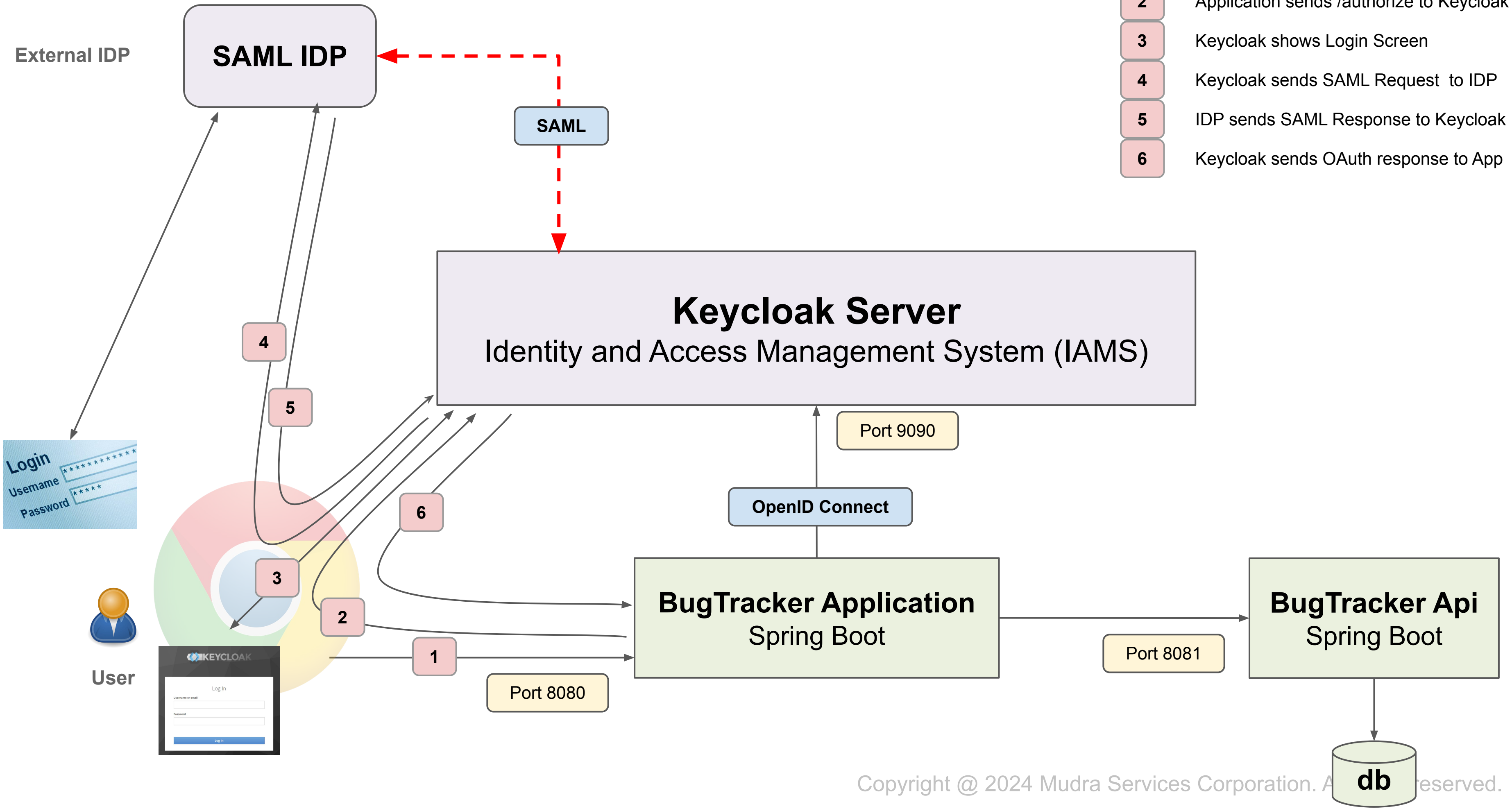
SAML Request

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    AssertionConsumerServiceURL="http://127.0.0.1:9090/realms/oauthrealm/..."
    AttributeConsumingServiceIndex="0"
    Destination="https://dev-83512084.okta.com/app/dev-83512084_bugtracker_1/..."
    ForceAuthn="false" ID="ID_b7e54936-f643-409d-a4b5-744b8587cf0f"
    IssueInstant="2023-12-19T22:02:27.041Z"
    ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" Version="2.0">
    <saml:Issuer>http://127.0.0.1:9090/realms/oauthrealm</saml:Issuer>
    <samlp:NameIDPolicy AllowCreate="true"
        Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" />
</samlp:AuthnRequest>
```

```xml
<saml2p:Response Destination="http://127.0.0.1:9090/realms/oauthrealm/broker/besttraining/endpoint" ...>
    <saml2:Issuer ...>http://www.okta.com/exke0e67d5YHL4zbZ5d7</saml2:Issuer>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
    <saml2p:Status xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">...</saml2p:Status>
    <saml2:Assertion ID="id76018112014025024763 6793" ...>
        <saml2:Issuer ...>http://www.okta.com/exke0e67d5YHL4zbZ5d7</saml2:Issuer>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
        <saml2:Subject xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
            <saml2:NameID ...>jimdoe@besttraining.com</saml2:NameID>
            <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">...</saml2:SubjectConfirmation>
        </saml2:Subject>
        <saml2:Conditions>
            <saml2:AudienceRestriction>
                <saml2:Audience>http://127.0.0.1:9090/realms/oauthrealm</saml2:Audience>
            </saml2:AudienceRestriction>
        </saml2:Conditions>
        <saml2:AuthnStatement>...</saml2:AuthnStatement>
        <saml2:AttributeStatement xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
            <saml2:Attribute Name="groups" ...>
                <saml2:AttributeValue ...>bugtracker-user</saml2:AttributeValue>
            </saml2:Attribute>
        </saml2:AttributeStatement>
    </saml2:Assertion>
</saml2p:Response>
```

SAML Response

OpenID Connect and SAML



1 User accesses Application

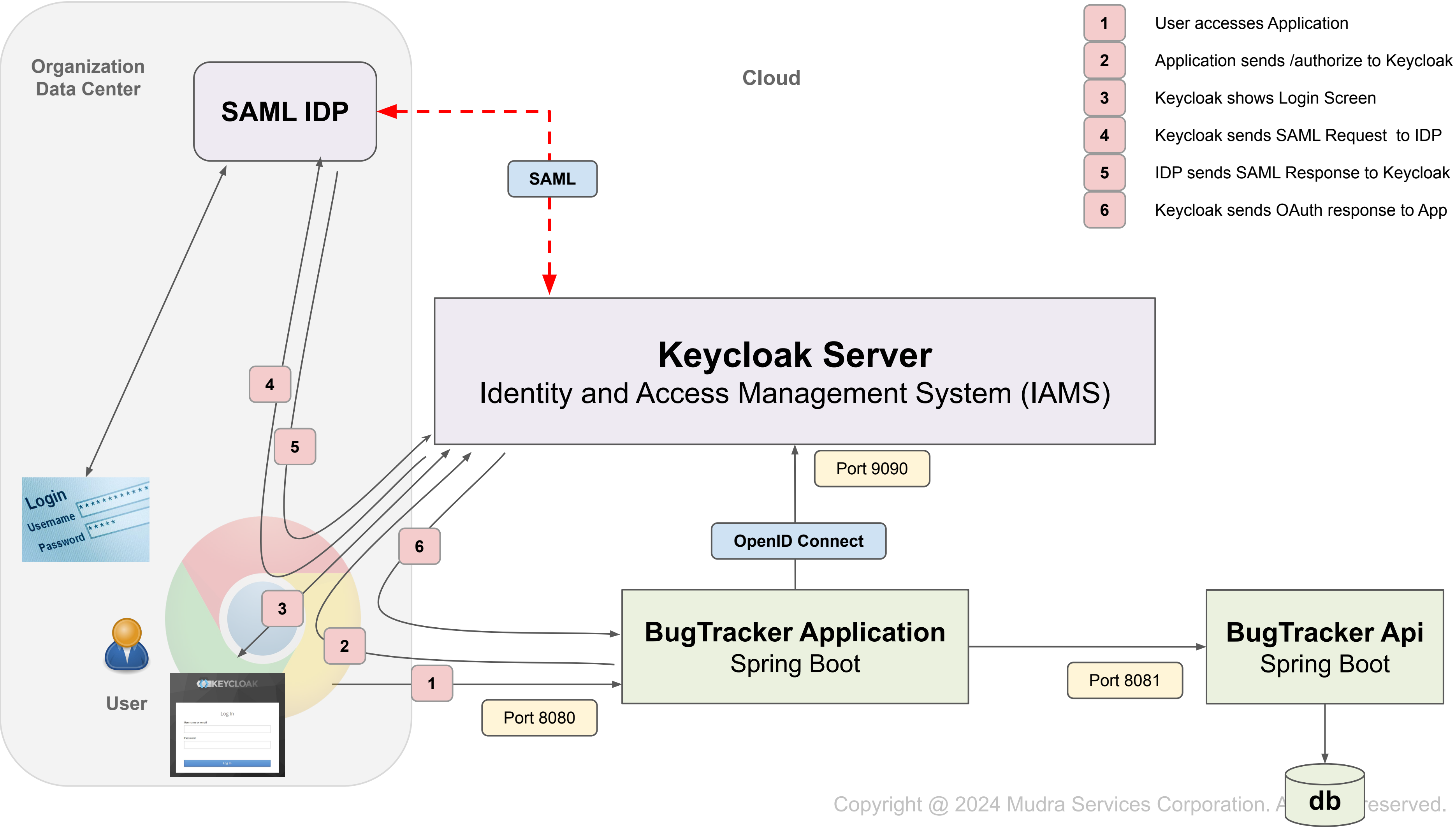
2 Application sends /authorize to Keycloak

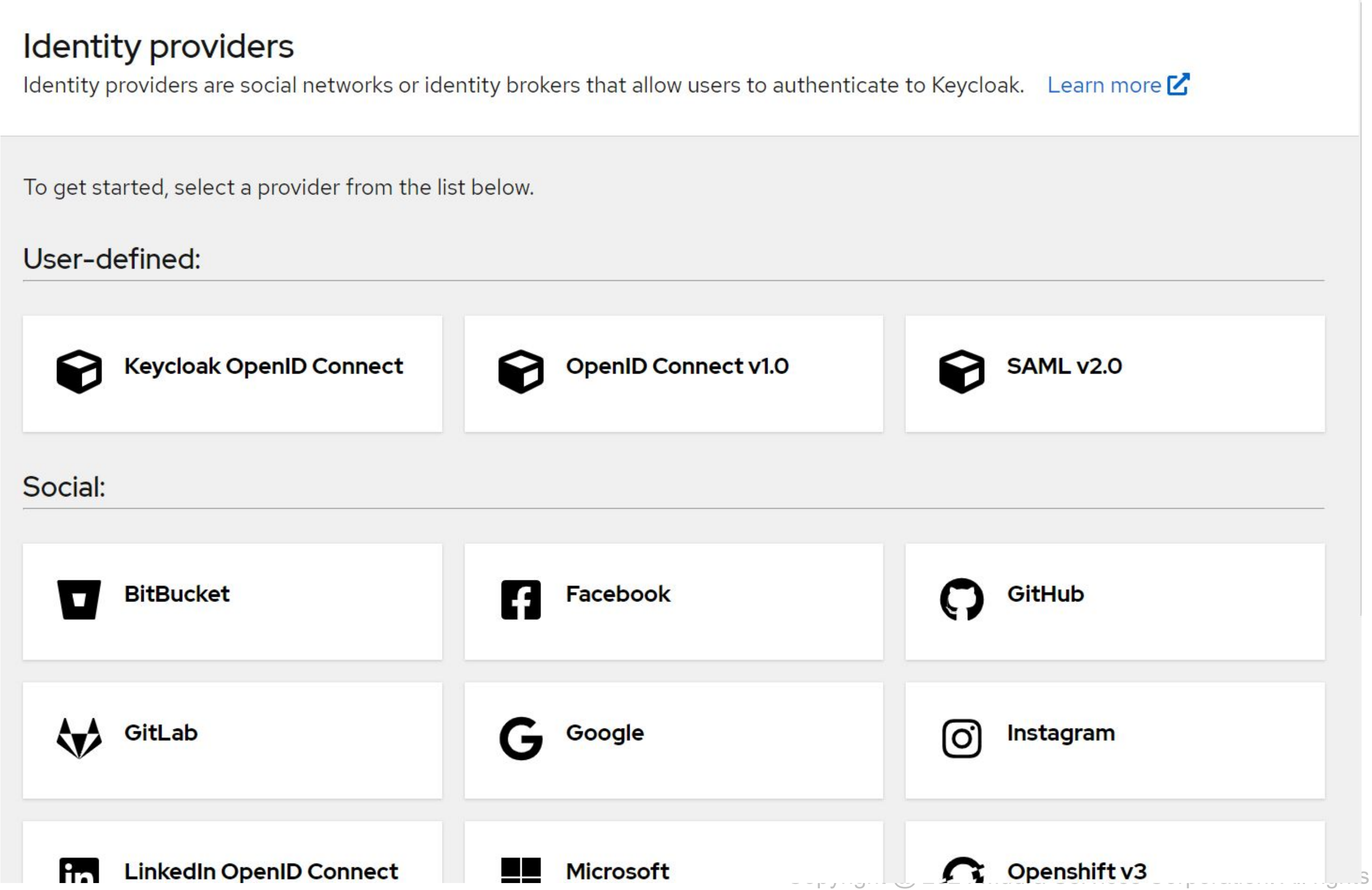
3 Keycloak shows Login Screen

4 Keycloak sends SAML Request to IDP

5 IDP sends SAML Response to Keycloak

6 Keycloak sends OAuth response to App





Identity providers

Identity providers are social networks or identity brokers that allow users to authenticate to Keycloak. Learn more

To get started, select a provider from the list below.

User-defined:

- Keycloak OpenID Connect
- OpenID Connect v1.0
- SAML v2.0

Social:

- BitBucket
- Facebook
- GitHub
- GitLab
- Google
- Instagram
- LinkedIn OpenID Connect
- Microsoft
- Openshift v3

OpenID Connect and SAML Integration Demonstration

1 User accesses Application

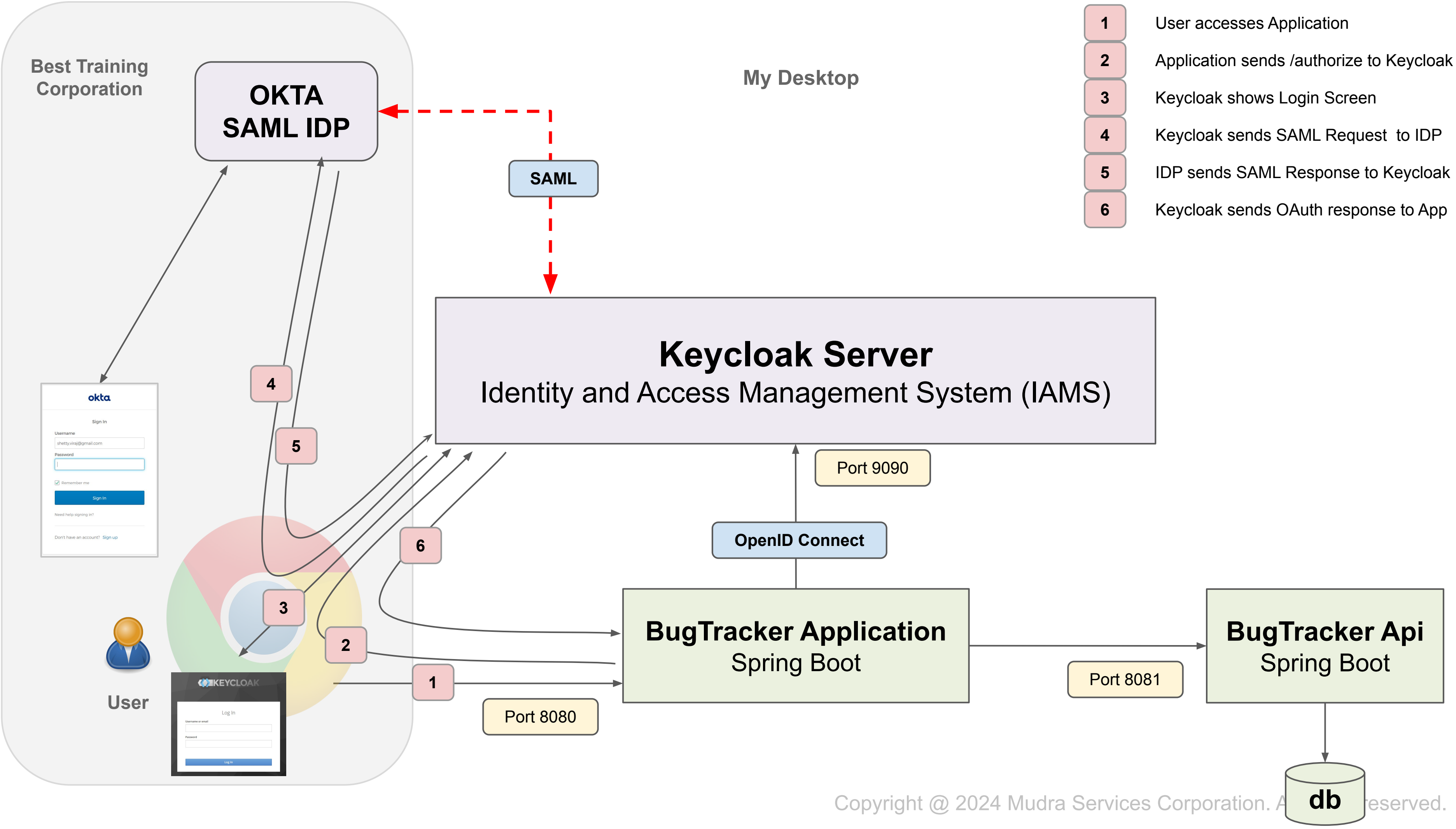
2 Application sends /authorize to Keycloak

3 Keycloak shows Login Screen

4 Keycloak sends SAML Request to IDP

5 IDP sends SAML Response to Keycloak

6 Keycloak sends OAuth response to App



Best Training Corporation

OKTA SAML IDP

SAML

My Desktop

Keycloak Server

Identity and Access Management System (IAMS)

Port 9090

OpenID Connect

BugTracker Application

Spring Boot

BugTracker Api

Spring Boot

Port 8081

Port 8080

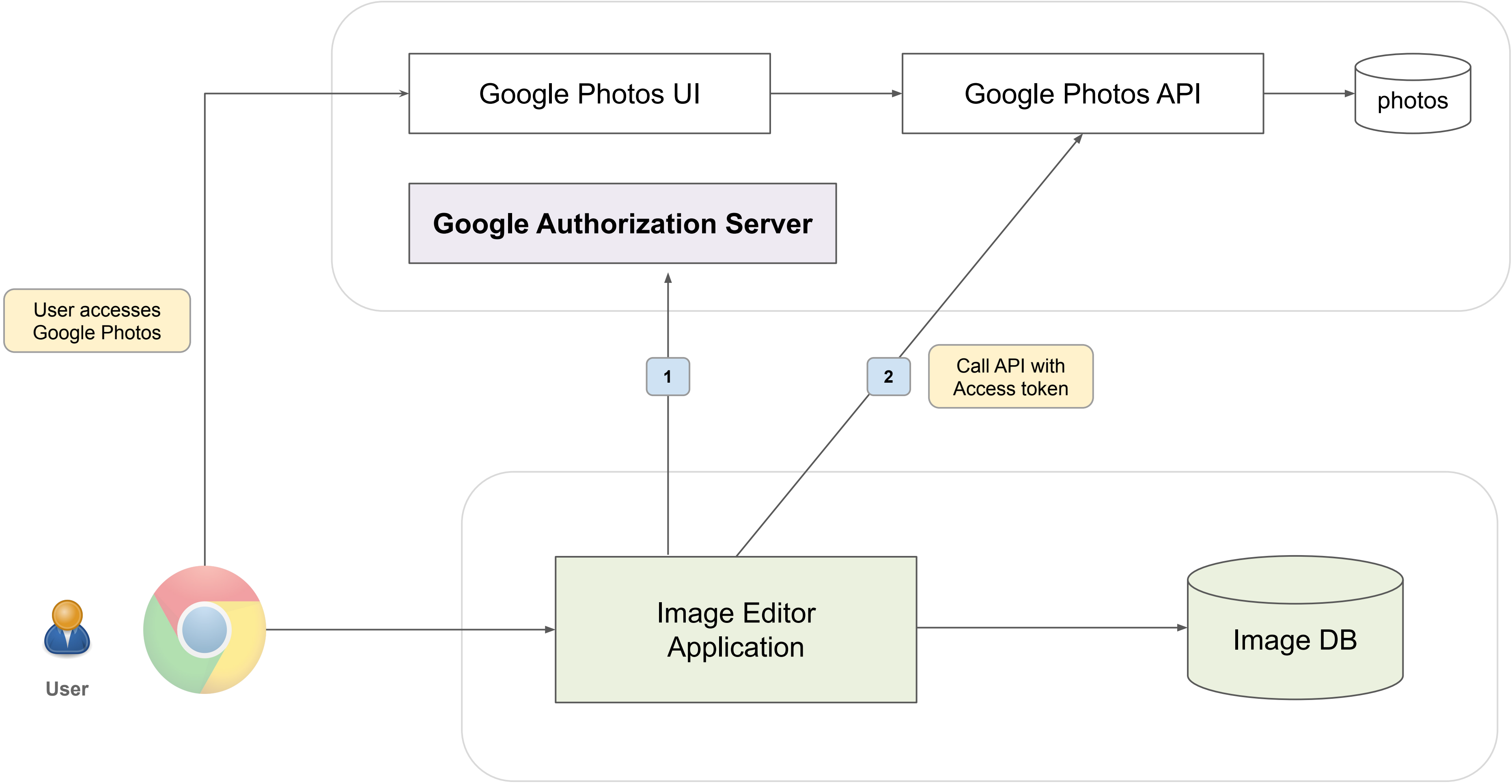
db

User

BugTracker Demonstration

- Create groups in Okta
  - bugtracker-admin, bugtracker-user
- Create a user in Okta
  - Associate with a group
- Create BugTracker SAML application in Okta
  - Associate groups to application
- Create SAML Identity Provider in Keycloak
  - Import Okta SAML Metadata
- Create Keycloak Mapping
  - Convert Okta groups to Keycloak role

Exclusive OAuth in Practice



Google Photos UI

Google Photos API

photos

Google Authorization Server

User accesses Google Photos

1

2

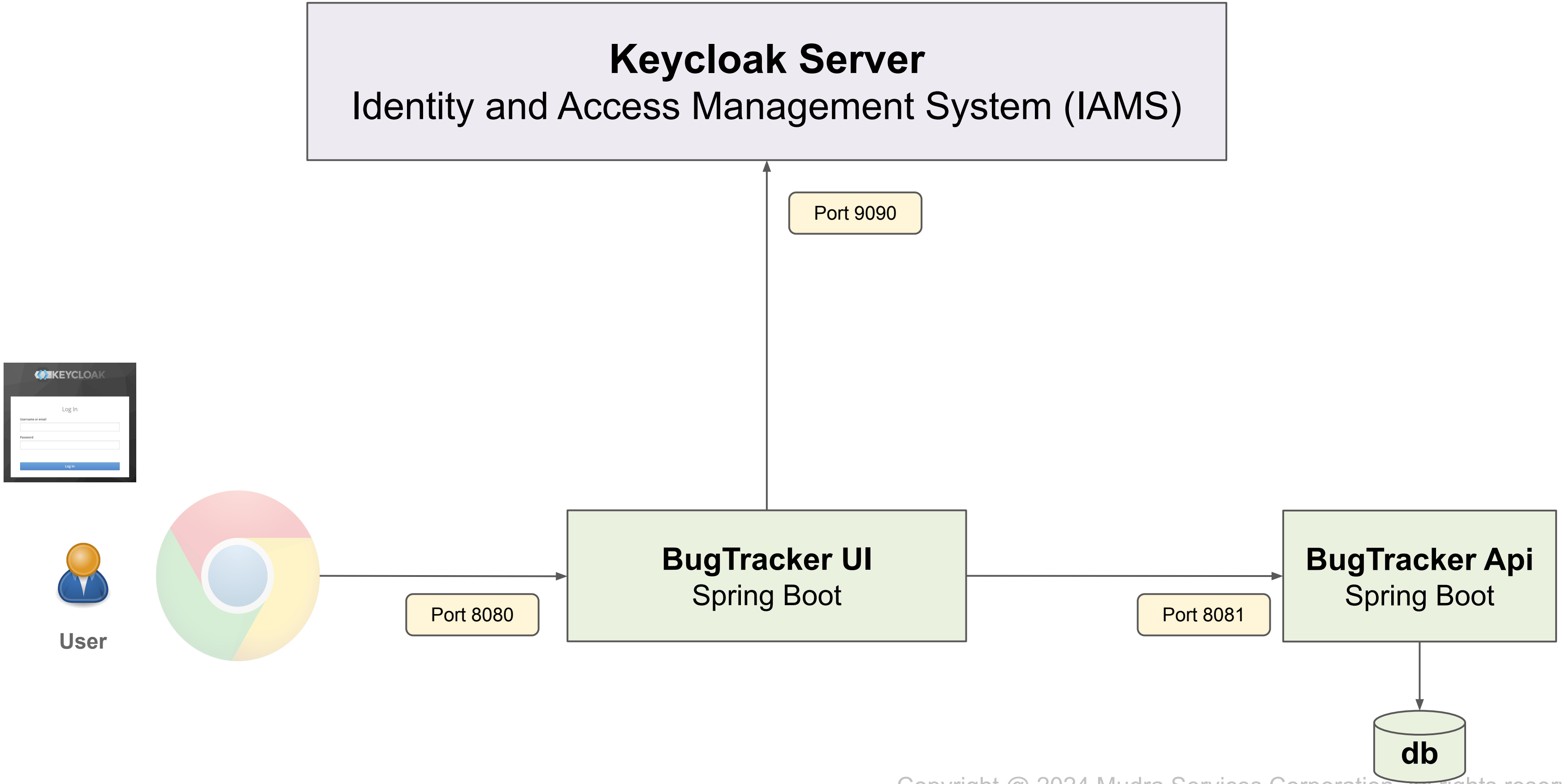
Call API with Access token

Image Editor Application

Image DB

User

Desktop



Keycloak Server
Identity and Access Management System (IAMS)

Port 9090

BugTracker UI
Spring Boot

Port 8080

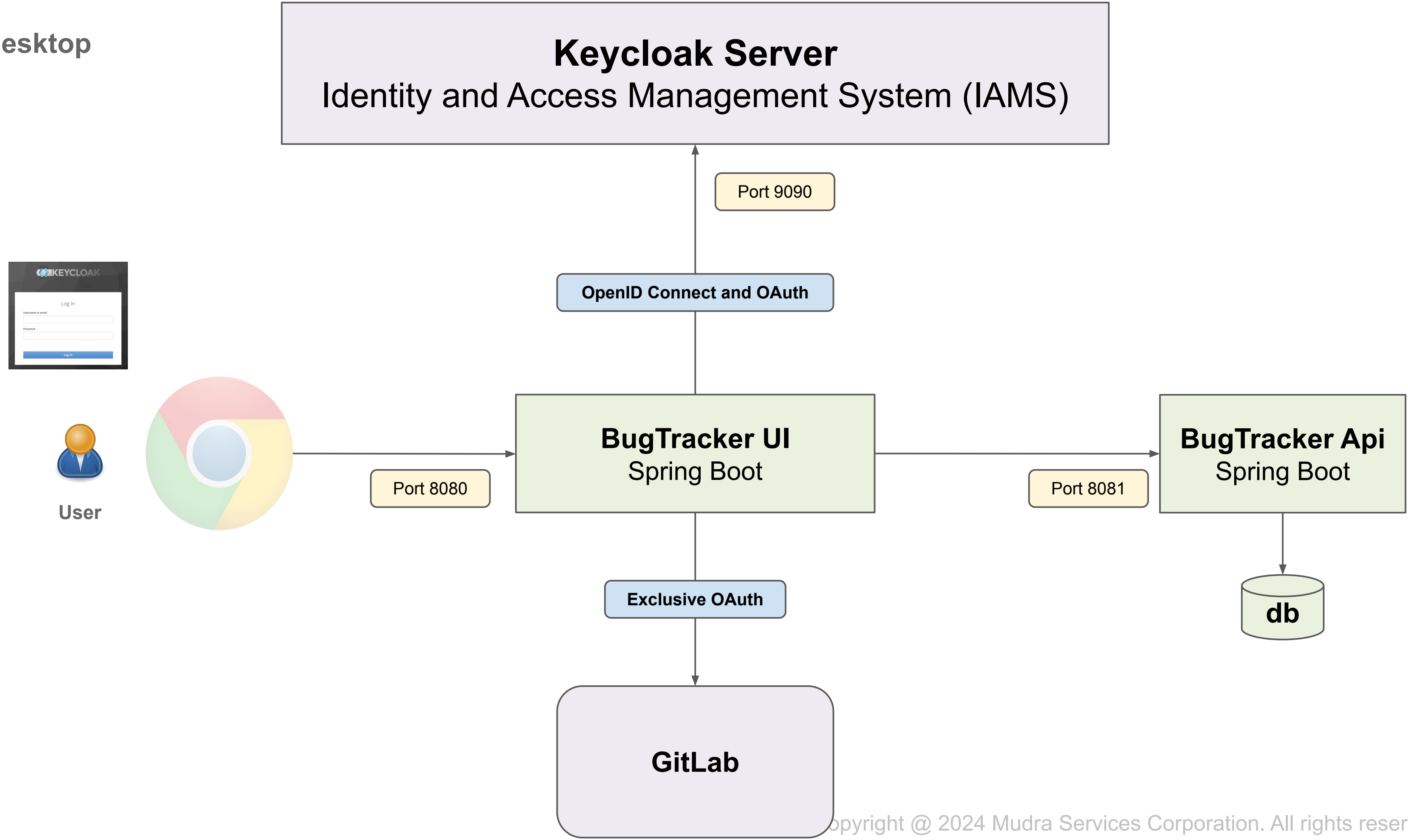
BugTracker Api
Spring Boot

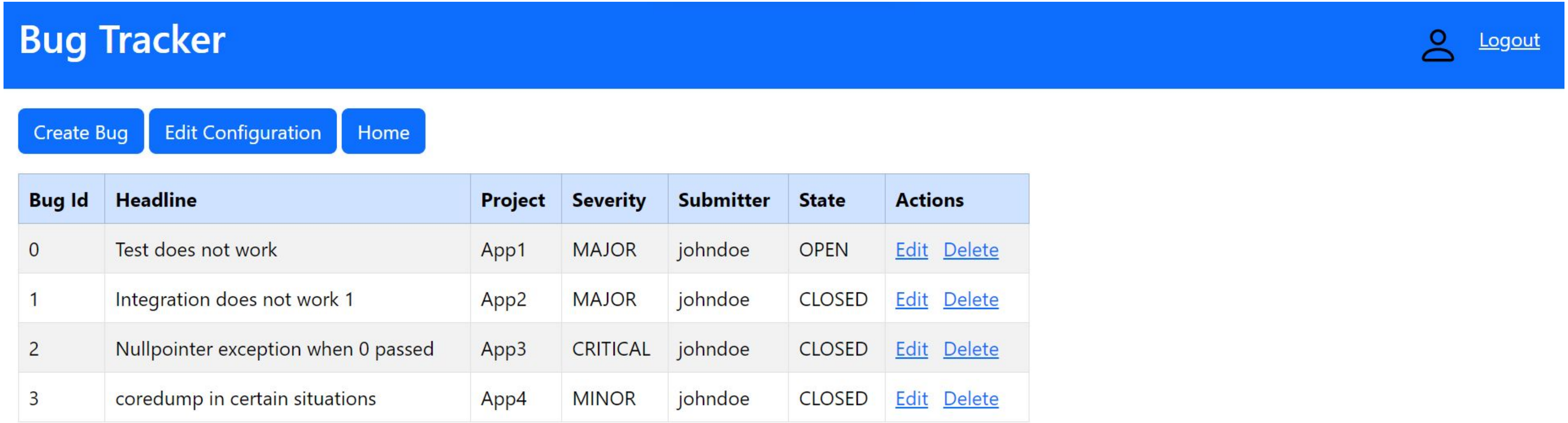
Port 8081

db

User

Desktop


Keycloak Server
Identity and Access Management System (IAMS)
Port 9090
OpenID Connect and OAuth
KEYCLOAK
Log In
Username or email
Password
Log In
User
Port 8080
BugTracker UI
Spring Boot
Port 8081
BugTracker Api
Spring Boot
db
Exclusive OAuth
GitLab



Bug Id	Headline	Project	Severity	Submitter	State	Actions
0	Test does not work	App1	MAJOR	johndoe	OPEN	Edit Delete
1	Integration does not work 1	App2	MAJOR	johndoe	CLOSED	Edit Delete
2	Nullpointer exception when 0 passed	App3	CRITICAL	johndoe	CLOSED	Edit Delete
3	coredump in certain situations	App4	MINOR	johndoe	CLOSED	Edit Delete

Conclusion