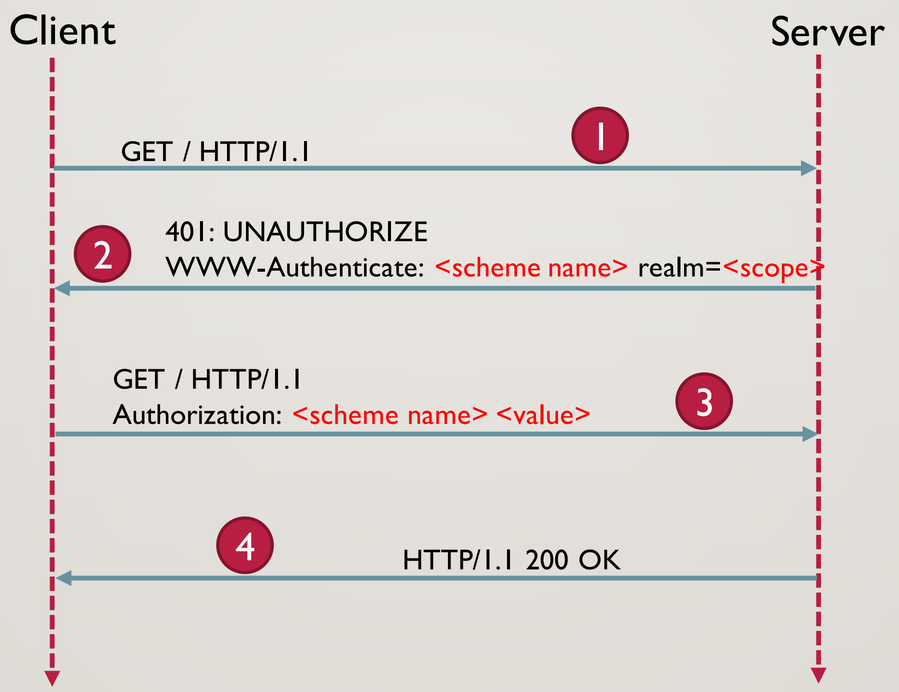
AUTHENTICATION & AUTHORIZATION IN A NUTSHELL

THANH TRAN | FROM EWS-SAAS2

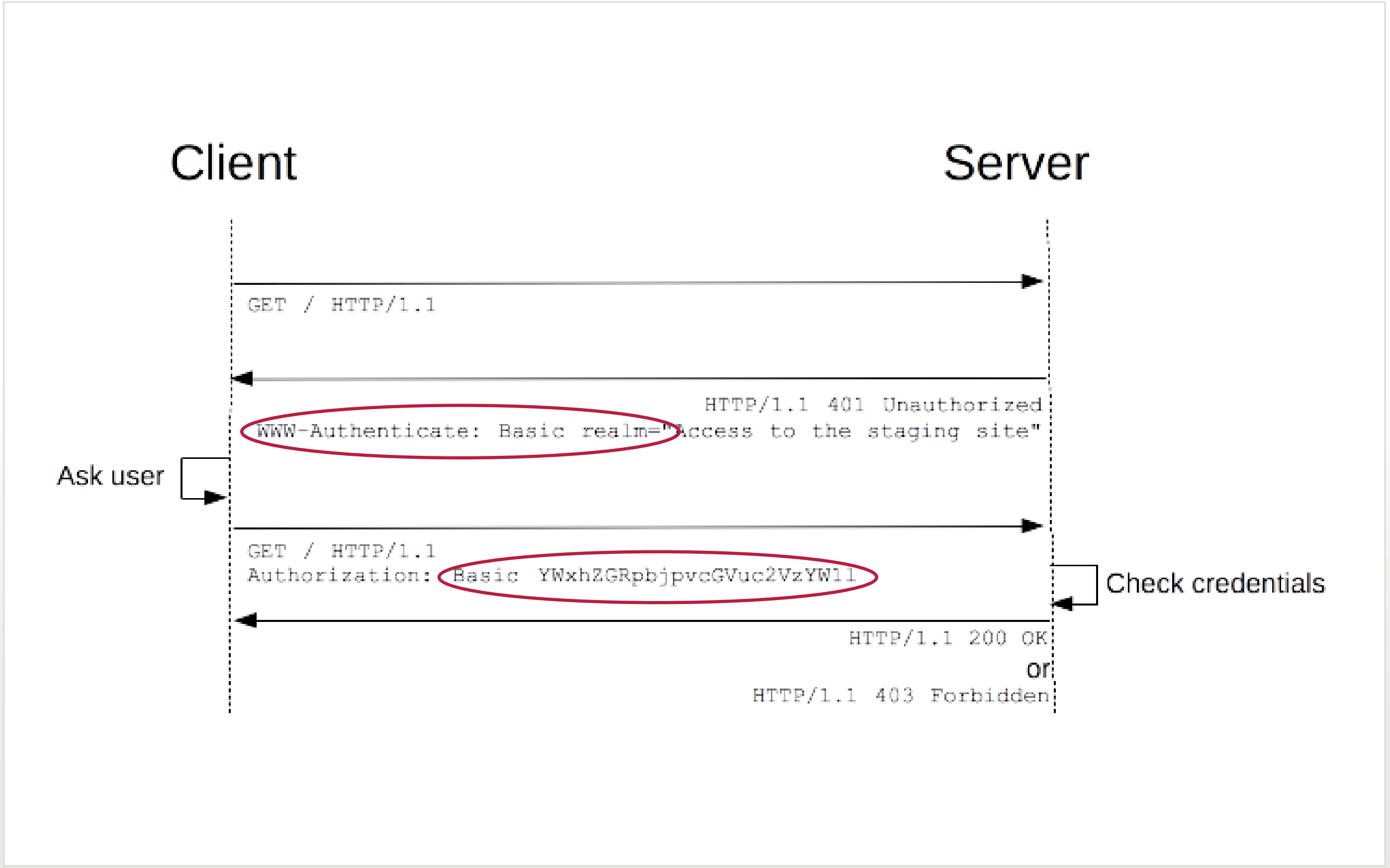
AGENDA (PART 1)

- Basic Authentication
- Bearer Authentication
  - OAuth 1.0
  - OAuth 2.0
- Open Id Connect
  - Jwt (Json Web Token)
- IdentityServer4

BASIC FLOW


Client
Server
1
GET / HTTP/1.1
2
401: UNAUTHORIZE
WWW-Authenticate: <scheme name> realm=<scope>
GET / HTTP/1.1
Authorization: <scheme name> <value>
3
4
HTTP/1.1 200 OK

BASIC AUTHENTICATION



```
GET / HTTP/1.1

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"

GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l

HTTP/1.1 200 OK
or
HTTP/1.1 403 Forbidden
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

TERMINOLOGY

- Realm: indicate a scope of protection.
- Protection space: defined by the canonical root URI of the server being accessed.
  - Ex: http://example.com/docs/index.html
    - Same scope with
      - http://example.com/docs/
      - http://example.com/docs/test.doc
      - http://example.com/docs/?page=1
    - Difference scope with
      - http://example.com/other/
      - https://example.com/abc/

INTERACTION

Client	Server
↳	Validate Authenticate. + Valid - served request + Otherwise - add WWW-Authenticate header with scheme is Basic, include scope
1. obtains the user-id and password from the user 2. Constructs token = user:pass 3. Encodes credential = base64(token) 4. Send Authorization request with Header Authorization: Basic <credential>	↰

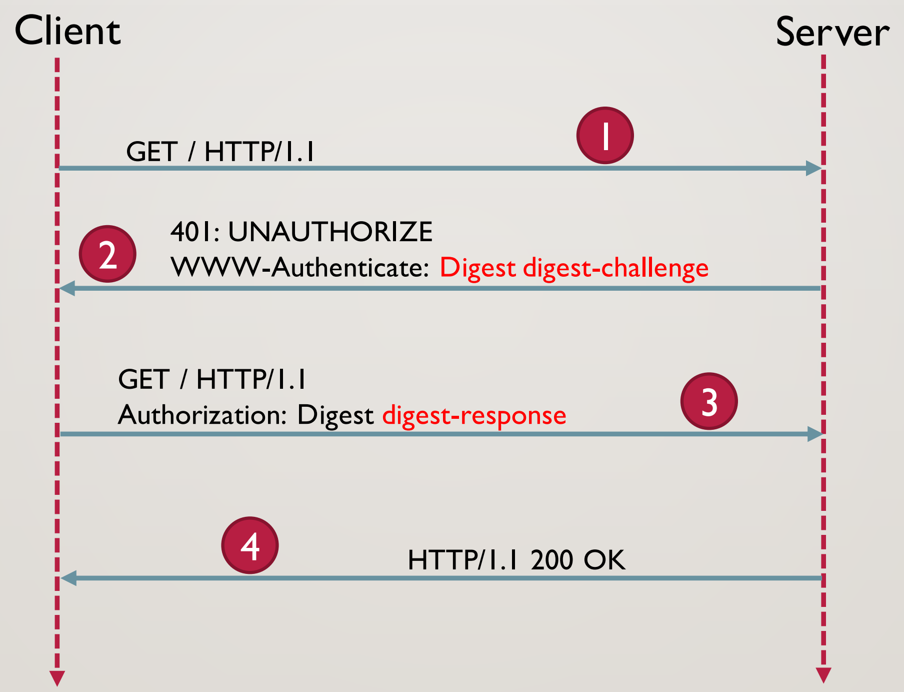
DEMO

Basic Authentication

CONSIDERATIONS

- Cleartext transmission → should use with https.
- Spoofing by counterfeit servers

DIGEST AUTHENTICATION


Client
Server
1
GET / HTTP/1.1
2
401: UNAUTHORIZE
WWW-Authenticate: Digest digest-challenge
GET / HTTP/1.1
Authorization: Digest digest-response
3
4
HTTP/1.1 200 OK

CONSIDERATIONS

- Authenticated transactions interact with shared caches
- Eavesdrop Attacks
- Online dictionary attacks
- Man in the Middle
- Precomputed dictionary attacks
- Batch brute force attacks
- Spoofing by Counterfeit Servers
- Storing passwords

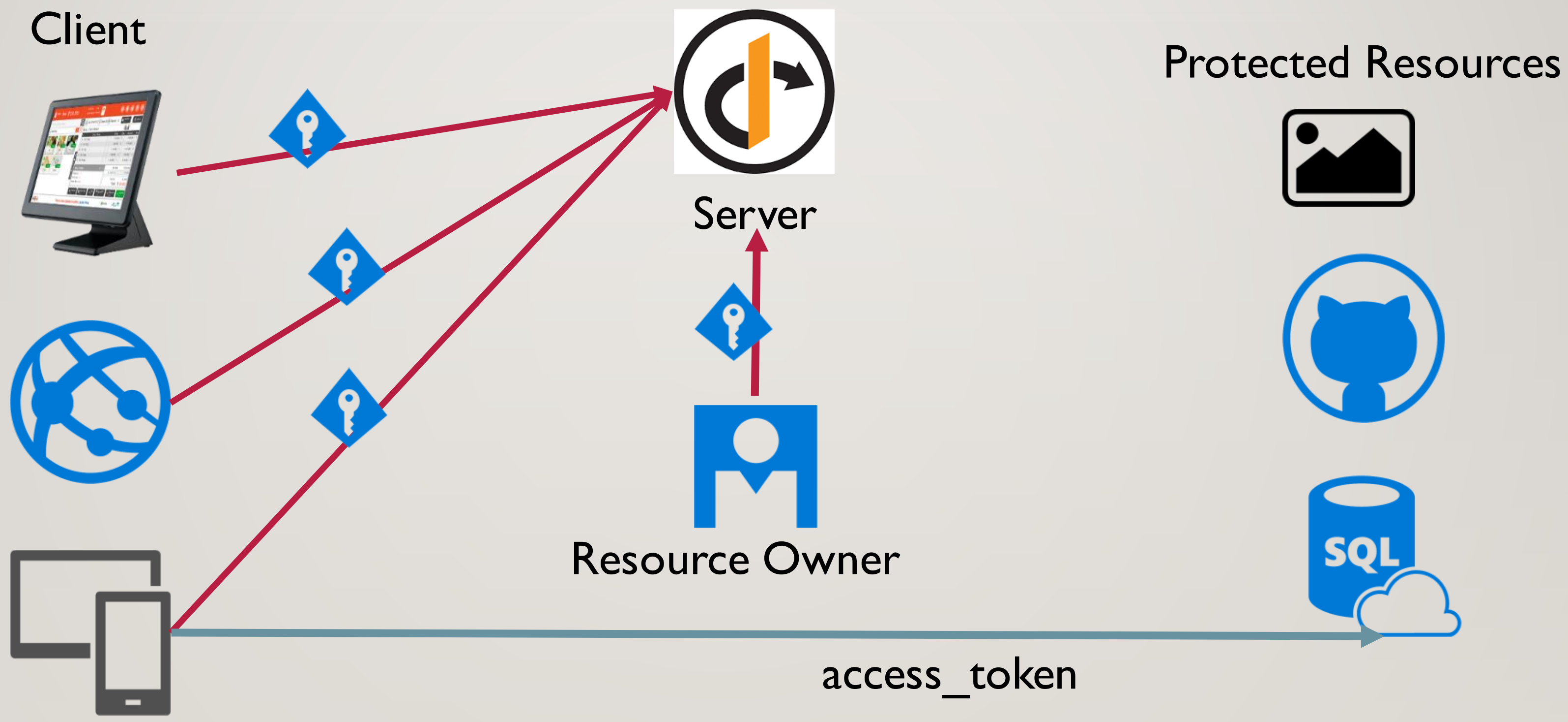
BEARER AUTHENTICATION

- OAuth provides a method for clients to access server resources on behalf of a resource owner
- It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections

TERMINOLOGY

- Client: HTTP client capable of making OAuth-authenticated requests
- Server: An HTTP server capable of accepting OAuth-authenticated requests
- Protected resource: An access-restricted resource that can be obtained from the server using an OAuth-authenticated request
- Resource owner: User
- Credentials: Credentials are a pair of a unique identifier and a matching shared secret.
- Token: A unique identifier issued by the server and used by the client to associate authenticated requests

FACTORS

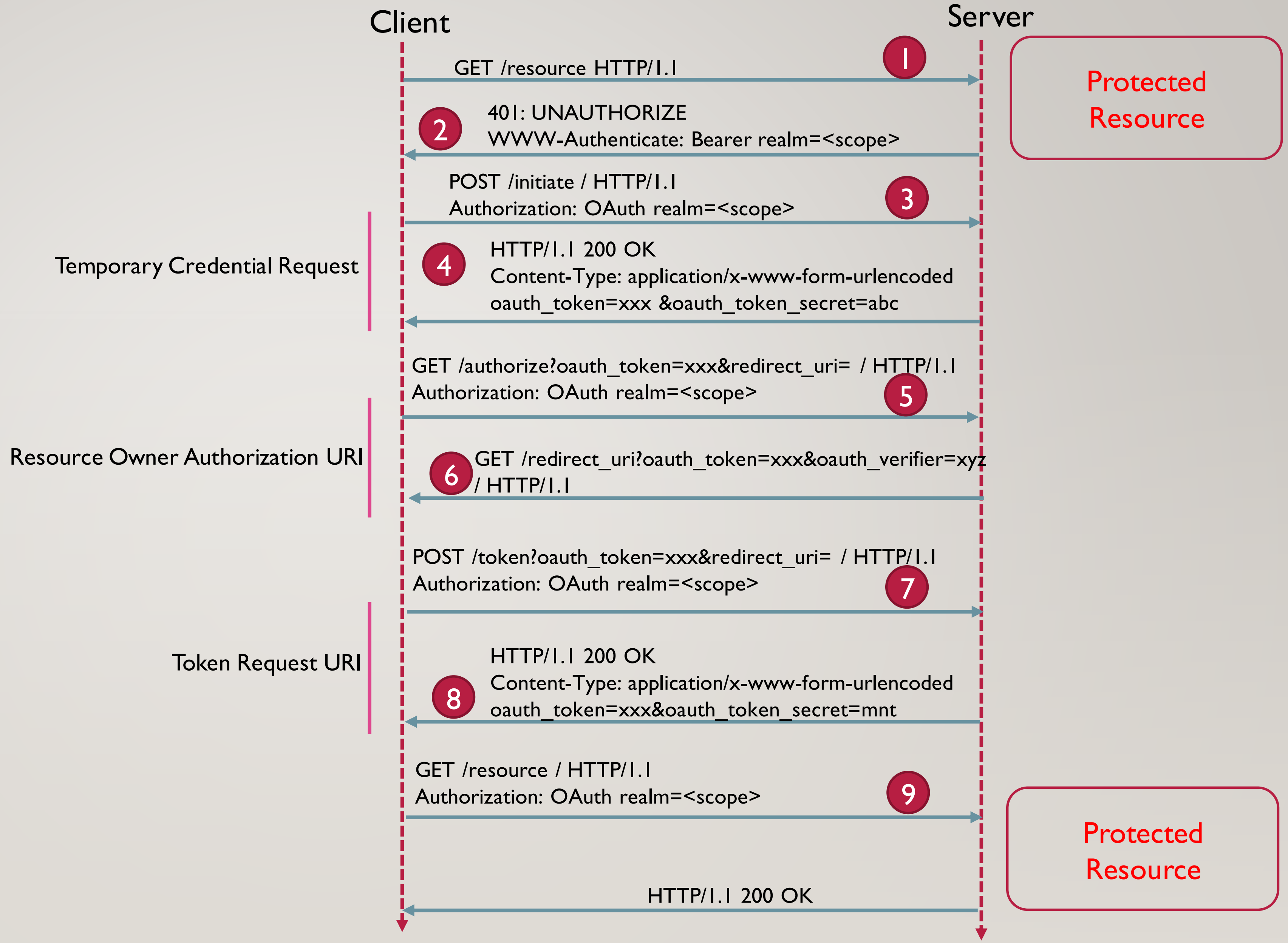

Client
Server
Protected Resources
Resource Owner
SQL
access_token

BEARER AUTHENTICATION

OAuth 1.0

OAuth 2.0

OAUTH 1.0



Client
Server
1
GET /resource HTTP/1.1
Protected Resource
2
401: UNAUTHORIZE
WWW-Authenticate: Bearer realm=<scope>
Temporary Credential Request
3
POST /initiate / HTTP/1.1
Authorization: OAuth realm=<scope>
4
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=xxx &oauth_token_secret=abc
Resource Owner Authorization URI
5
GET /authorize?oauth_token=xxx&redirect_uri= / HTTP/1.1
Authorization: OAuth realm=<scope>
6
GET /redirect_uri?oauth_token=xxx&oauth_verifier=xyz / HTTP/1.1
Token Request URI
7
POST /token?oauth_token=xxx&redirect_uri= / HTTP/1.1
Authorization: OAuth realm=<scope>
8
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=xxx&oauth_token_secret=mnt
9
GET /resource / HTTP/1.1
Authorization: OAuth realm=<scope>
Protected Resource
HTTP/1.1 200 OK

OAuth 2.0

BEFORE WE START

- The client had to registers with the authorization server
- When registering a client:
  - specify the client type
  - provide its client redirection URIs
  - include any other information required by the authorization server (e.g., application name, website, description, logo image, the acceptance of legal terms).

CLIENT TYPE

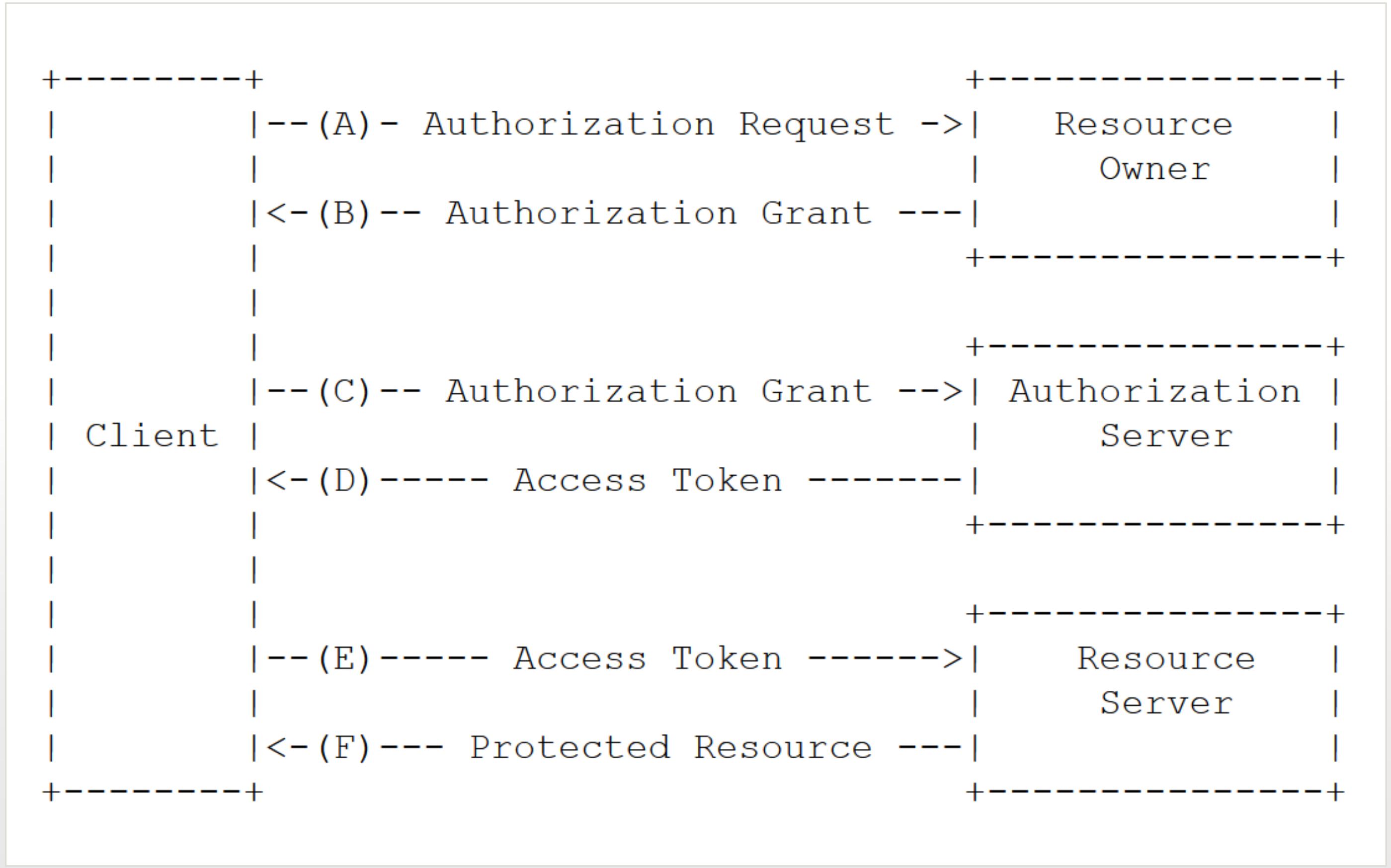
Confidential: Clients capable of maintaining the confidentiality of their credentials or capable of secure client authentication using other means. (web application)

Public: Clients incapable of maintaining the confidentiality of their credentials, and incapable of secure client authentication via any other means. (user-agent-based, native application)

RECEIVE CLIENT REGISTRATION

- Client Identifier (client_id)
- Client Authentication (client_secret)

OAUTH 2.0



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

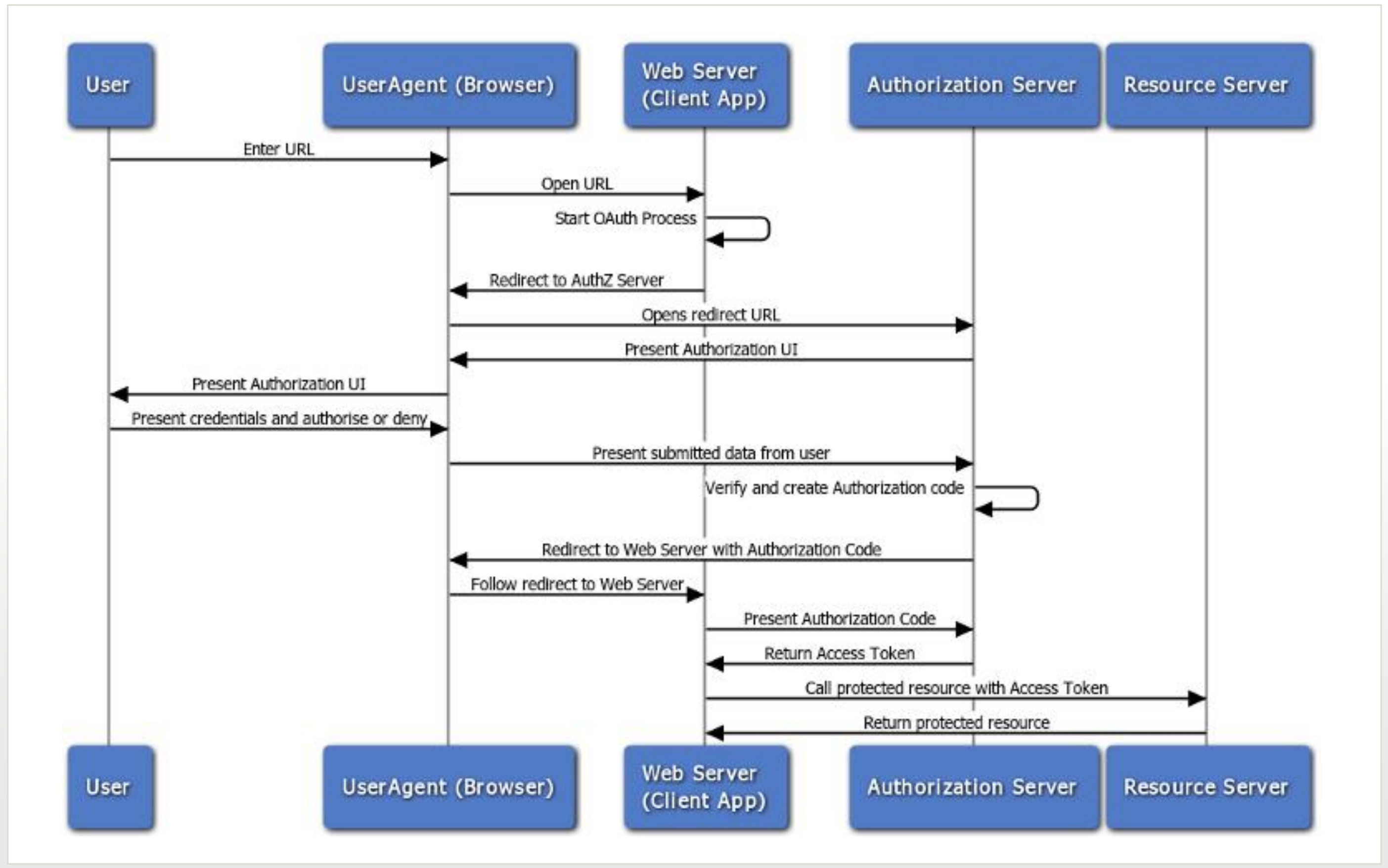
OAUTH 2.0 FLOW

- A: The client requests authorization from the resource owner
- B: Credential representing the resource owner's authorization, expressed using one of four grant types
- C: Requests an access token
- D: Authenticates the client and validates the authorization grant, and if valid, issues an access token
- E: Client using access token to access protected resource.
- F: The resource server validates the access token, and if valid,serves the request

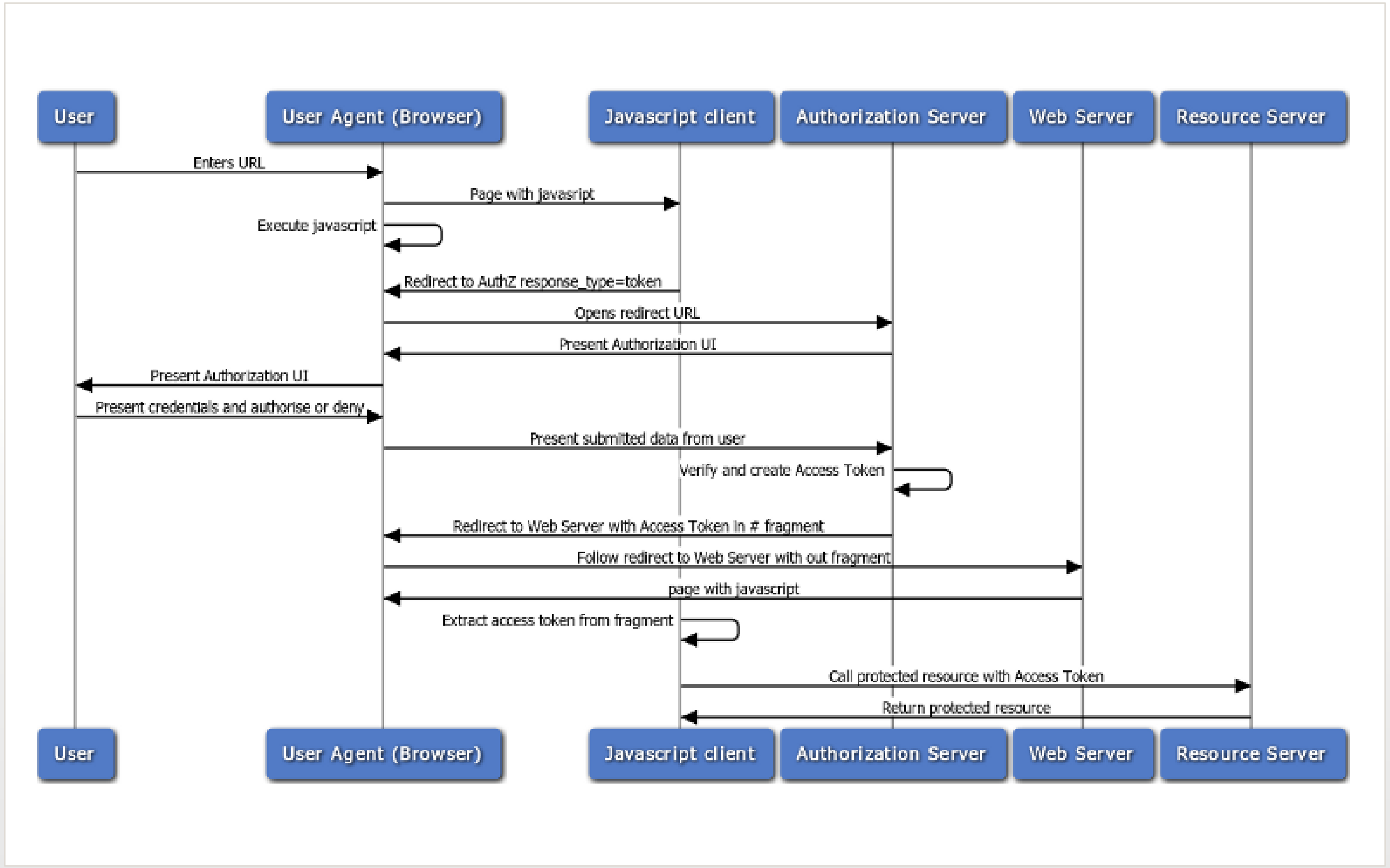
GRANT TYPES

Grant type	Usage
Authorization code	Client directs the resource owner to an authorization server
Implicit	Issued an access token directly
Resource owner password	Use resource owner’s username and password
Client credential	Use when resource has limited scope

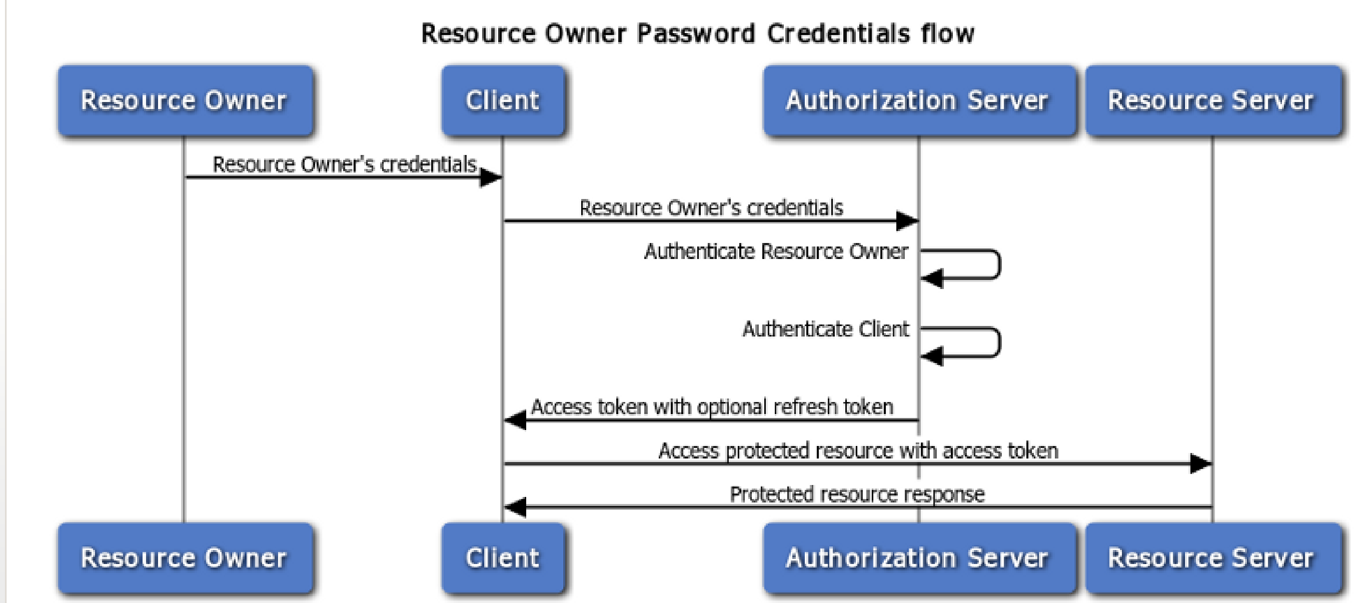
AUTHORIZATION CODE



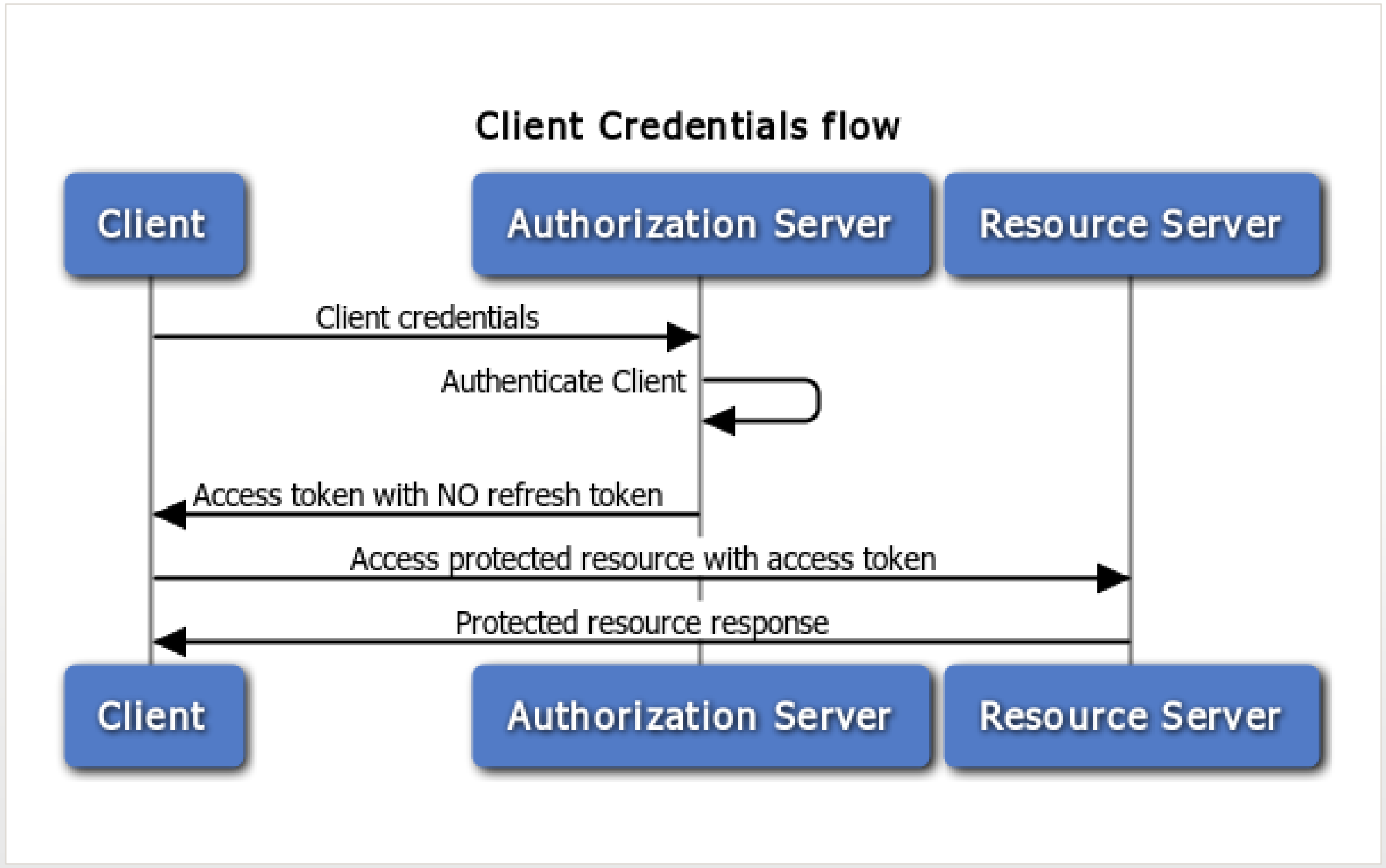
IMPLICIT


User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server
Enters URL
Page with javasript
Execute javascript
Redirect to AuthZ response_type=token
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Access Token
Redirect to Web Server with Access Token in # fragment
Follow redirect to Web Server with out fragment
page with javascript
Extract access token from fragment
Call protected resource with Access Token
Return protected resource

RESOURCE OWNER CREDENTIAL


Resource Owner Password Credentials flow
Resource Owner
Client
Authorization Server
Resource Server
Resource Owner's credentials
Resource Owner's credentials
Authenticate Resource Owner
Authenticate Client
Access token with optional refresh token
Access protected resource with access token
Protected resource response
Resource Owner
Client
Authorization Server
Resource Server

CLIENT CREDENTIAL

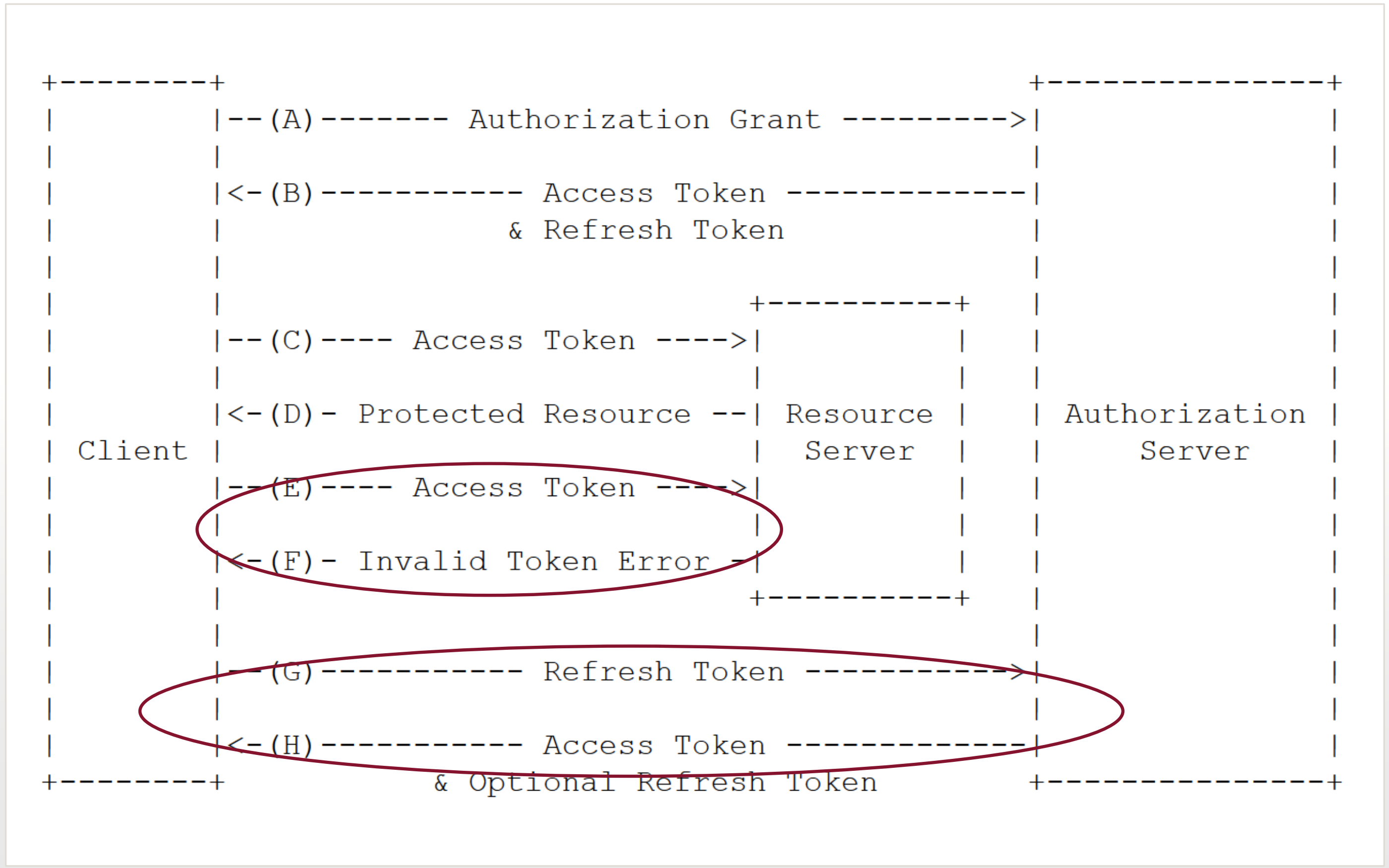


TOKEN

Access Token: credentials used to access protected resources

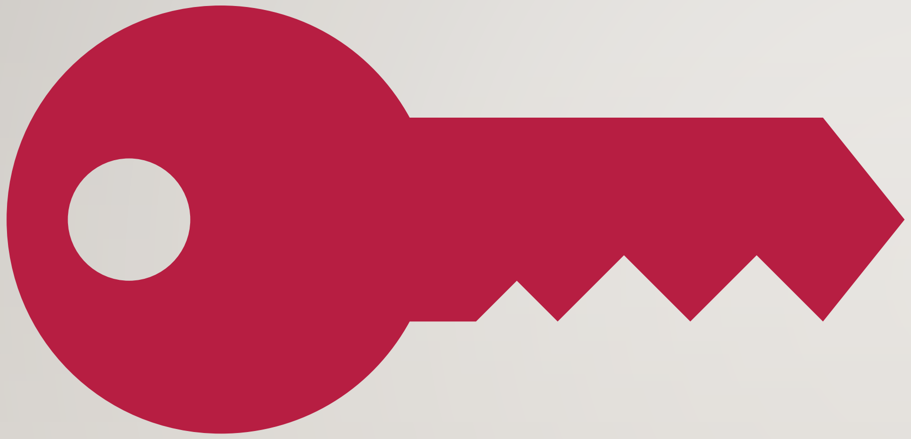
Refresh Token: credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires.

REFRESH TOKEN



```
+--------+                                           +---------------+
|        |--(A)------- Authorization Grant --------->|               |
|        |                                           |               |
|        |<-(B)----------- Access Token -------------|               |
|        |               & Refresh Token             |               |
|        |                                           |               |
|        |                            +----------+   |               |
|        |--(C)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(D)- Protected Resource --| Resource |   | Authorization |
| Client |                            |  Server  |   |     Server    |
|        |--(E)---- Access Token ---->|          |   |               |
|        |                            |          |   |               |
|        |<-(F)- Invalid Token Error -|          |   |               |
|        |                            +----------+   |               |
|        |                                           |               |
|        |--(G)----------- Refresh Token ----------->|               |
|        |                                           |               |
|        |<-(H)----------- Access Token -------------|               |
+--------+           & Optional Refresh Token        +---------------+
```

ENDPOINTS



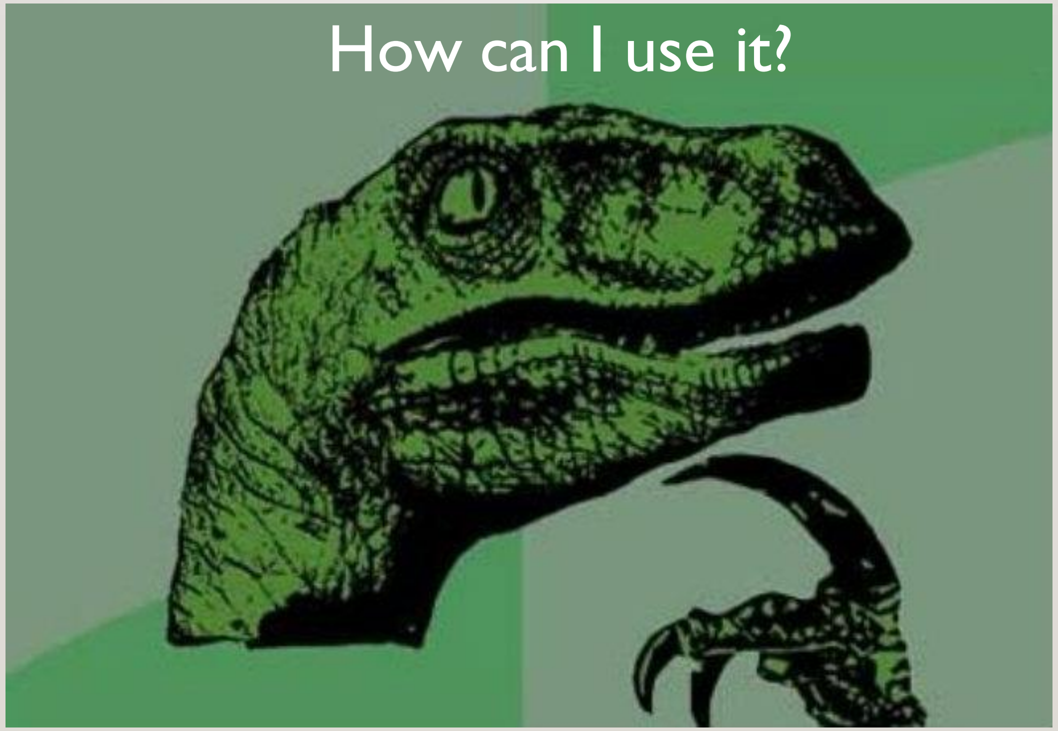
- Authorization endpoint (/authorize)
- Token endpoint (/token)
- Refresh Token endpoint (/refresh)

SUMMARY OAUTH 2.0

- Using bearer authentication scheme
- Define 4 roles: client, authorization server, resource owner and protected resource
- Endpoints: /authorization, /token, /refresh
- Client Credentials: authorization code, implicit, resource owner credentials and client credentials
- Client type: public and confidential

OK, SOUND GOOD

BUT



OPEN ID CONNECT

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol.

OpenID Connect allows clients of all types, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users

ID TOKEN

- The primary extension that OpenID Connect makes to OAuth 2.0 to enable End-Users to be Authenticated is the **ID Token data structure**. The ID Token is a security token that contains **Claims about the Authentication** of an End-User by an Authorization Server when using a Client, and potentially other requested Claims
- The ID Token is represented as a **JSON Web Token (JWT)**

JSON WEB TOKEN (JWT)

- Representing claims to be transferred between two parties.
- Structure:
  - <base64-encoded header>.<base64-encoded payload>.<base64-encoded signature>

  with

  - header: indicating cryptographic operations applied
  - payload: claims with three types such as : reserved, public and private .
  - Signature: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

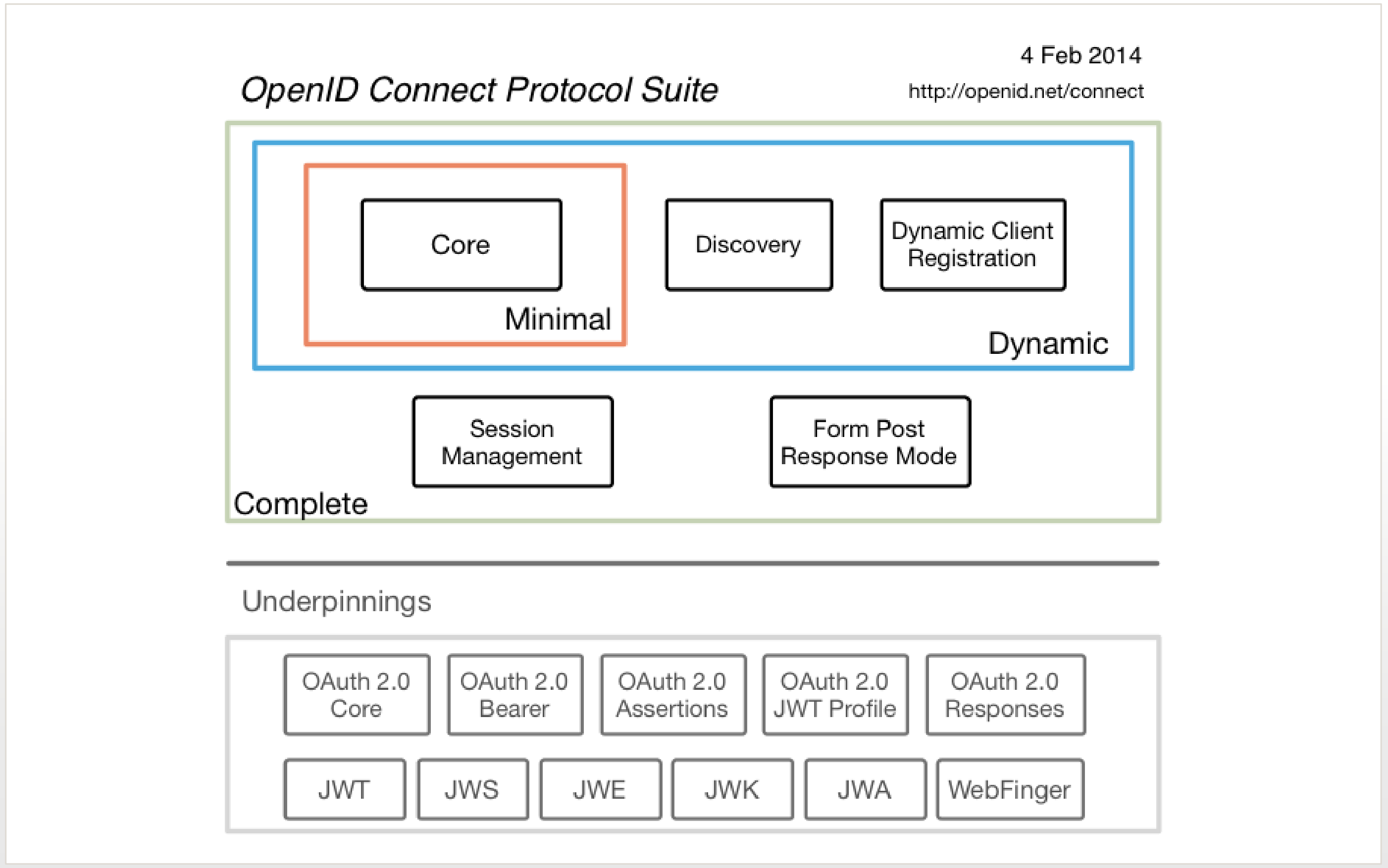
JWT DETAIL

Part	
Header	**alg, typ**
Payload	**iss, sub, aud, exp, iat**, nonce, acr,, amr, azp, and claims
Signature	blackbox

DEMO ID TOKEN

- eyJhbGciOiJSUzI1NiIsImtpZCI6IkQ5OUYzNTkxMkU3QjRBQjE5ODIzRjg5NzhFNTQwRkI3RTVCRTI1O
DkiLCJ0eXAiOiJKV1QiLCJ4NXQiOiIyWjgxa1M1N1NyR1lJX2lYamxRUHQtVy1KWWsifQ.eyJuYmYiOj
E1MjE4NjcyMDksImV4cCI6MTUyMTg3MDgwOSwiaXNzIjoiaHR0cHM6Ly9sb2dpbi13d3cxLmF6dXJld2Vi
c2l0ZXMubmV0IiwiYXVkIjpbImh0dHBzOi8vbG9naW4td3d3MS5henVyZXdlYnNpdGVzLm5ldC9yZXNv
dXJjZXMiLCJjYW1waW5nYm9va2luZ2FwaSJdLCJjbGllbnRfaWQiOiJjYW1waW5nYm9va2luZyIsInNjb3B
lIjpbInNjb3BlLmZpbmFuY2UiXX0.xBpf0X5xLeY5jC3vstXyTOW_tnBUTkM21J38LVLc3zthXaSCc8pfexc
odQQokFr65f-
0r4oi41tcdb5mTcwlhS4DaGZ0I12jZPHTocnGy5I_64JwkuPO4h7dx1jcpdfvc0tf8dgFZjHorVb83lIWIGiUy
BsYdBYUzgKQq5g8BEO76Y4Zzle8tZ0_A-
ypQo4oOJh97UGsG0SfZvBjnhBlDuo70zkkuEDcn_FkhpD7gX89zxZkI8vrb4CDErTh5bVSzhCHljsliYL2vq
IBKIuWnxNwzeVmcCI3afZf3qy4y4n0X6D4fZsY39mwNJNMZTuSQxmAhQRc_9oaqVd_r_Q3qw

OPENID CONNECT ARCHITECTURE



RESPONSE TYPE

code token	response MUST include an Access Token, an Access Token Type, and an Authorization Code.
code id_token	response MUST include both an Authorization Code and an id_token.
id_token token	response MUST include an Access Token, an Access Token Type, and an id_token.
code id_token token	response MUST include an Authorization Code, an id_token, an Access Token, and an Access Token Type.

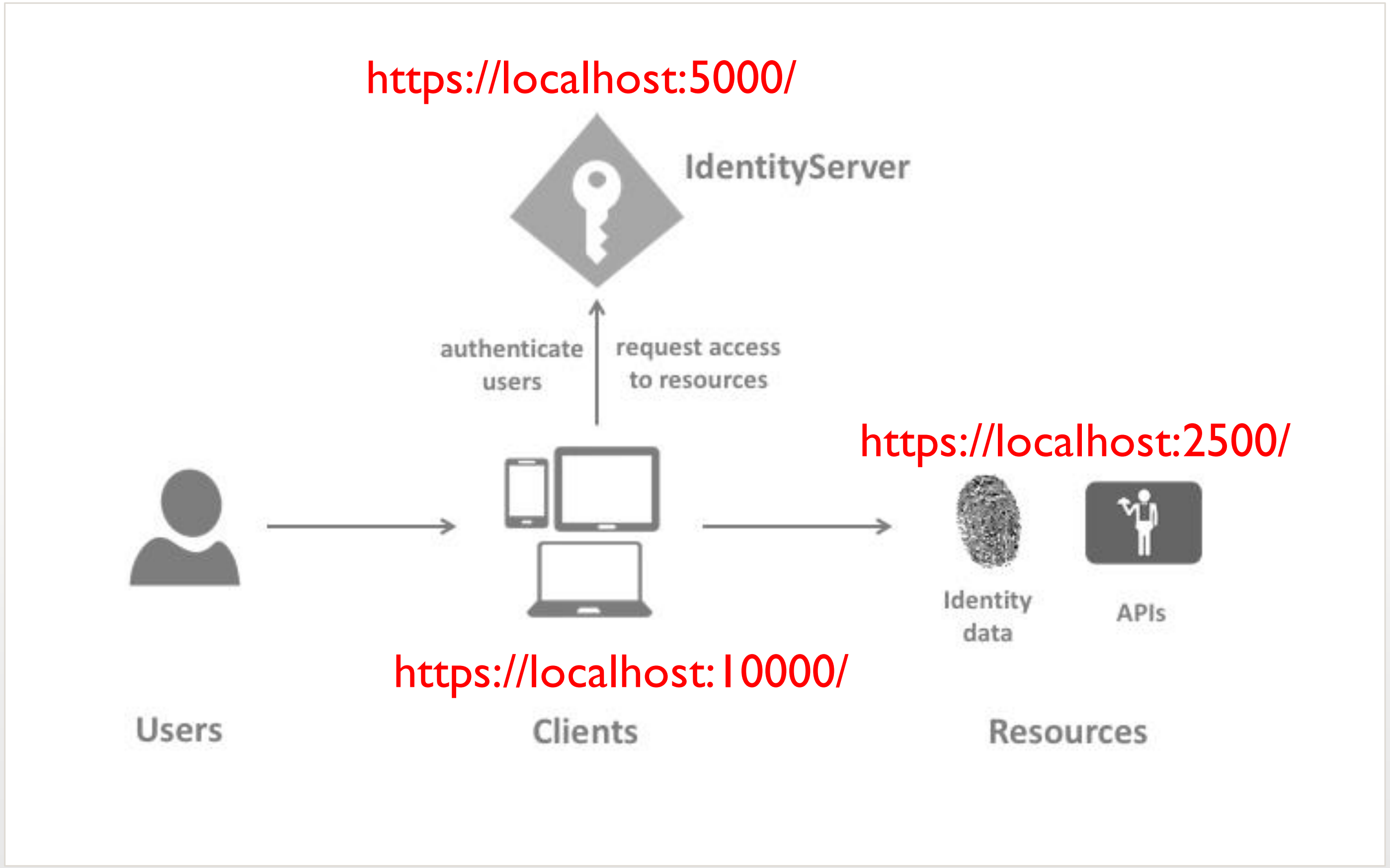
AGENDA (PART 2)

- IdentityServer4
- Azure Active Directory
- Role Base Access Control (RBAC)
- Swagger document with protected api

IDENTITYSERVER4

- OpenID Connect provider - it implements the OpenID Connect and OAuth 2.0 protocols.
- Features:
  - protect your resources
  - authenticate users using a local account store or via an external identity provider
  - provide session management and single sign-on
  - manage and authenticate clients
  - issue identity and access tokens to clients
  - validate tokens

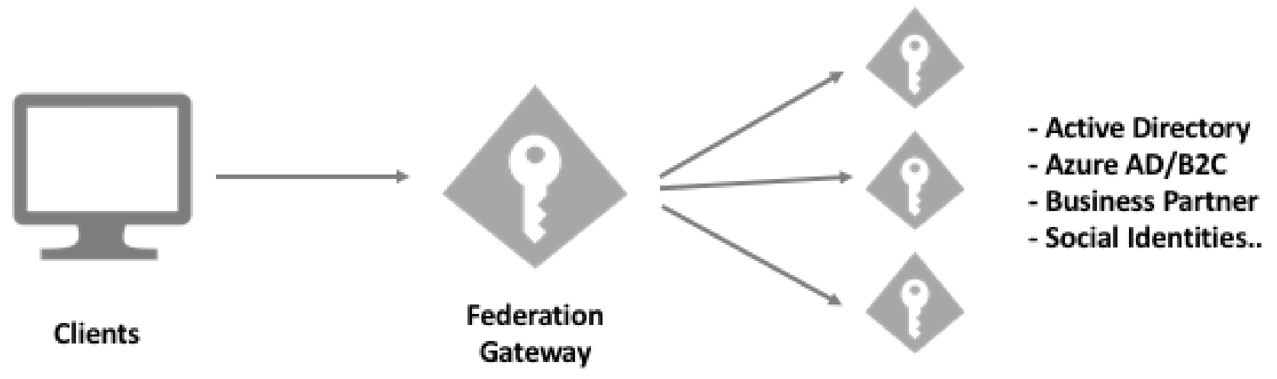
DEMO


https://localhost:5000/
IdentityServer
authenticate users
request access to resources
https://localhost:2500/
Identity data
APIs
https://localhost:10000/
Users
Clients
Resources

DEMO FLOW

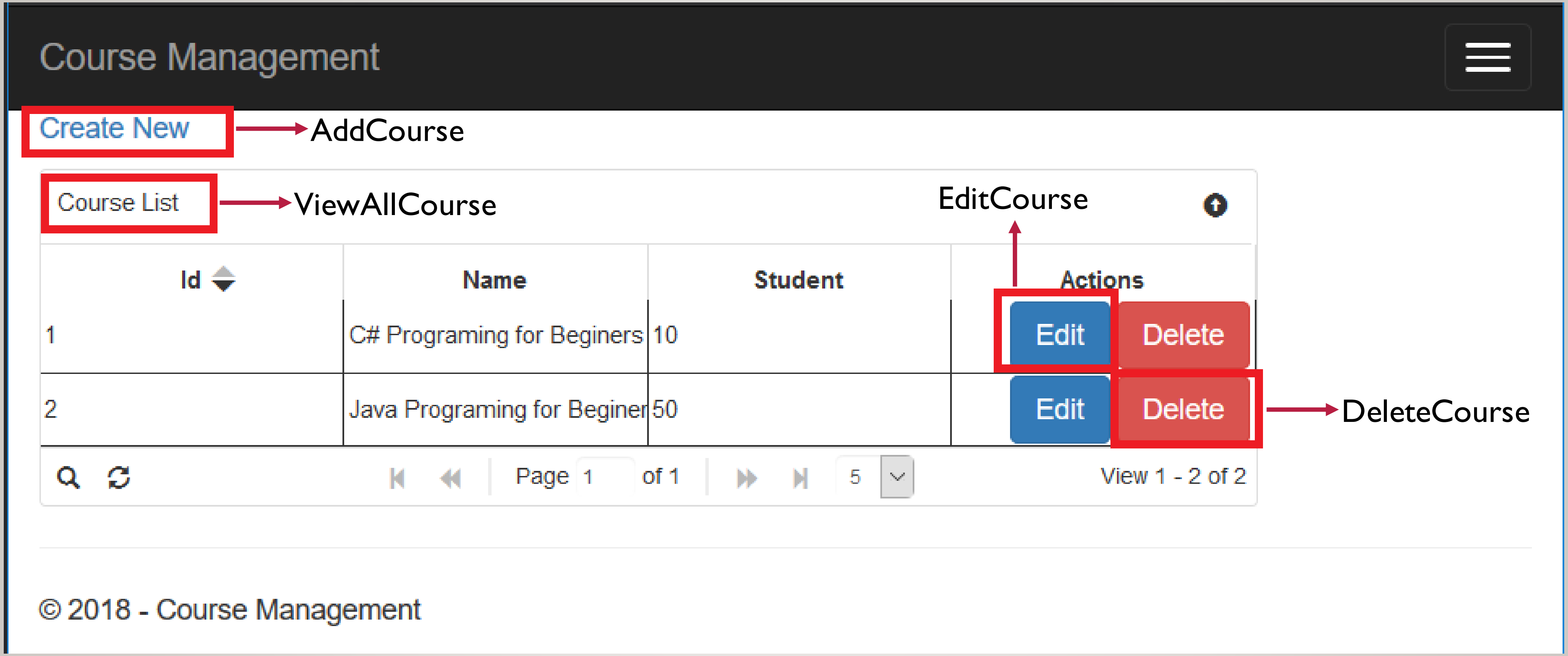
- Client credentials
- Resource owner password
- Implicit
- Authorization Code
- Hybrid

EXTERNAL LOGIN



AZURE AD

- Create Azure AD
  - Create new client
  - Setting Callback (Reply) Url
  - Generate client credentials
- Add Azure AD client to ID4
  - Get authority url
  - Set client credentials



Course Management

Create New

Course List

Id	Name	Student	Actions
1	C# Programing for Beginers	10	Edit Delete
2	Java Programing for Beginer	50	Edit Delete

Page 1 of 1 5 View 1 - 2 of 2

© 2018 - Course Management

ROLE & RIGHT

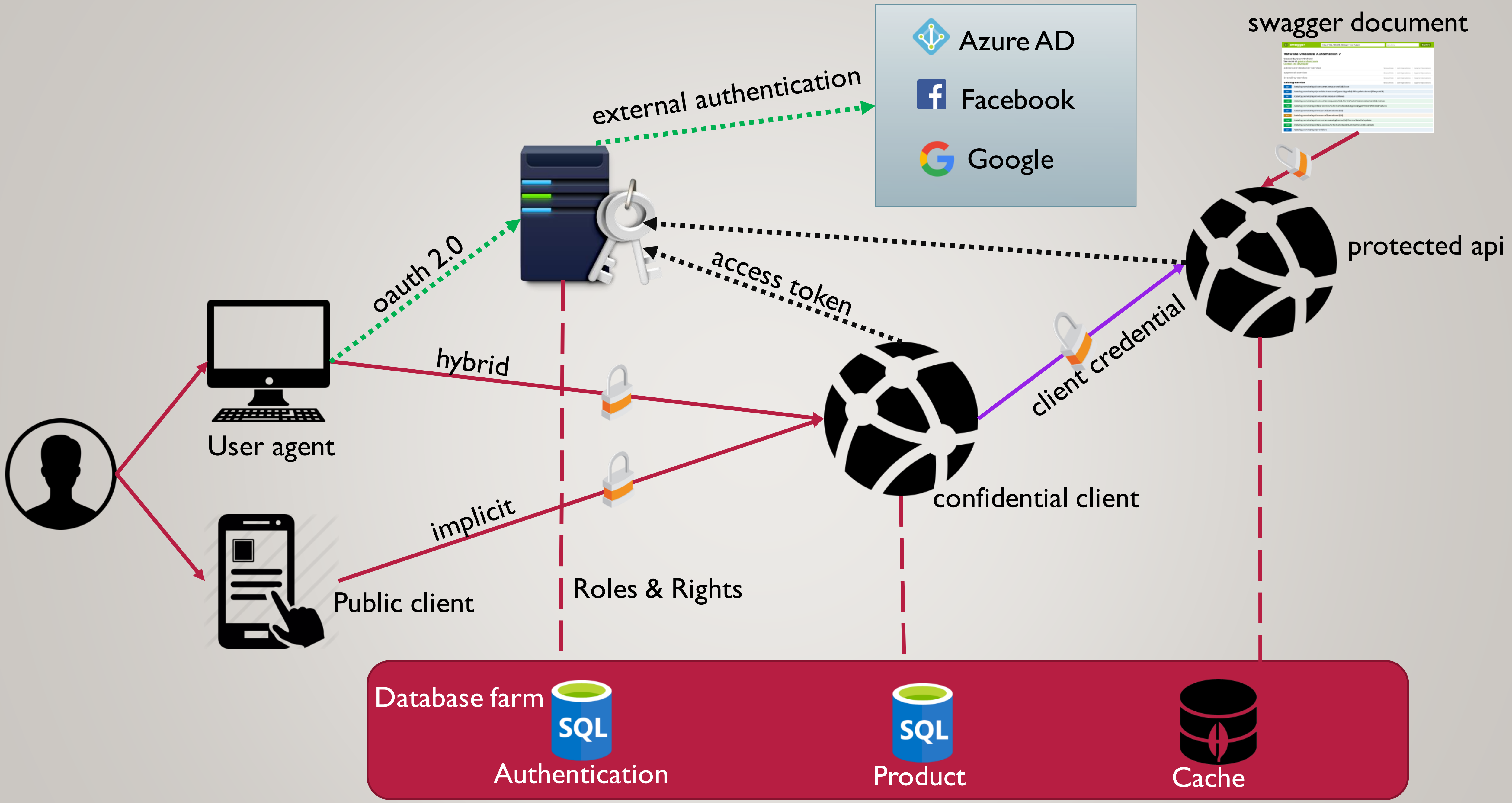
Course Management Configuration				
Role\Right	View all Course	AddCourse	EditCourse	DeleteCourse
Admin	x	x	x	x
Manager	x		x	
Visitor	x			

COURSE MANAGEMENT SYSTEM

- Decorate controller with authorize attribute, include Role configuration.
- Show/Hide feature base on Right name.

DEMO

AUTHORIZE IN SWAGGER


swagger document
Azure AD
Facebook
Google
external authentication
oauth 2.0
access token
protected api
hybrid
client credential
User agent
implicit
confidential client
Public client
Roles & Rights
Database farm
SQL
Authentication
SQL
Product
Cache

ORTHERS

Microsoft Identity

Oauth.net

REFERENCES

- https://tools.ietf.org/html/rfc2617
- https://tools.ietf.org/html/rfc7616
- https://tools.ietf.org/html/rfc7617
- https://tools.ietf.org/html/rfc6749
- https://tools.ietf.org/html/rfc5849
- https://tools.ietf.org/html/rfc6750
- https://simple.wikipedia.org/wiki/Chosen-plaintext_attack
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

REFERENCES

- https://tools.ietf.org/html/rfc7519
- https://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/part_oauth.html
- http://openid.net/specs/openid-connect-core-1_0.html