Authentication & Authorization

auth: (userName, password) ⇒ Option<UserId>

whether we can recognize that user

authorize: (UserId) ⇒ List<Permission>

context ≡ connection to the DB and access to table

auth: (userName, password, context) => Option<UserId>

```
AuthService {
  AuthRepo repo;

  auth: repo.findByLogin(userName) == password
}
```

→ Option.empty

→ Option.of(user)

where to set this combination

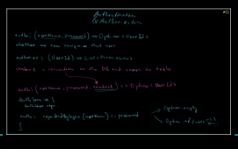
- command line
- function parameter
- part of Cookie
- part of HttpHeader
- part of the Body Request

⋮

what about storing password.

- password stored in a hashed way

plain password ⇒ encode it ⇒ compare encoded



Authorization

Home

based on user Id we can provide custom page for each user.

=> whether user is allowed to do some operation (visit page, create resource, make payment)

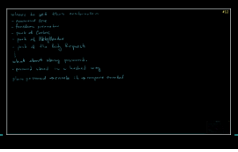
user : List<Permission> → user : list.empty

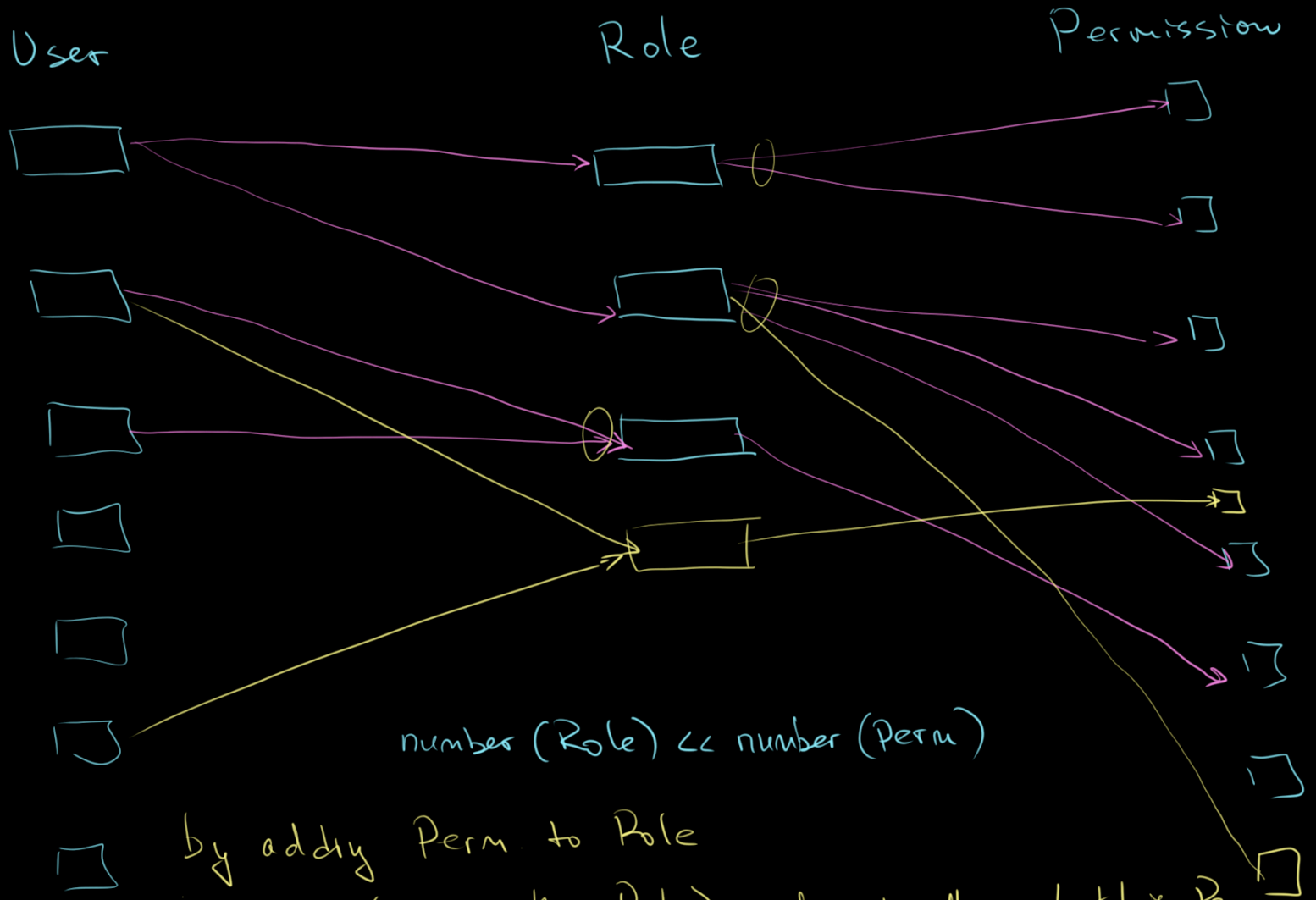
Map<UserId, List<Permission>>

these permissions are similar

=> we group similar permission List and call them properly => ROLE

USER, ADMIN, MANAGER, ACCOUNTANT .....




User
Role
Permission
number (Role) << number (Perm)
by addig Perm to Role
all users (having this Role) automatically get this Perm.

Role Based Authentication

Role Based Access Control RBAC

auth(user, password) → Option<UserId>

authoriz.(UserId) → List<Role>

Role → List<Permission>

```
auth(user, passw)
  .map(u -> authorize(u))
```

// Option<UserId>

// Option<List<Role>>

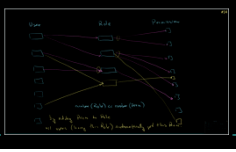
RBAC

auth ← login + password
token
identity provider
Google, Facebook, GitHub OAuth

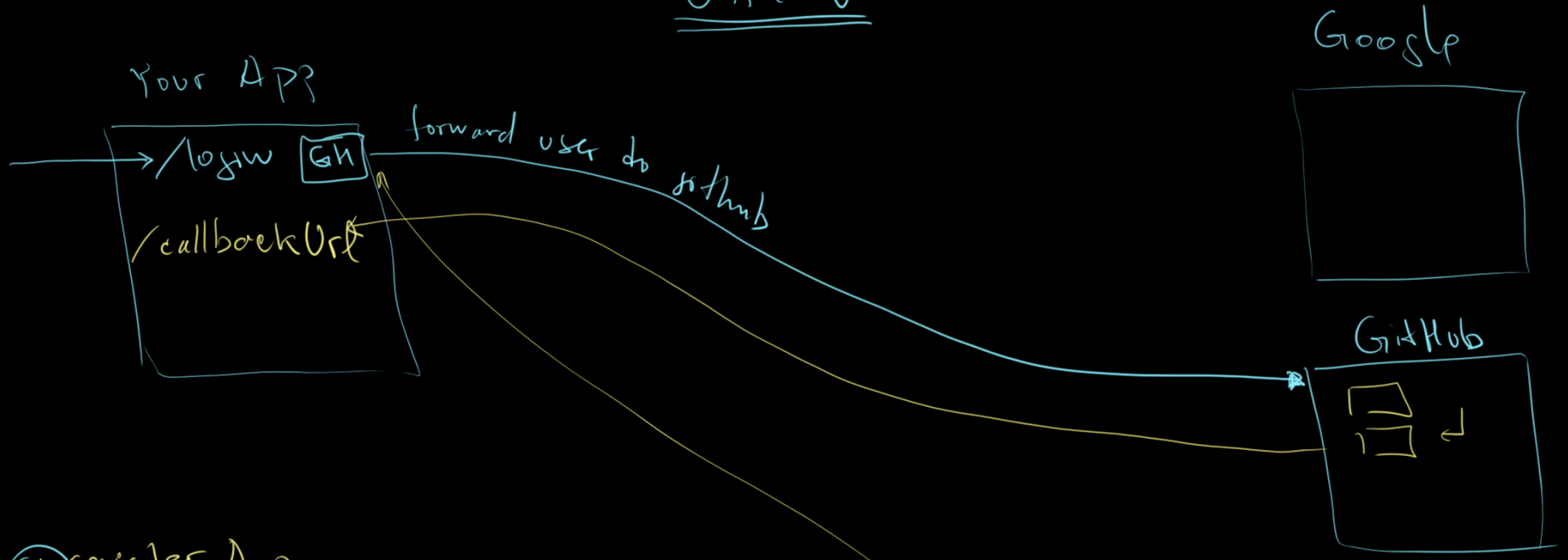
id | error

auth → can be delegated

authoriz. → can not
nobody is aware
of your structure
your roles
your details



OAuth



1. register App
   - provide callback Url
   - set auth Url

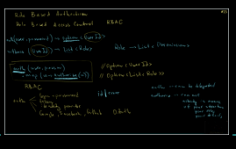
2.a. login: forward user to auth Url

2.b. GitHub handles everything

2.c. as a result GitHub forwards to

① your App Needs to be registered and you will get unique link https://github/auth...

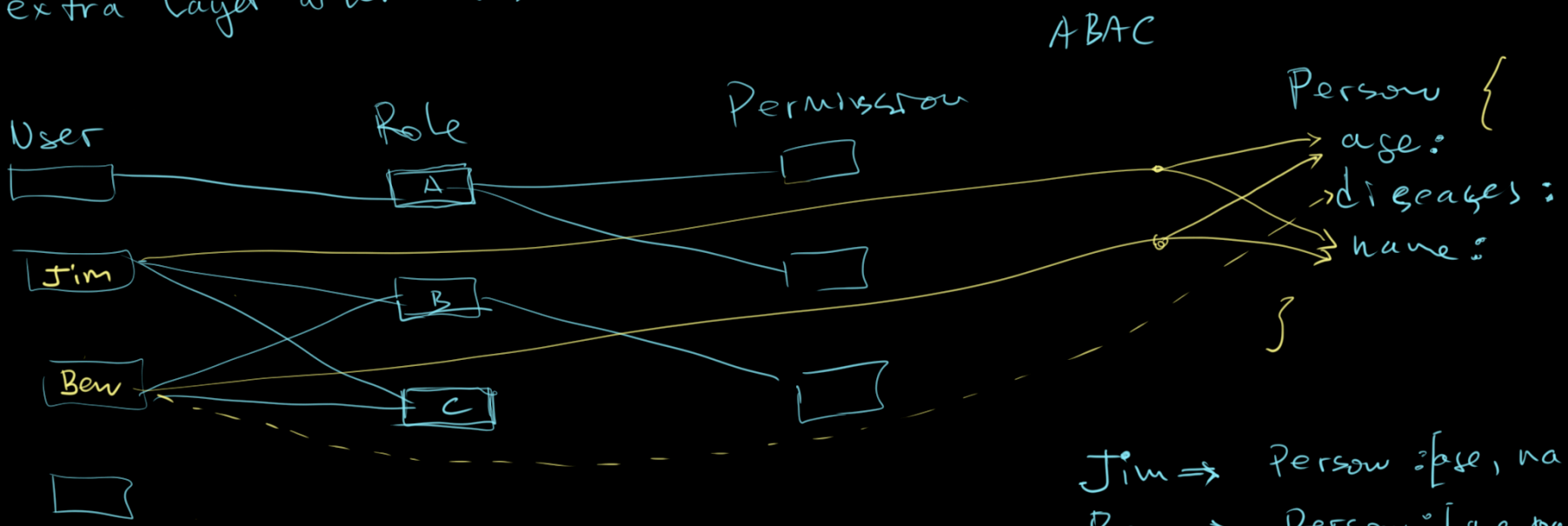
② you need to provide callBackUrl



Attribute Based Authorization (ABAC)

GDPR

=> extra layer after roles.



Jim ⇒ Person: [age, name]

Ben ⇒ Person: [age, name, diseases]

Full Access = Role Based + Attr. Based

erases user details

brings user details

"default strategies"

visible to everyb

hidden from everyb

it's done through Reflection ~ Serialization | Deserialization

serialize : Object → String Person → String

deserialize : String → Object | Error String => Person ? NO

String => Either [Person, Error]

deserialize : String ⇒ Either [Error, Object]

we provide custom serializer

Object → String

(Object, User, Context) → String

@JsonIgnore

@JsonView("")

(User) → Context

(Object, Context) → String.

having clear explicit serializer — will make it more maintainable