Western Norway
University of
Applied Sciences

DAT152 – Advanced Web Applications

Authentication and Authorization

Part 3 – Single Sign On

Goal for Today

Understand the OAuth2 and OIDC auth/authz framework
Understand the different grant types and purposes
Understand implementations of the schemes

Appendix: Oblig #2 (Cont'd + Q&A)

Bihis, Charles. *Mastering OAuth 2.0*. Packt Publishing Ltd, 2015. (pdf version available online)

OAuth2 Protocol

- A Third-Party authentication/authorization protocol
- OAuth 2.0 is a protocol that allows distinct parties to share information and resources in a secure and reliable manner

Third Party – OAuth/OpenID

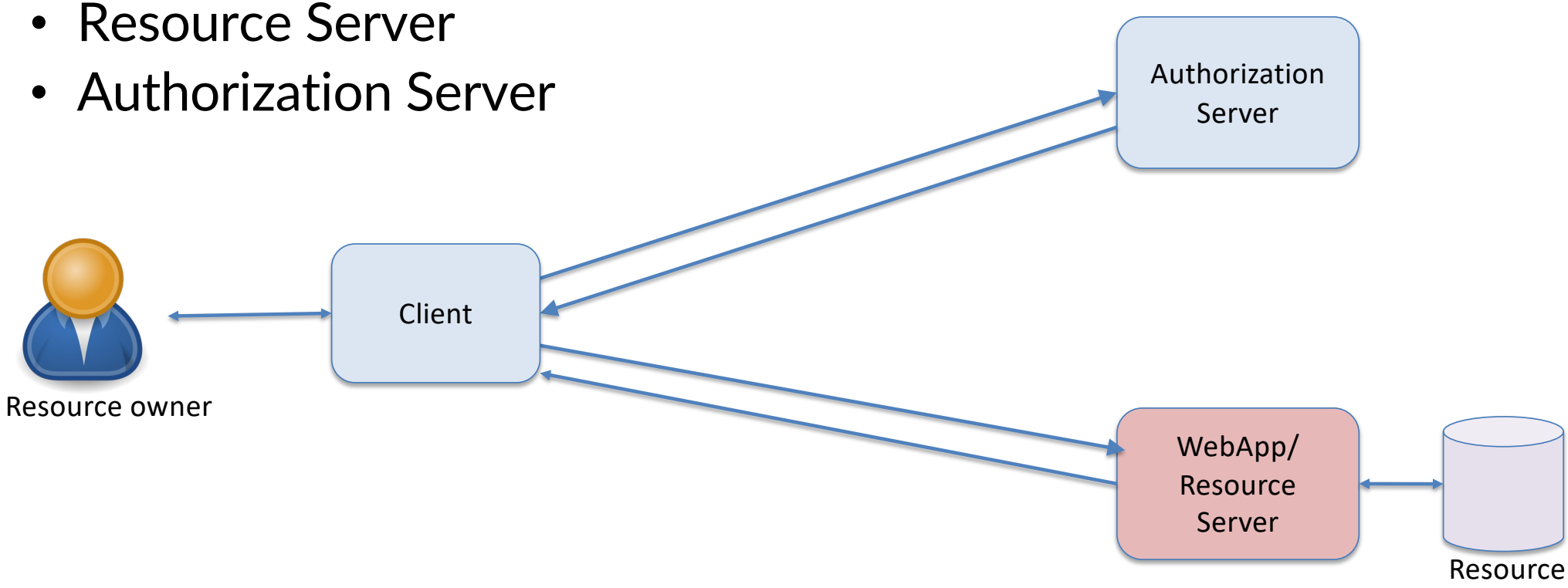
- Federated identity and Delegated authority

Federated Identity/Delegated Authority

- It refers to the concept that allows one service provider to allow authentication of a user using their identity with another service provider. Also known as single sign-on (SSO)
- The ability for a service or application to gain access to a user's resources on their behalf.
  - e.g., using Facebook for single sign-on to Amazon
  - LinkedIn suggesting contacts for you to add by looking at your Google contacts
  - or SSO on all google products (Youtube, Google Doc, Meet, Drive, Map, etc.)

Parties

- Resource Owner
- Client
- Resource Server
- Authorization Server

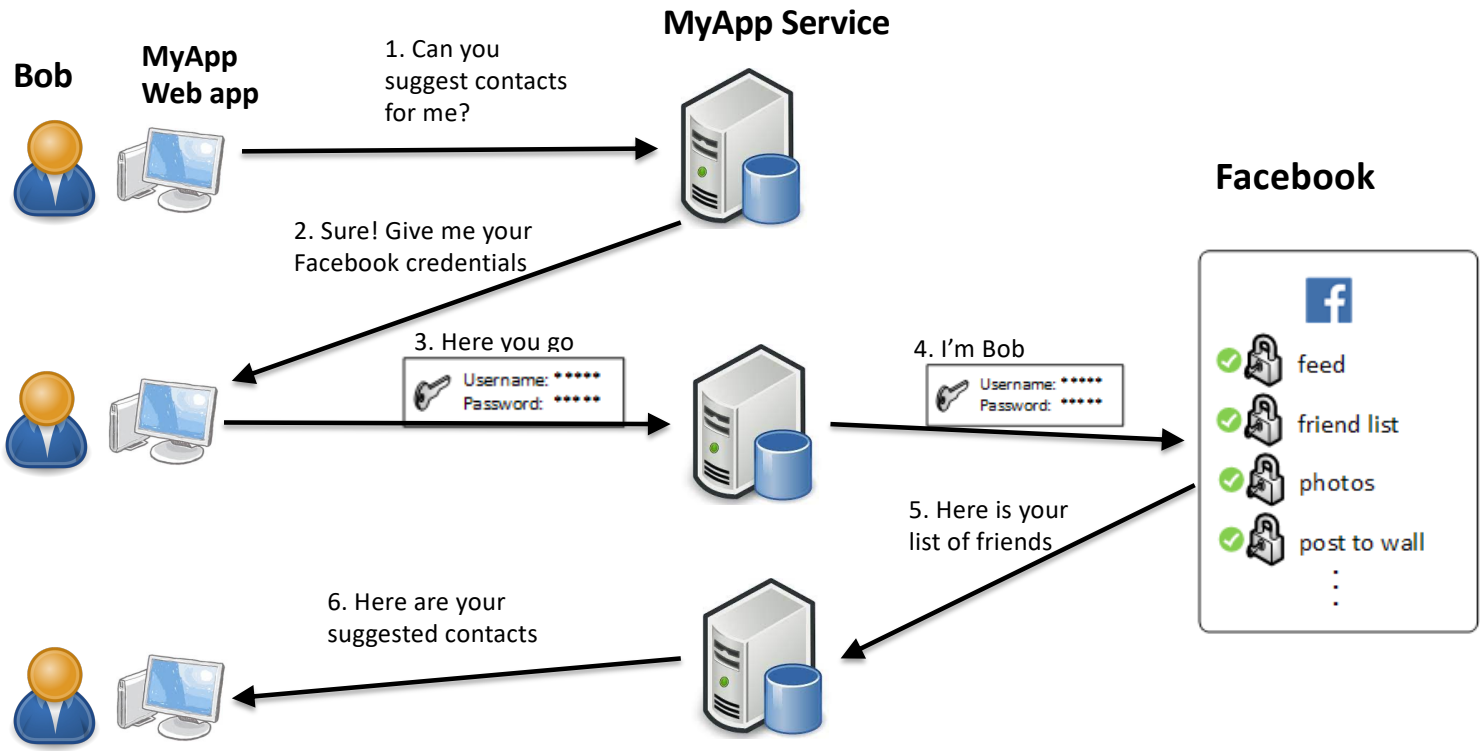


Parties

- **Resource Owner**: A person or system capable of granting access to a protected resource.
- **Application**: A client that makes protected requests using the authorization of the resource owner.
- **Authorization Server**: The Single Sign-On server that issues access tokens to client apps after successfully authenticating the resource owner.
- **Resource Server**: The server that hosts protected resources and accepts and responds to protected resource requests using access tokens. Apps access the server through APIs.

Example – Delegated Authority (without Oauth)

MyApp wants to suggest contacts by looking at your Facebook friends

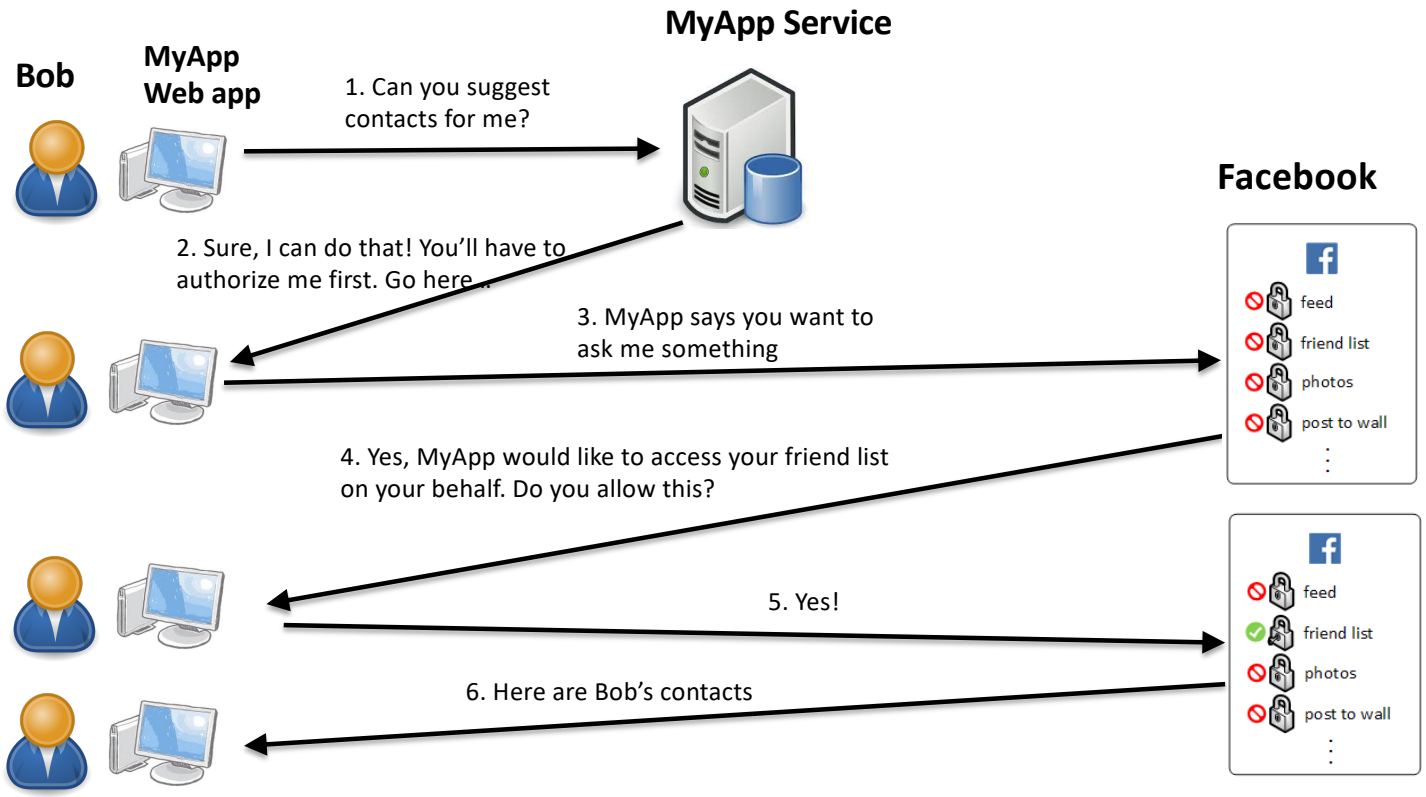


Challenges

- Bob has given his Facebook credentials to MyApp.
- MyApp can do anything with Bob’s credentials
- MyApp can save Bob’s password in an insecure way
- No way for Bob to revoke Facebook access if MyApp is acquired by evil org.
- …

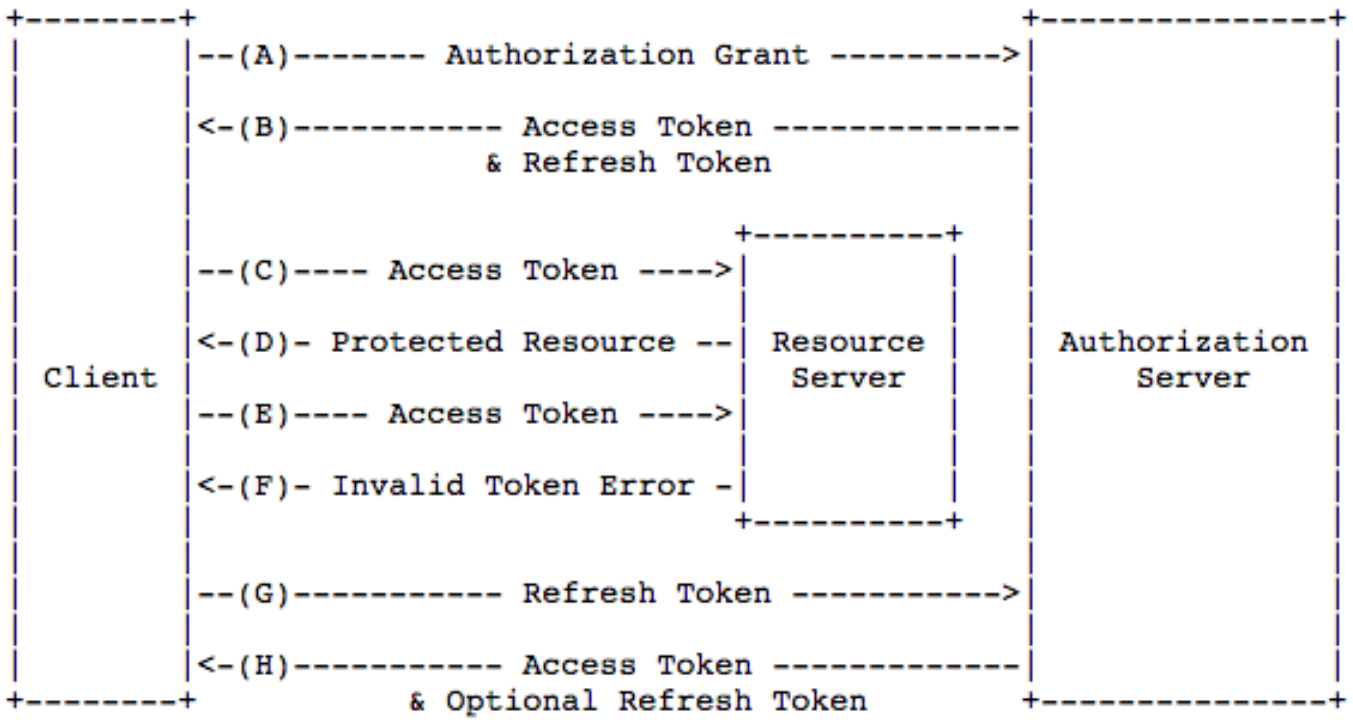
Example – Delegated Authority (with Oauth)

MyApp wants to suggest contacts by looking at your Facebook friends



OAuth2 Framework

- Oauth framework allows a client to negotiate an access token from an authorization server in order to access a protected resource
- Heavily depends on Json Web Token



OAuth2 Framework

- In OAuth 2.0, **grants** are the set of steps a Client has to perform to get resource access authorization.
- Grant Types
  - Authorization code grant
  - Implicit grant
  - Resource Owner Credentials Grant
  - Client Credentials Grant Type

Trusted (confidential) vs. untrusted (public) clients

- Oauth2 providers care about two levels of trust to decide what grant type and flow will be used by a client
  - Trusted and untrusted
- The categorization of a client into either of these two trust levels is determined by:
  - Ability to securely store information
  - Ability to securely transmit information

https://datatracker.ietf.org/doc/html/rfc6749#section-2.1

Trusted vs. untrusted clients

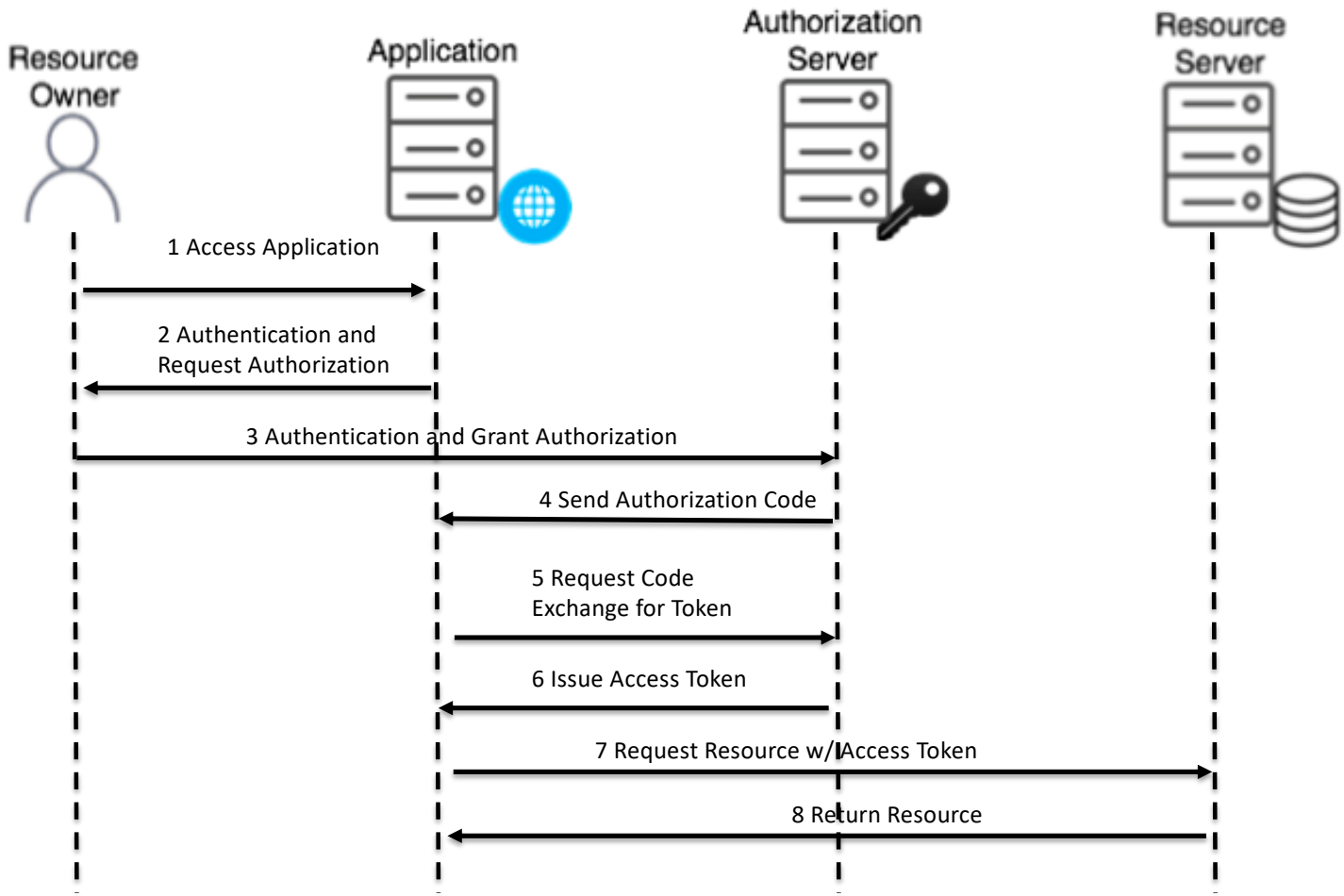
- A **trusted (confidential) client** is an application that is capable of securely storing and transmitting confidential information.
  - Can thus, be trusted to store their client credentials
  - e.g., a 3-tier client-server-database application.
- An **untrusted (public) client** is one which is incapable to securely store or transmit confidential information
  - A browser-based application
  - Native apps

https://datatracker.ietf.org/doc/html/rfc6749#section-2.1

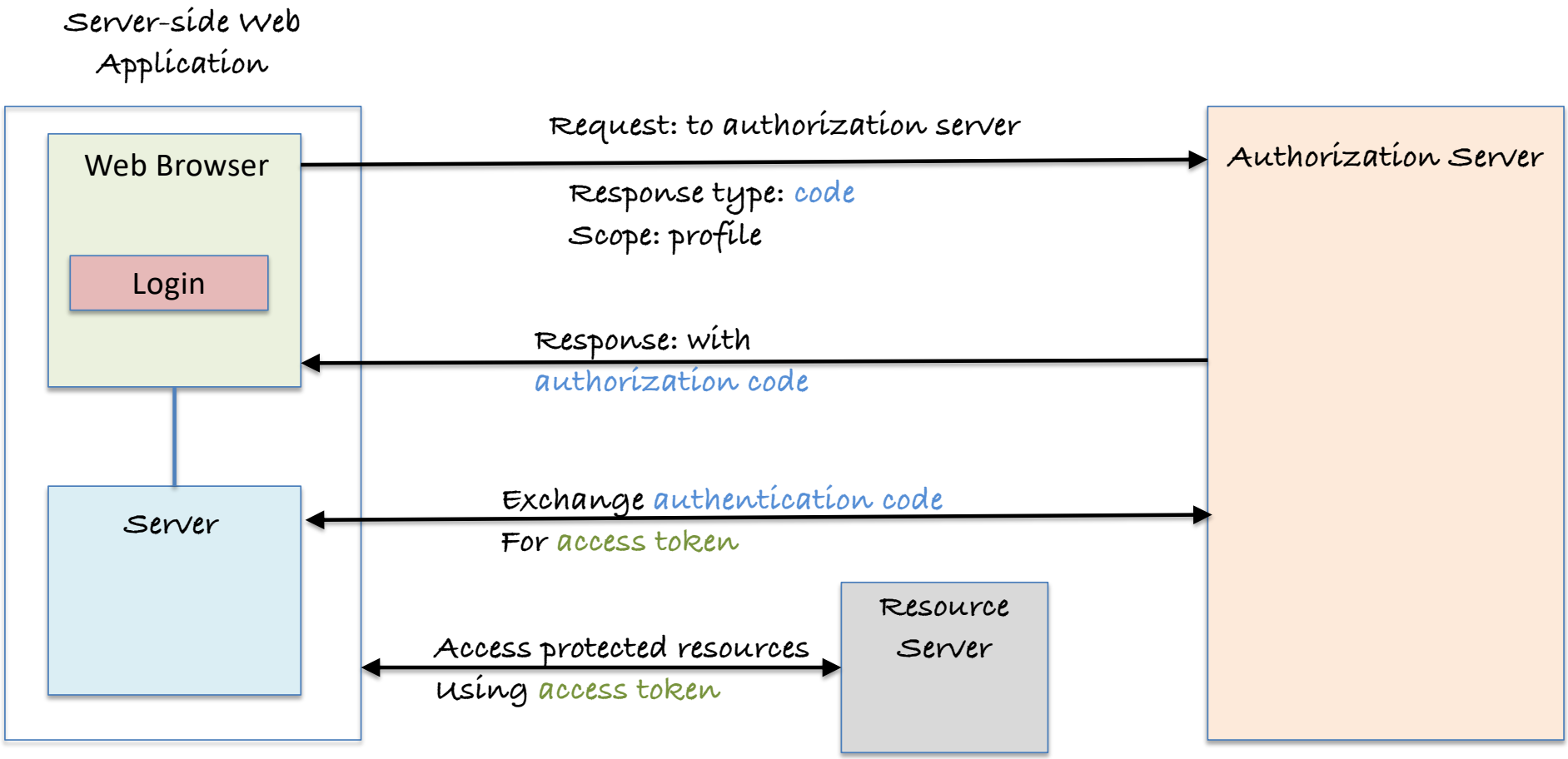
Grant Types

- **Authorization Code Flow**: used by Web Apps executing on a server.
- **Implicit Flow**: used by JavaScript-centric apps (Single-Page Applications) executing on the user's browser.
- **Resource Owner Password Flow**: used in situations where resource owner 'consent' is not needed.
- **Client Credentials Flow**: used for machine-to-machine communication.

Authorization Code Grant Type

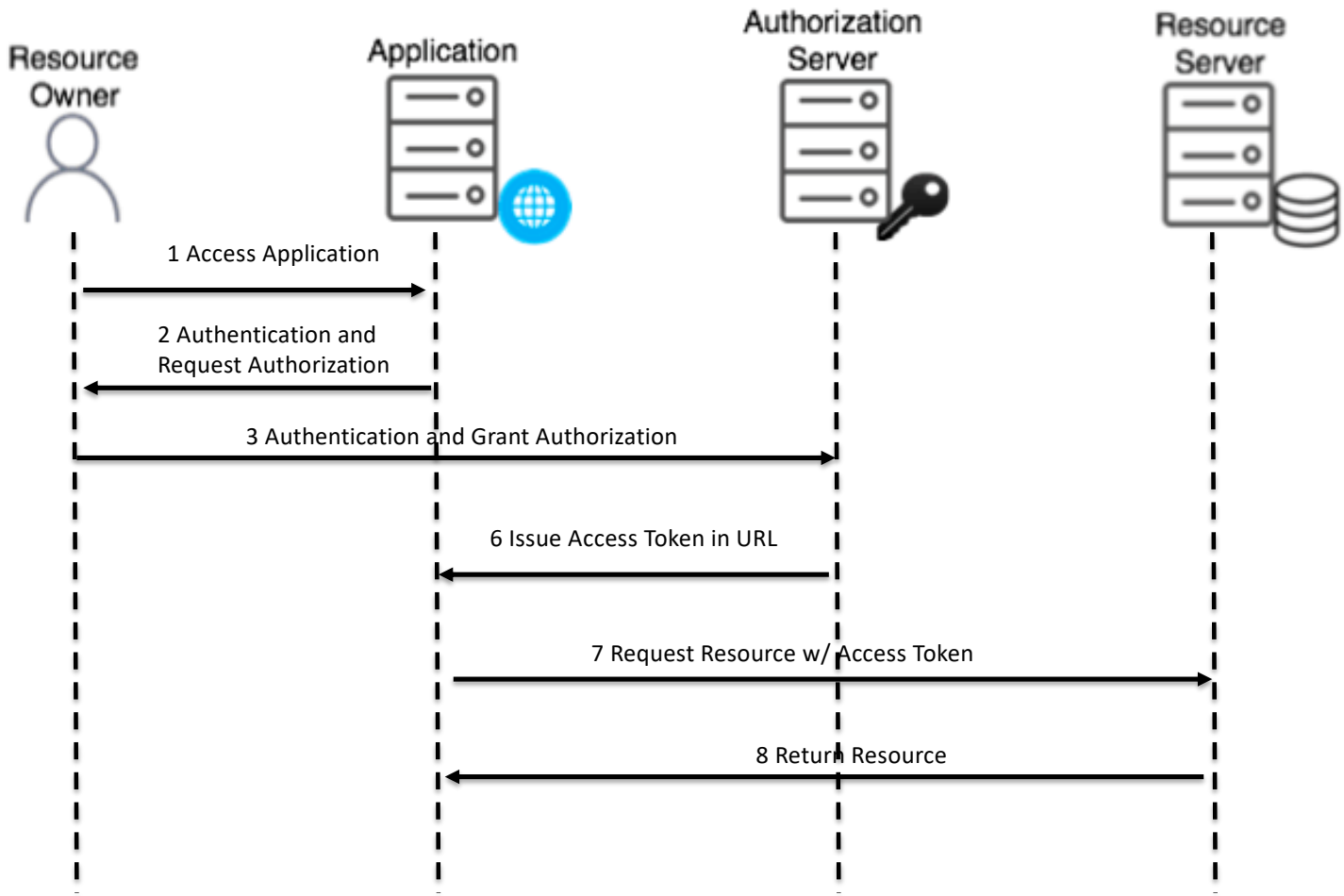


Authorization Code Flow

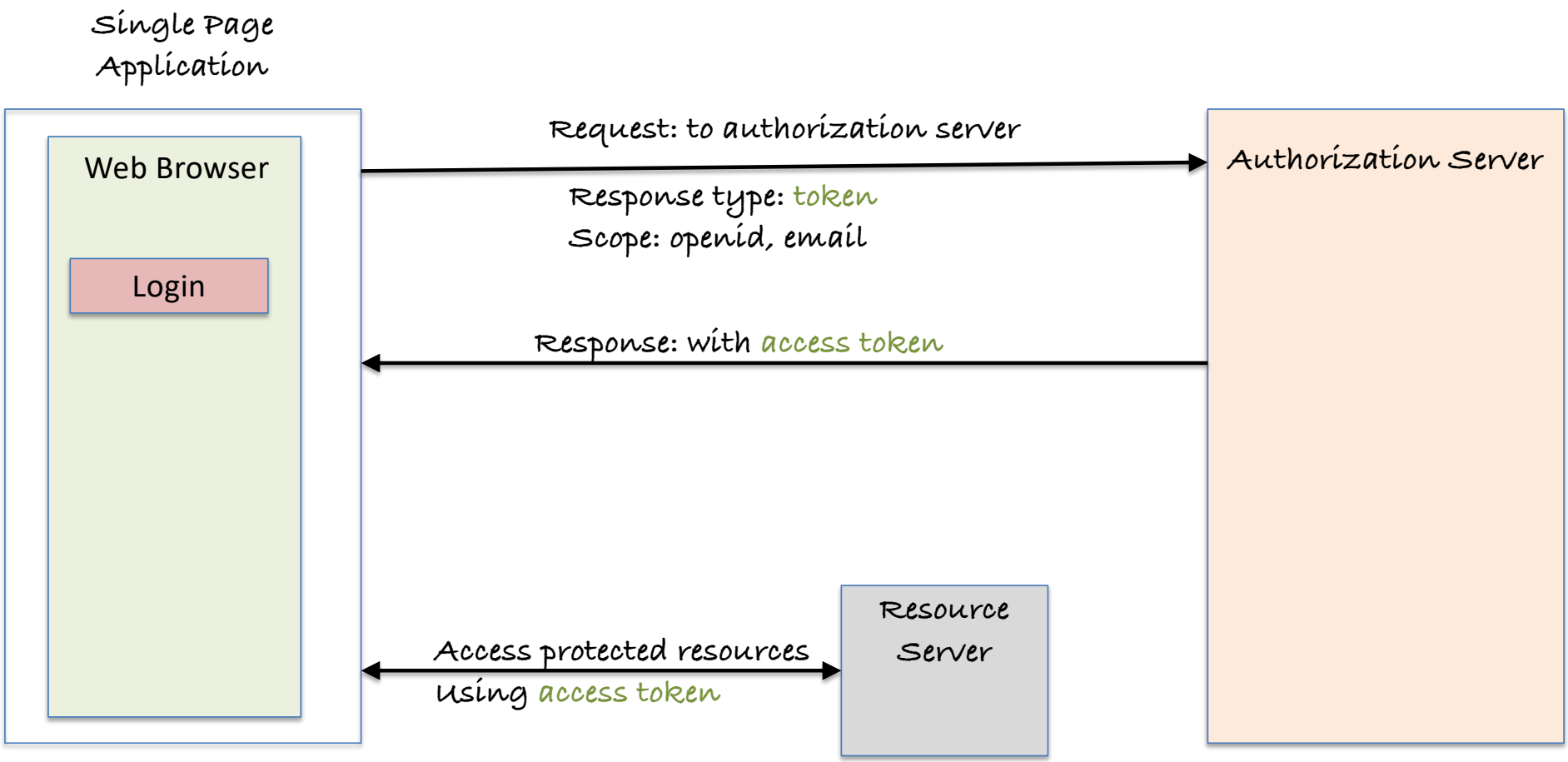


Adapted from: https://dev.to/shahbaz17/oauth-20-openid-connect-41d

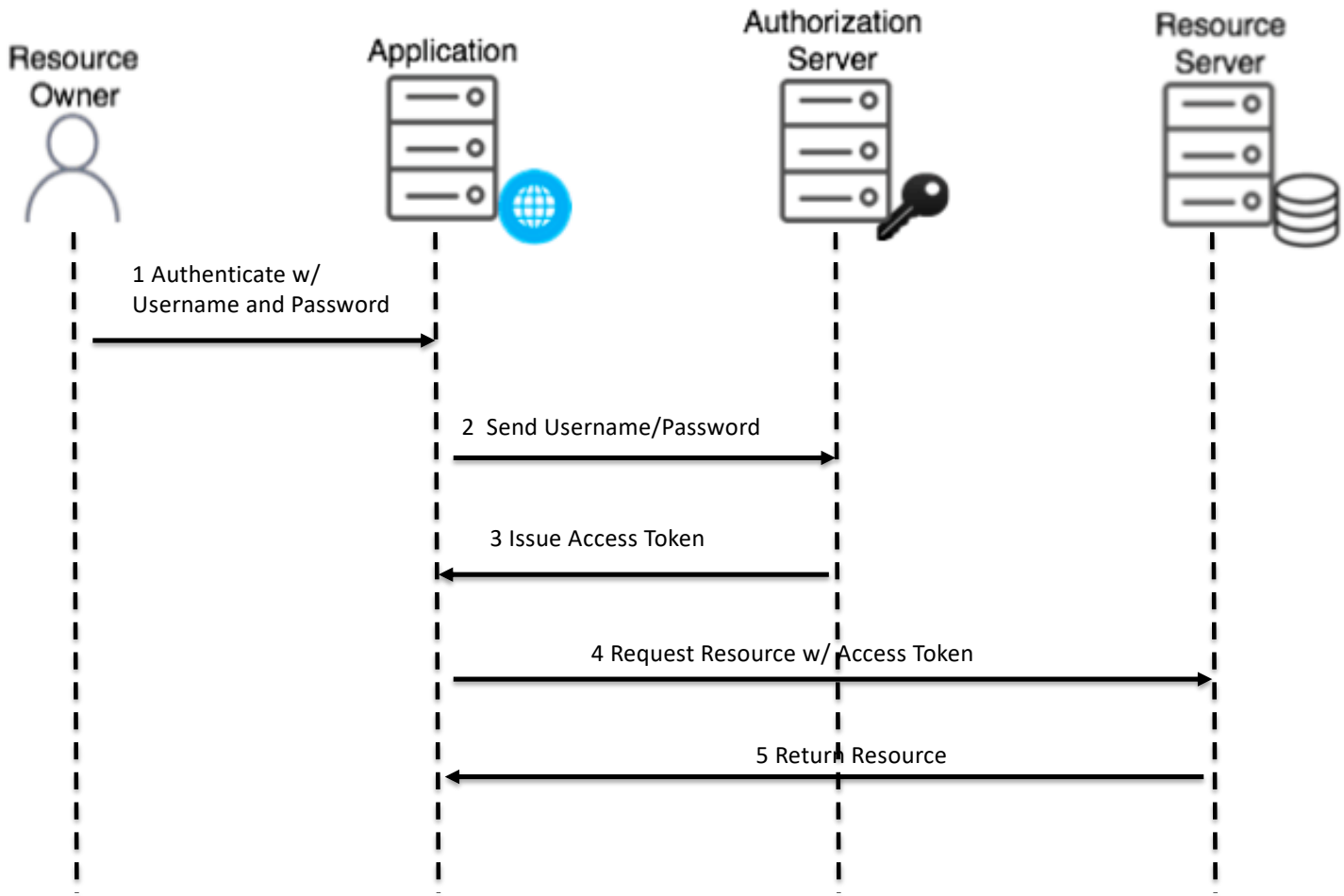
Implicit Grant Type



Implicit Flow

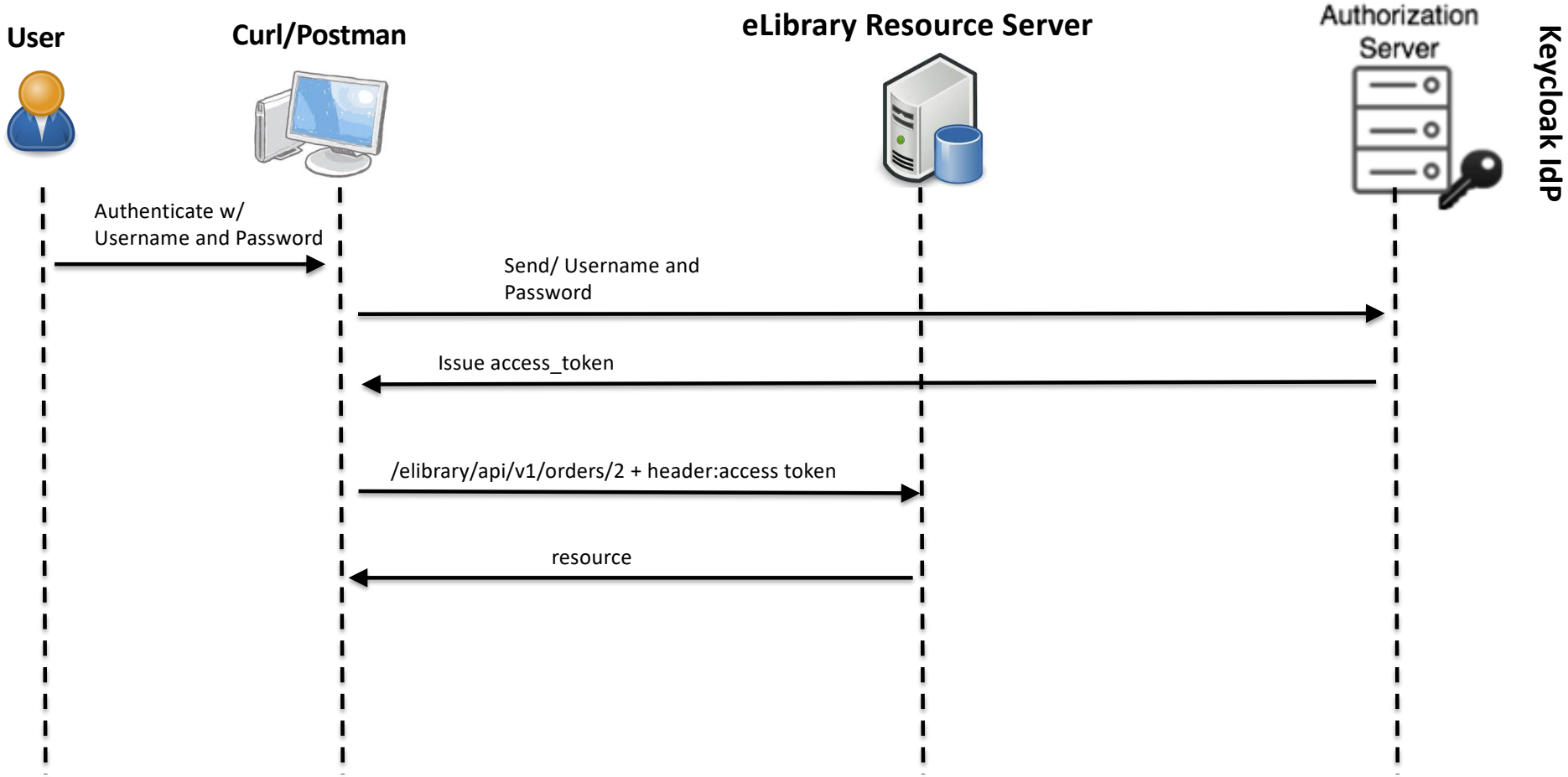


Resource Owner Password Grant Type



Resource Owner Password Grant

Curl/Postman wants to view a borrowed book from the eLibrary REST API server



Auth/Authz Flow

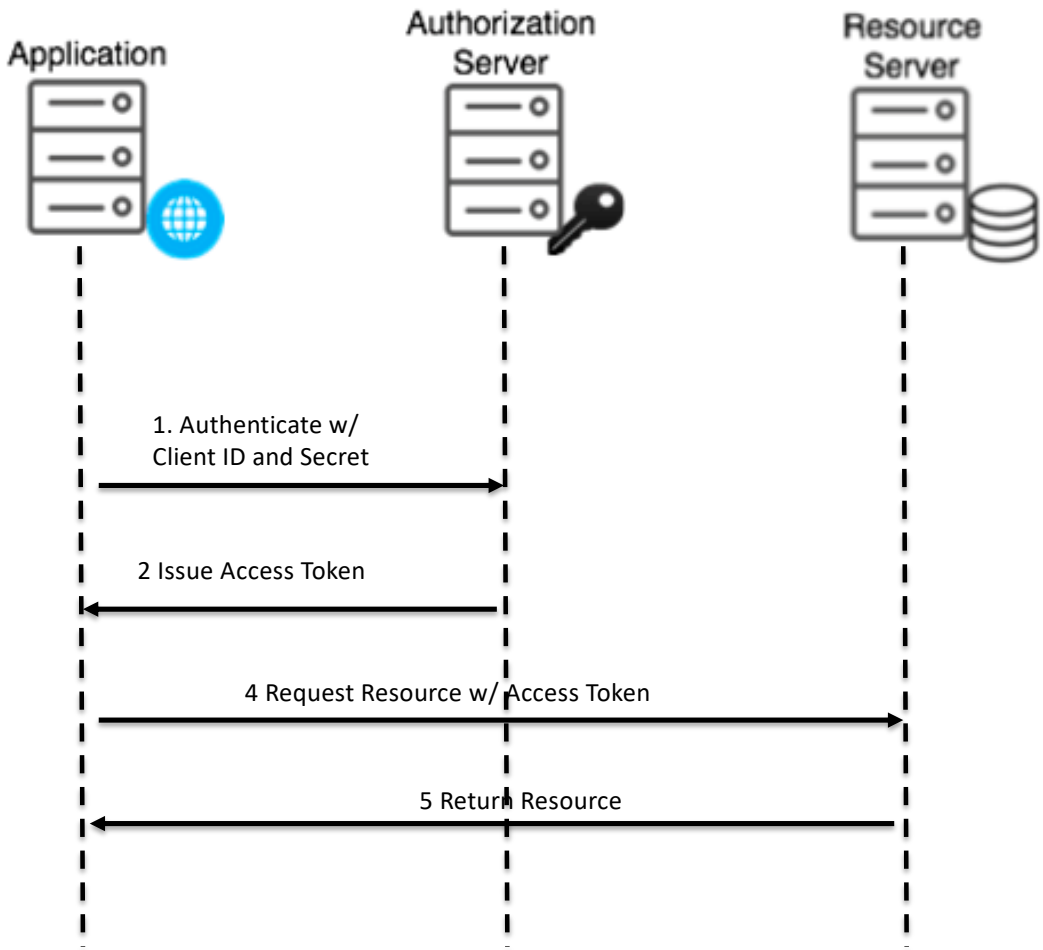
```
curl -X POST http://localhost:8080/realms/DAT152/protocol/openid-connect/token --data
'grant_type=password&client_id=dat152oblig2&username=user1&password=user1'
```


```
{
    "access_token":
        "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJyb2JlcnRAZW1haWwuY29tIiwiaXNzIjoiREFUMTUyLUxlY3R1cmVyQFRET1kiLC
        JmaXJzdG5hbWUiOiJSb2JlcnQiLCJsYXN0bmFtZSI6IklzYWFjIiwicm9sZXMiOlsiVVNFUiJdLCJpYXQiOjE2OTYwNDEy
        ODEsImV4cCI6MTY5NjEyNzY4MX0.NDoPBqNiQNlxIF8mmnjxMJ_QQfrmVPU6H38Ez1fsg-c",
    "expires_in": 900, "refresh_expires_in": 3600,
    "refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI2NzFhOWY5My1hYmYxLTRiYmItODg2ZC1lN2UxYjI
    0ZTZkNjYifQ.eyJleHAiOjE2OTYwNTI...”,
    "type": "Bearer"
}
```


```
curl -v -H "Authorization: Bearer <accesstoken>" localhost:8090/elibrary/api/v1/orders/2
```


```
{
    "id": 2,
    "isbn": "abcde1234",
    "expiry": "2023-10-21"
}
```

Client Credentials Flow

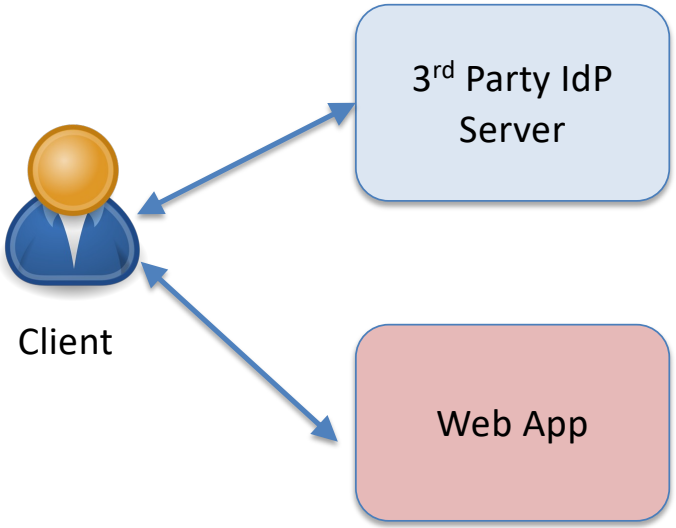


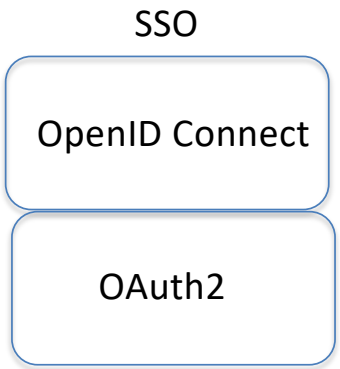
Differences

Grant	Simplicity	Security	Access duration
Server-side flow (authorization code grant flow)	**More complex**: In order to facilitate the secure storage and transmission of confidential data, a backend server and data store must be maintained.	**More secure**: The server-side flow never exposes the key to the browser, and so has a significantly smaller chance of being leaked.	**Long-term**: Because an application using the authorization code grant flow is trusted to store confidential information, it can store properties needed for long-term, even offline, access.
Client-side flow (implicit grant flow)	**Less complex**: Due to the more relaxed requirements around security for untrusted applications, no backend server or data store is required. Everything can happen from the browser.	**Less secure**: The key is passed directly to the browser and so has a much larger chance for this key to be obtained by unauthorized parties.	**Short-term**: Since applications using the implicit grant flow are considered untrusted, they should only be given short-lived tokens due to the increased likelihood of such tokens being leaked.

OAuth2 Scenarios

- Single Sign On
  - OpenID connect
- Grant Flows
  - Authorization Code Flow
  - Implicit Grant
- Use cases
  - Used for Web Applications with user interfaces (UI)
  - Uses OpenID token



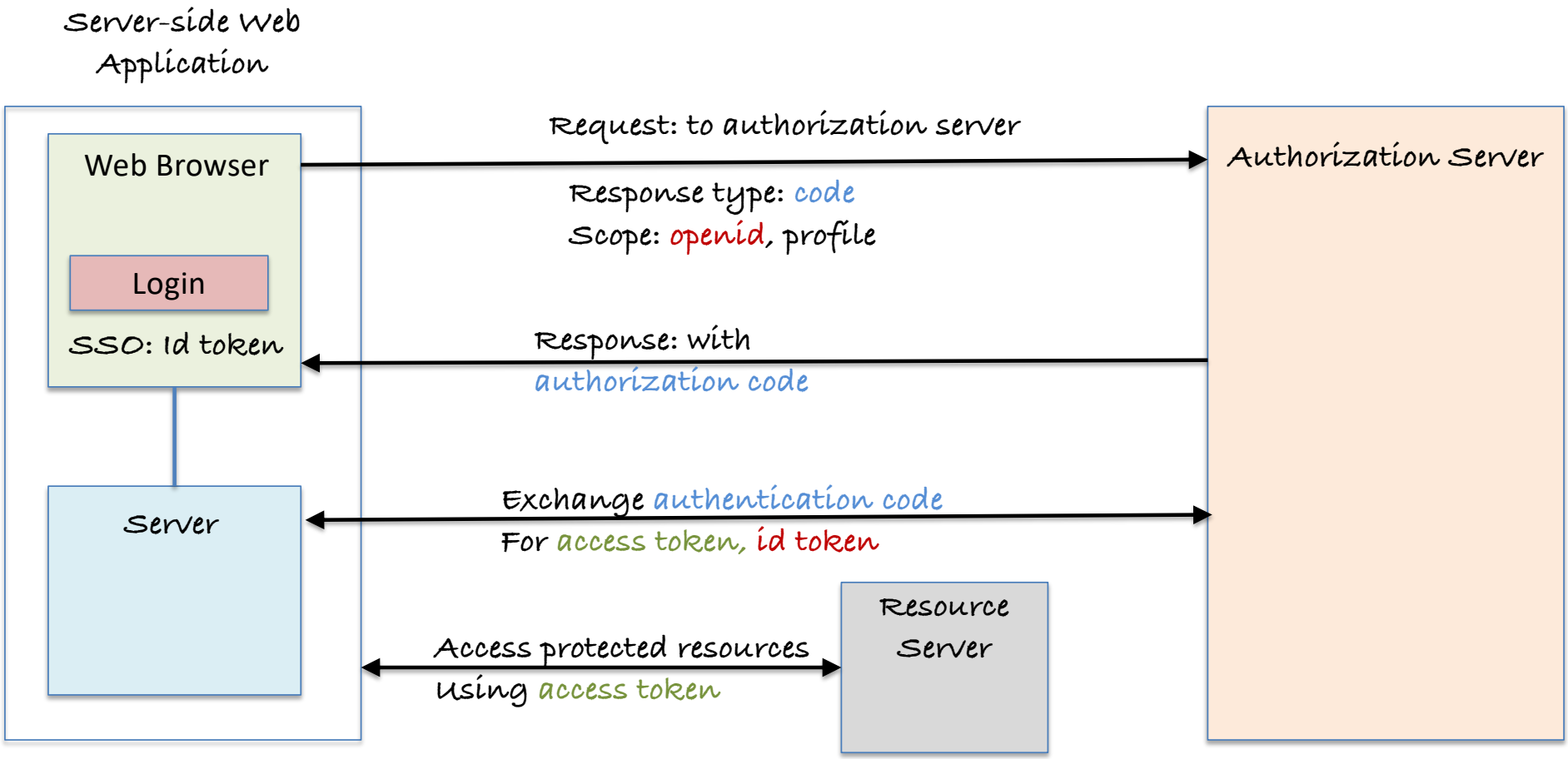


OIDC Authorization code grant - SSO

- OpenID Connect
  - Authentication layer
  - Uses OpenID Token
  - returns claims based on the scope specified by the user
  - Scope: openid, profile, email, etc.

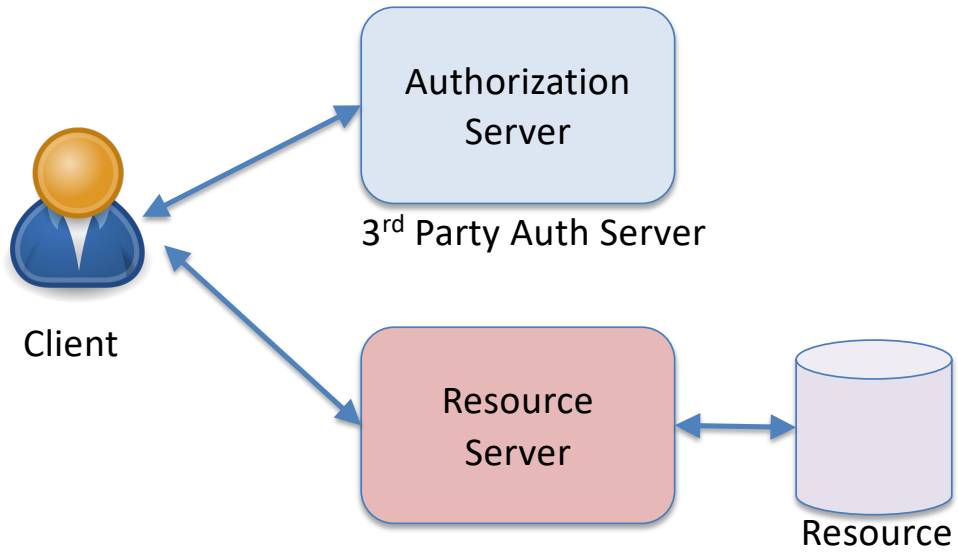

```
{
  "exp": 1763327318,
  "iat": 1759871318,
  "auth_time": 1759871318,
  "jti": "1ef10d6b-1af4-57fc-d3f2-d0193af146ba",
  "iss": "http://localhost:8080/realms/DAT152",
  "aud": "dat152WebSSOApp",
  "sub": "8a0d7313-adfa-4751-aa05-d8ce219e5568",
  "typ": "ID",
  "azp": "dat152WebSSOApp",
  "nonce": "wnWyi9-_IB27hqU80sCHtQxolbTrW8raOw_ya8gKW6M",
  "sid": "7ea708ae-00af-b8ad-cf4b-07e3e7645b92",
  "at_hash": "ZUquCQamyYoscwXmv2hYxQ",
  "acr": "1",
  "email_verified": false,
  "name": "User3_Firstname User3_Lastname",
  "preferred_username": "user3",
  "given_name": "User3_Firstname",
  "family_name": "User3_Lastname",
  "email": "user3@email.com"
}
```

OIDC Authorization Code Flow



OAuth2 Scenarios

- Resource Sharing
  - OAuth2
- Typical grant flows
  - Resource Owner Password Grant
  - Client Credentials Flow
- Use cases
  - To request resources from resource server
  - Uses Access Token

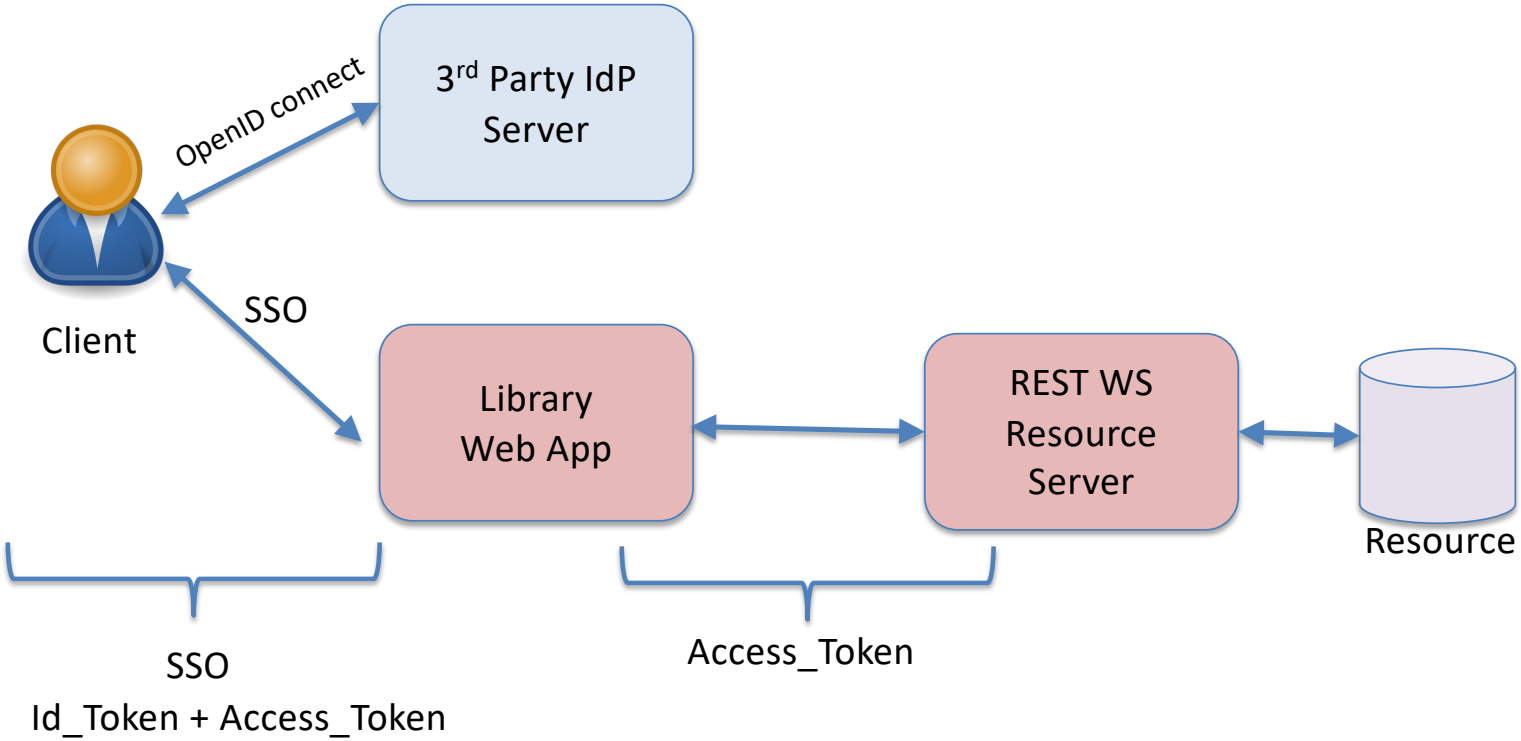


Example – SSO + Resource Sharing

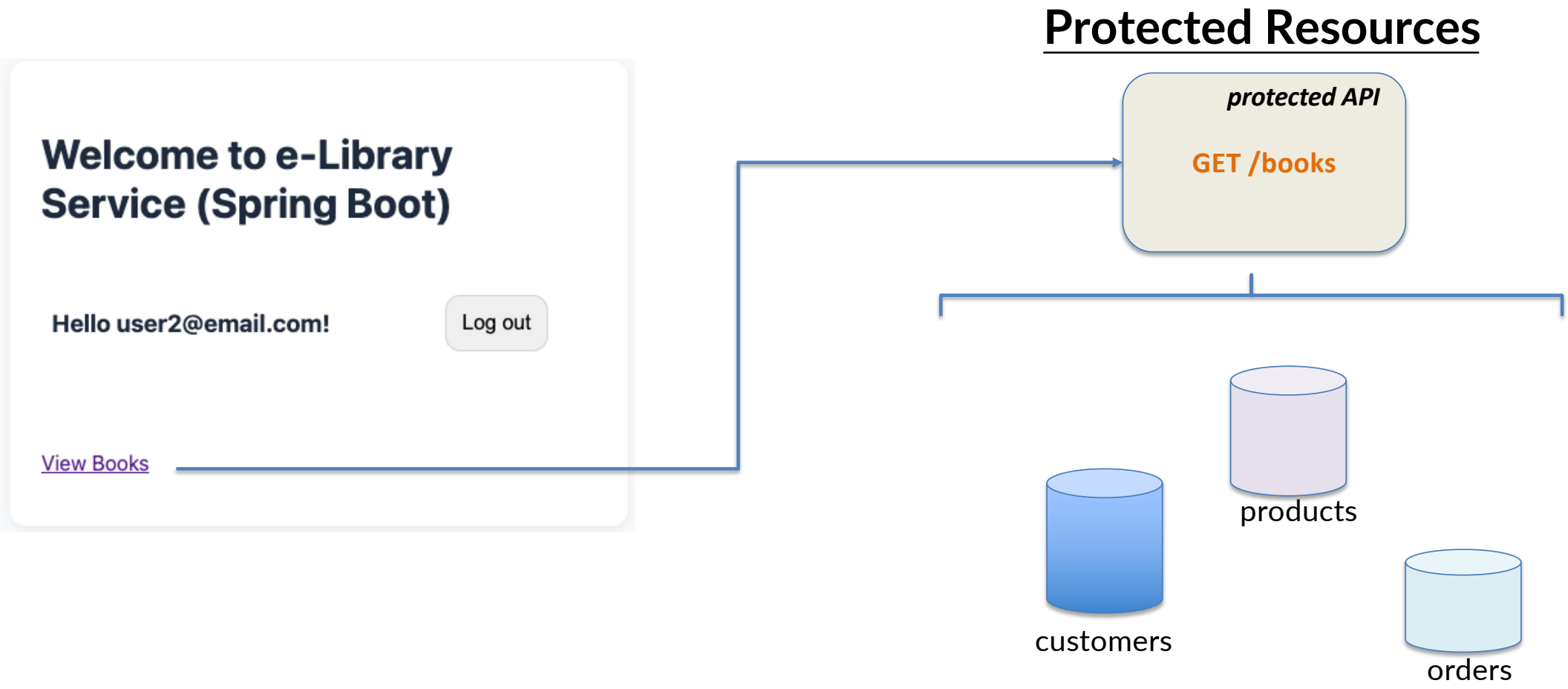
Authorization Code Grant + Resource Owner Password Grant

- User (Resource owner)
- Library Web Application
- Resource Server (REST WS for library reources)
- Identity Server

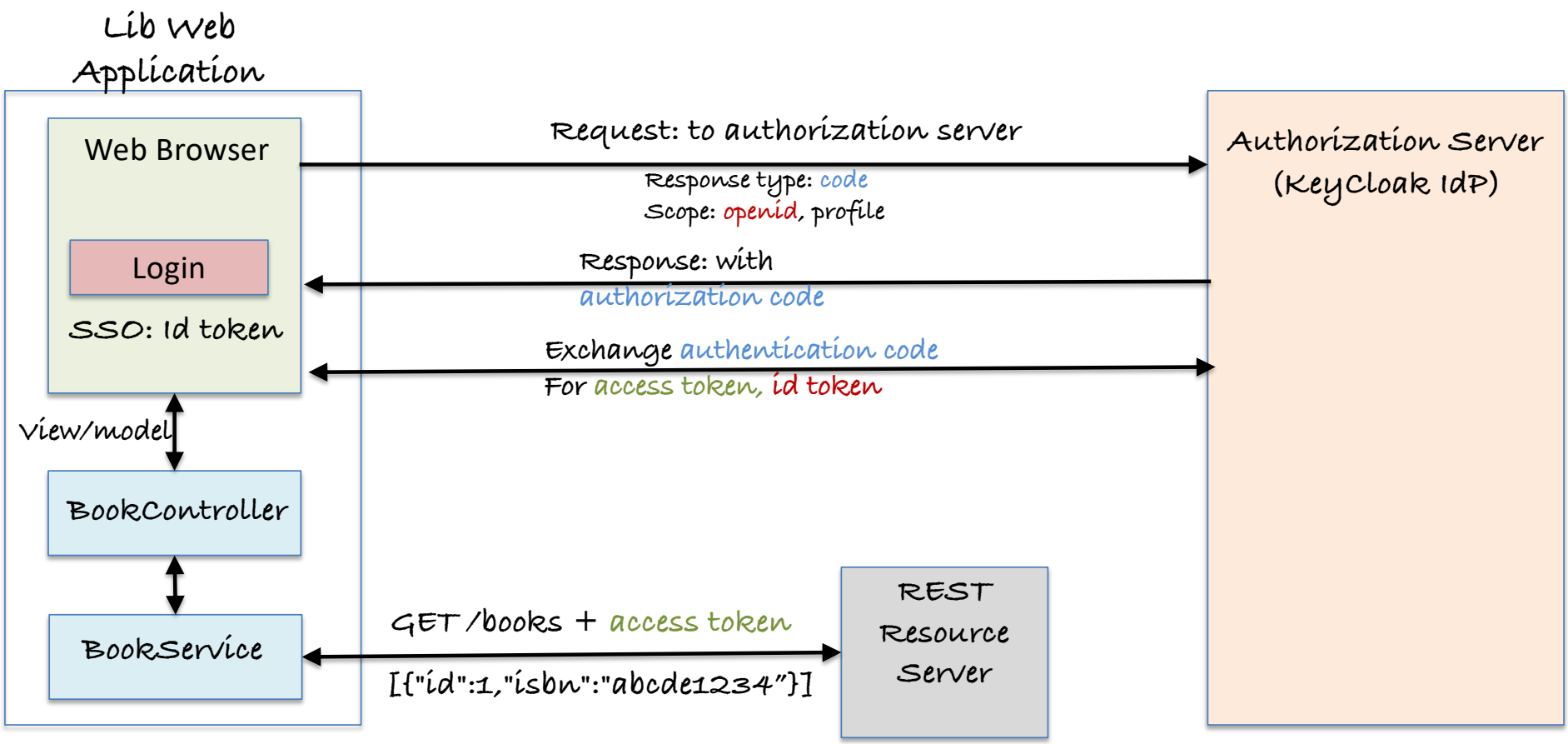
Library Web App - SSO



LibWebApp and REST WS Resource Server



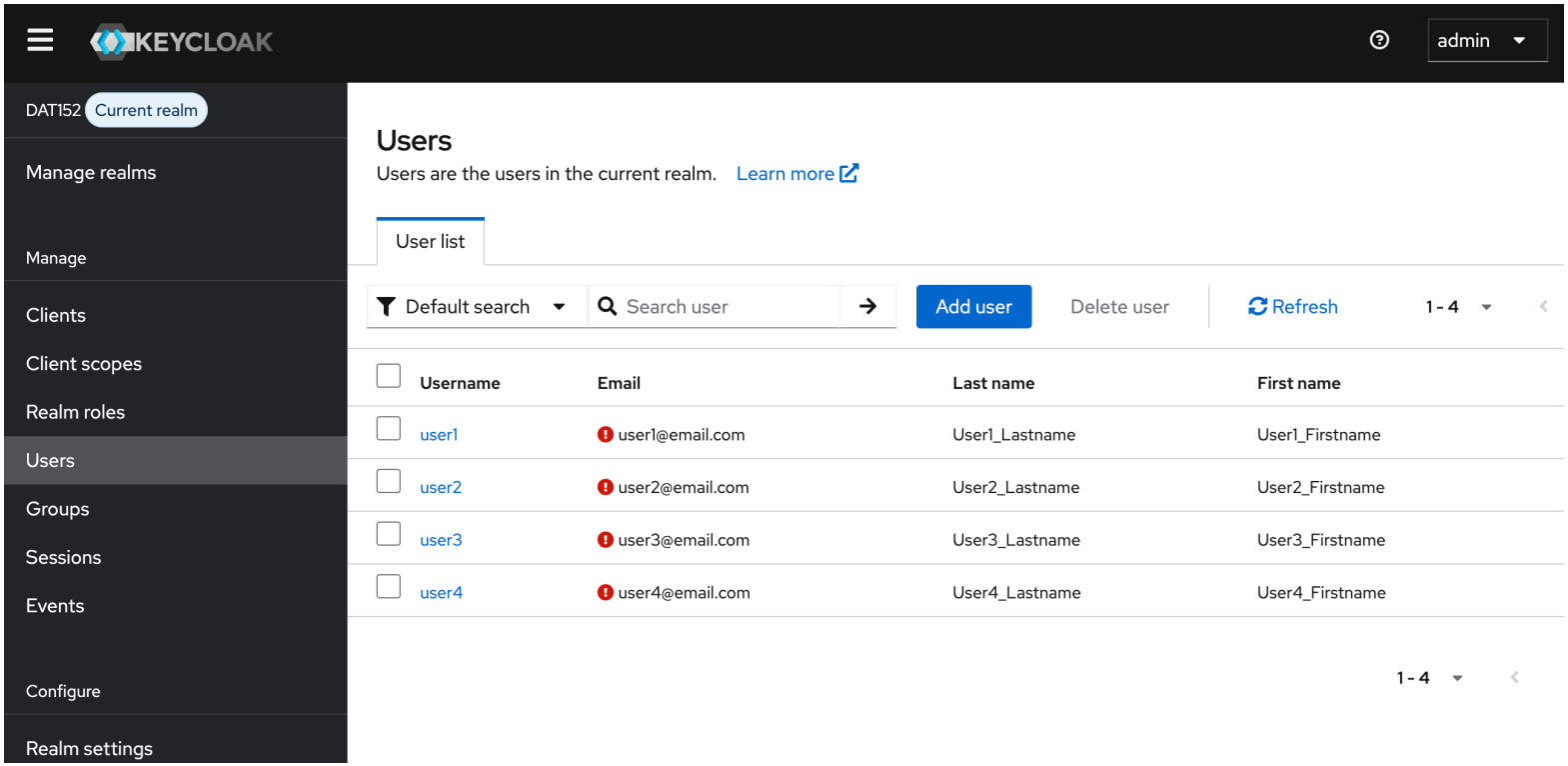
OIDC Authorization Code Flow



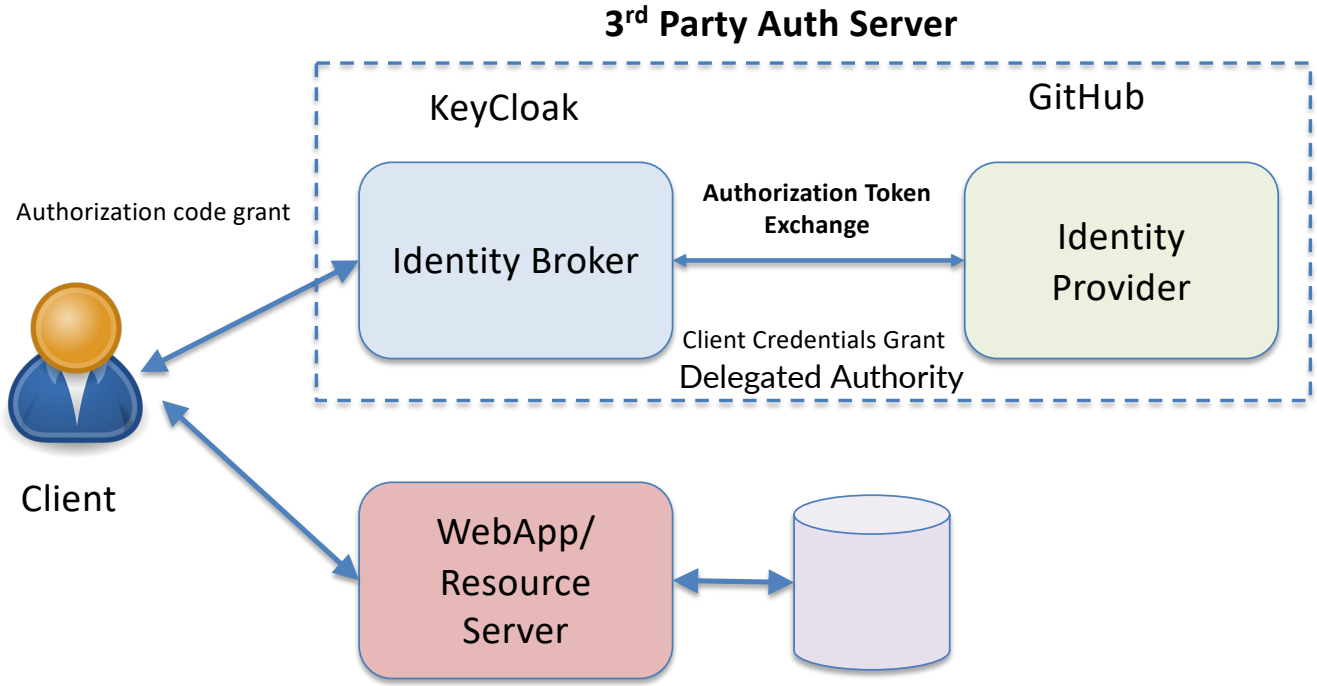
Oblig2

3rd Party Authorization Server

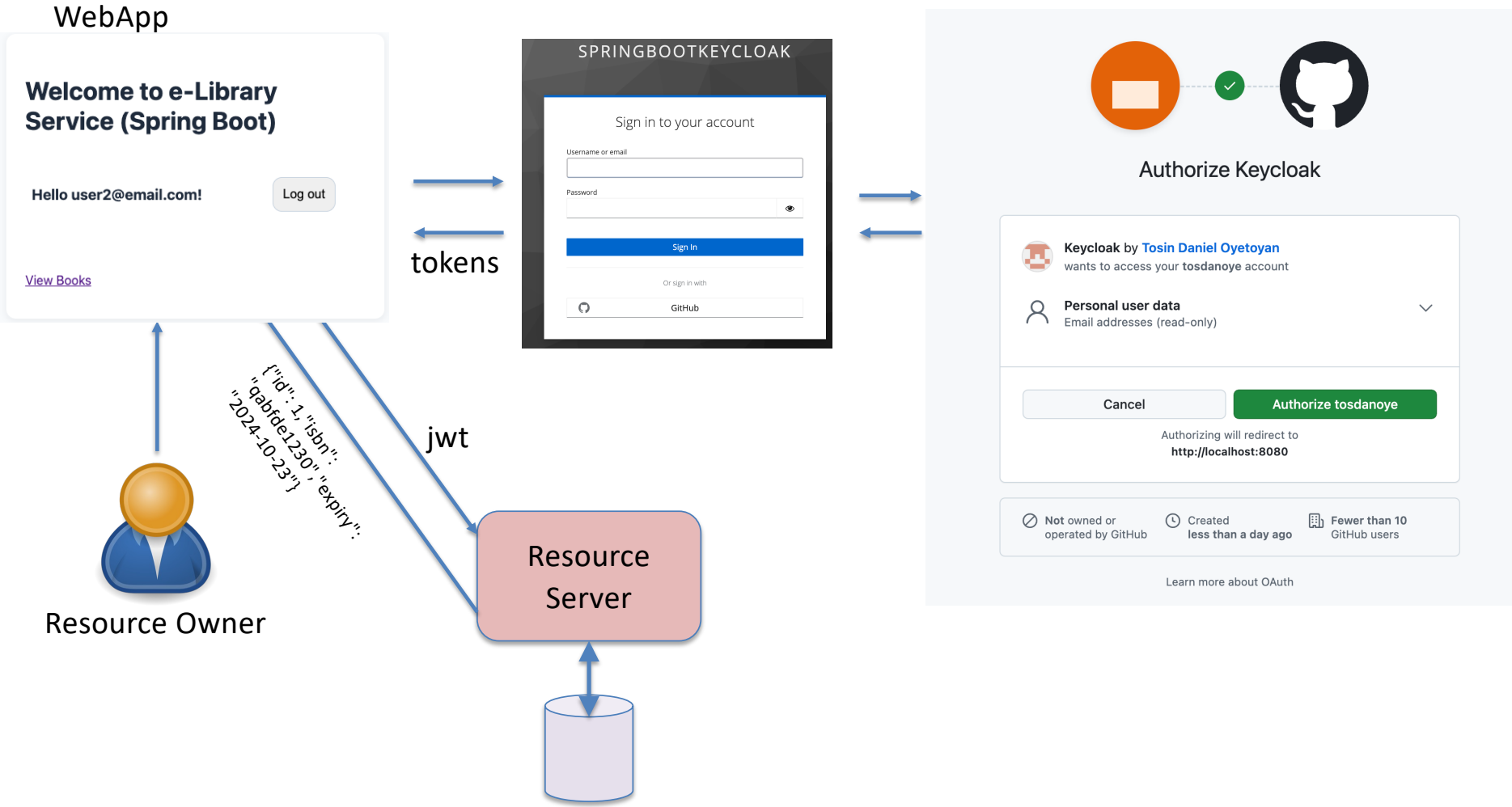
- Example: Keycloak IdP



Authorization code grant – Keycloak/GitHub



Authorization code grant



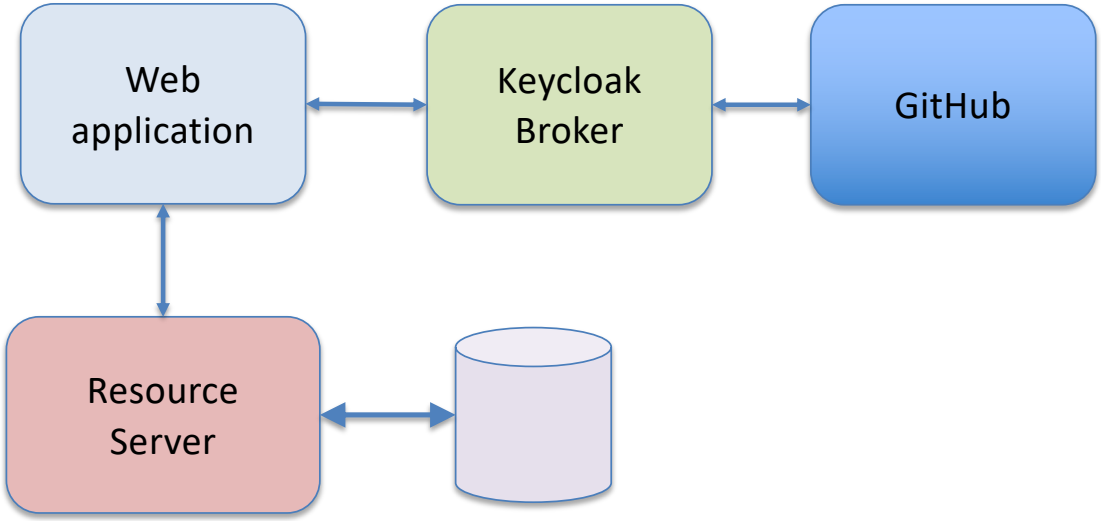
Demo

Exercises

- Using the default code on Canvas (with Keycloak as IdP)
  - Extend the application such that an authenticated user:
    - can send an order (borrow) request to the REST API server and view the created order on the web app.
    - The created order can be updated
    - The order can be cancelled
- Configure the web application to use your github or google account for SSO

Exercises

- Configure Keycloak to broker the identity between GitHub and the Web Application



library-spring-ws-rest-security-oauth

Oblig 2 – Cont’d + Q&A