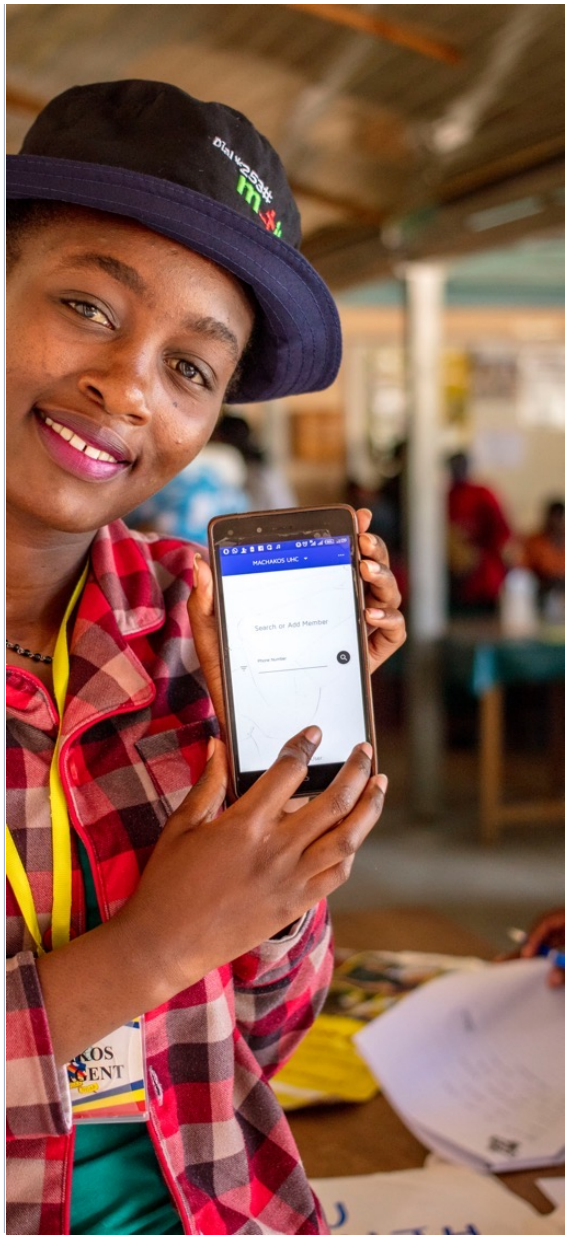
Spring Security goes to Zanzibar

Barry Lagerweij

Table of Contents

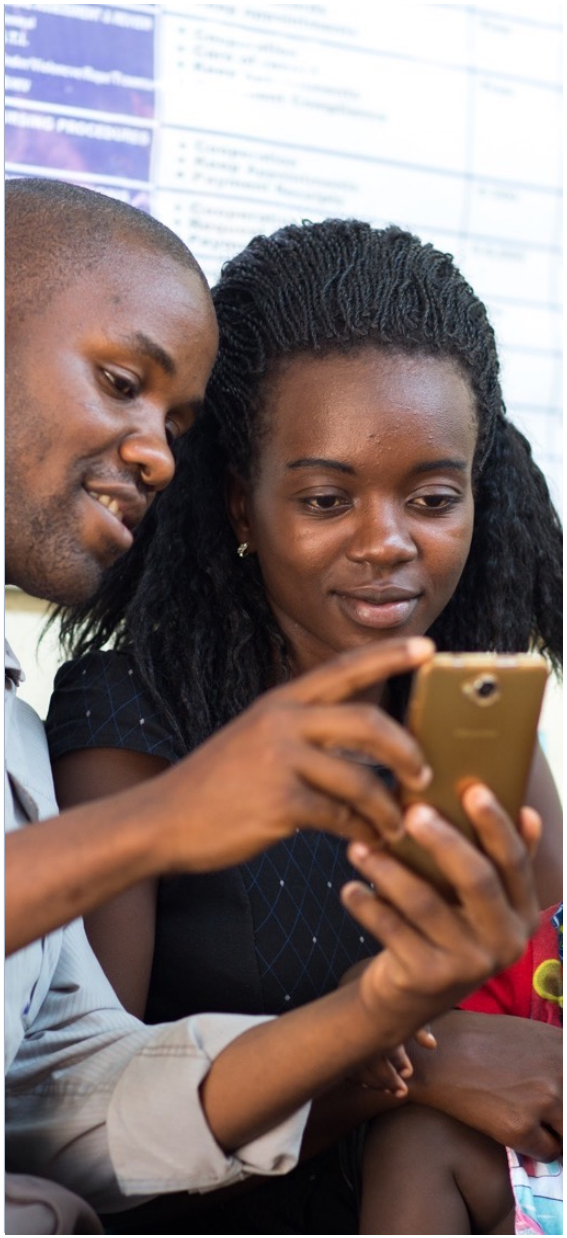
carepay



Why data access security: A01 : Broken Access Control

- Bypass access control checks
- Unauthorized access to accounts
- Unauthorized creation, reading, updating and deletion of data
- Elevation of privilege

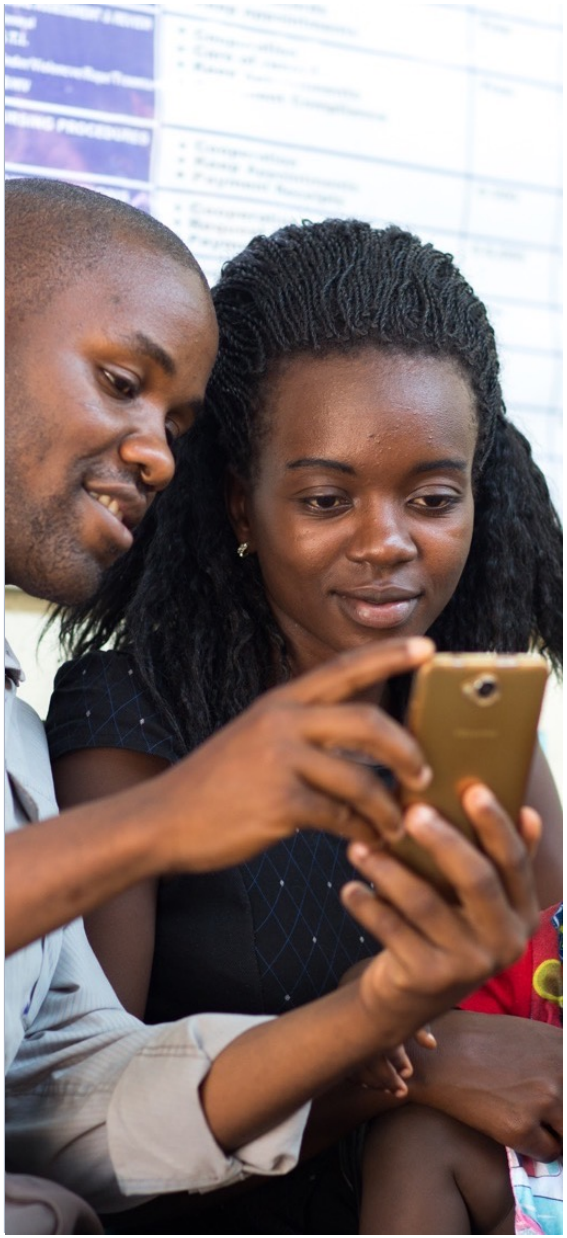
- Privacy and regulatory impacts
- The biggest breaches and largest costs



Evolution of authorization models

1. Role based
2. Backend checks for data access
3. Shared library
4. Network service

carepay



Requirements

1. Many concurrent users
2. Many requests per second
3. Many objects to protect
4. Low latency
5. Reliable
6. Global

carepay

Zanzibar Paper

Zanzibar: Google's Consistent, Global Authorization System

Ruoming Pang,[1] Ramón Cáceres,[1] Mike Burrows,[1] Zhifeng Chen,[1] Pratik Dave,[1]
Nathan Germer,[1] Alexander Golynski,[1] Kevin Graney,[1] Nina Kang,[1] Lea Kissner,[2]*
Jeffrey L. Korn,[1] Abhishek Parmar,[3]* Christina D. Richards,[1] Mengzhi Wang[1]
Google, LLC;[1] Humu, Inc.;[2] Carbon, Inc.[3]
{rpang,caceres}@google.com

Abstract

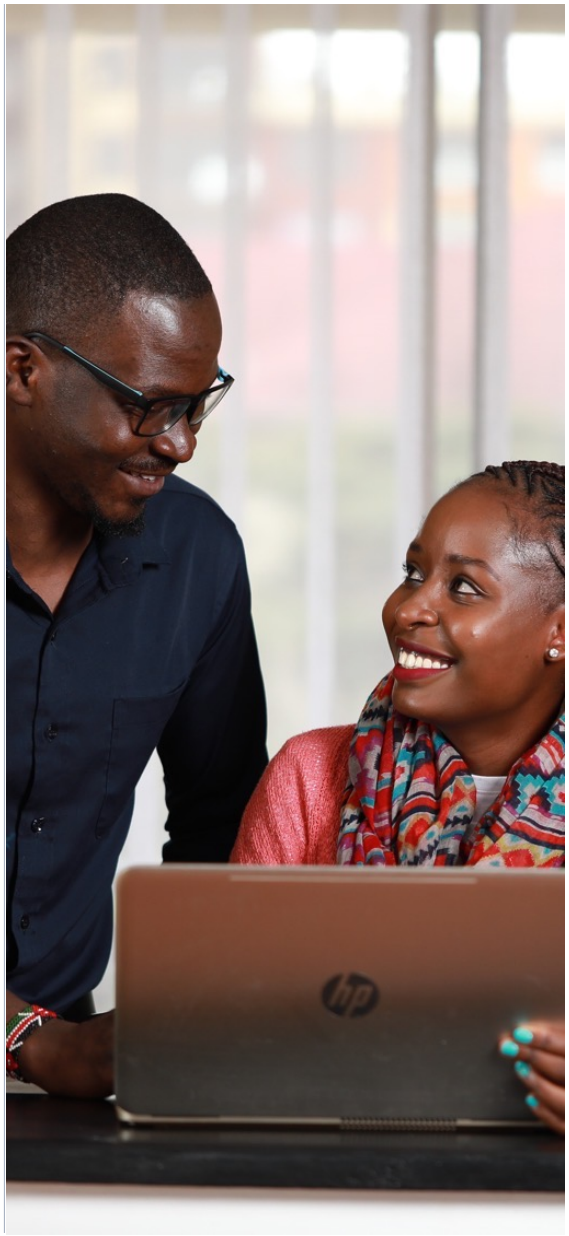
Determining whether online users are authorized to access digital objects is central to preserving privacy. This paper presents the design, implementation, and deployment of Zanzibar, a global system for storing and evaluating access control lists. Zanzibar provides a uniform data model and configuration language for expressing a wide range of access control policies from hundreds of client services at Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. Its authorization decisions respect causal ordering of user actions and thus provide external consistency amid changes to access control lists and object contents. Zanzibar scales to trillions of access control lists and millions of authorization requests per second to support services used by billions of people. It has maintained 95th-percentile latency of less than 10 milliseconds and availability of greater than 99.999% over 3 years of production use.

1 Introduction

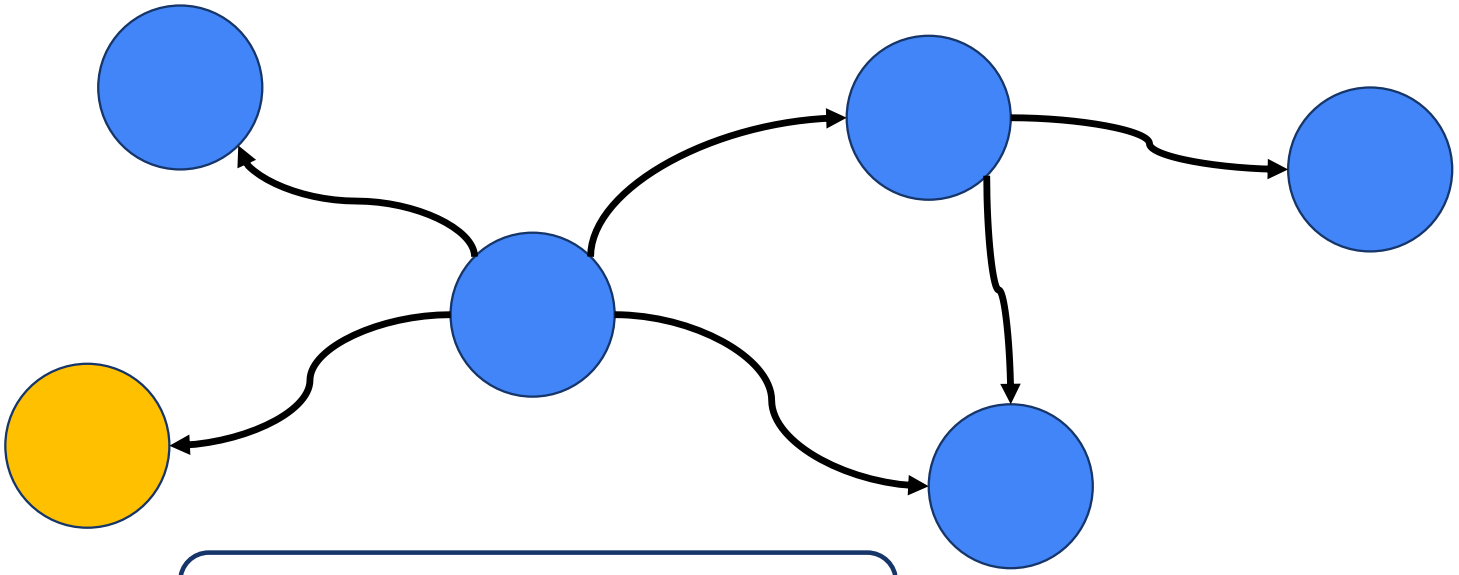
Many online interactions require authorization checks to confirm that a user has permission to carry out an operation semantics and user experience across applications. Second, it makes it easier for applications to interoperate, for example, to coordinate access control when an object from one application embeds an object from another application. Third, useful common infrastructure can be built on top of a unified access control system, in particular, a search index that respects access control and works across applications. Finally, as we show below, authorization poses unique challenges involving data consistency and scalability. It saves engineering resources to tackle them once across applications.

We have the following goals for the Zanzibar system:

- *Correctness*: It must ensure consistency of access control decisions to respect user intentions.
- *Flexibility*: It must support a rich set of access control policies as required by both consumer and enterprise applications.
- *Low latency*: It must respond quickly because authorization checks are often in the critical path of user interactions. Low latency at the tail is particularly important for serving search results, which often require tens to hundreds of checks.



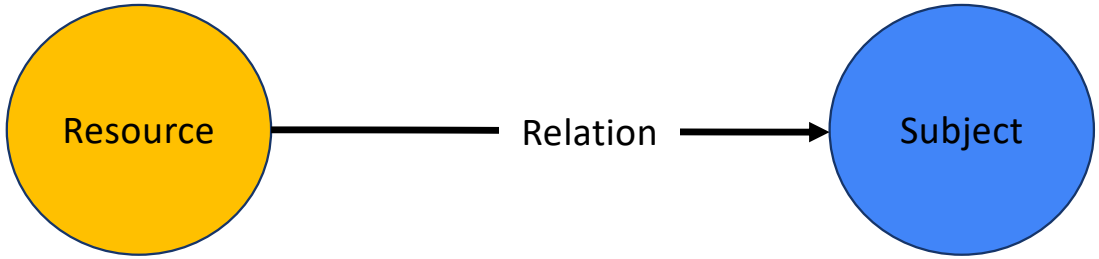
Relation Based Access Control (ReBAC)



Does user Jane have the permission to view document README.DOC ?

carepay

How Relation Based Access works



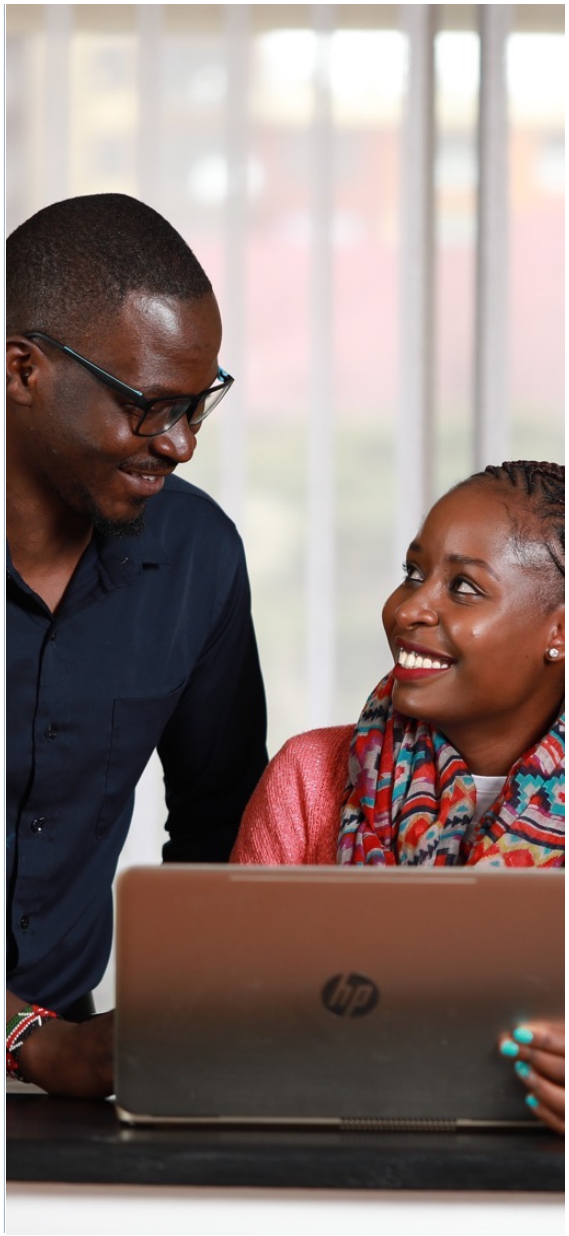
- **Namespace**: type of resource
- **Object**: unique identifier
- **Relation**: how resource is related to subject
- **Subject**: user or another relation tuple

```
namespace:object#relation@subject
```

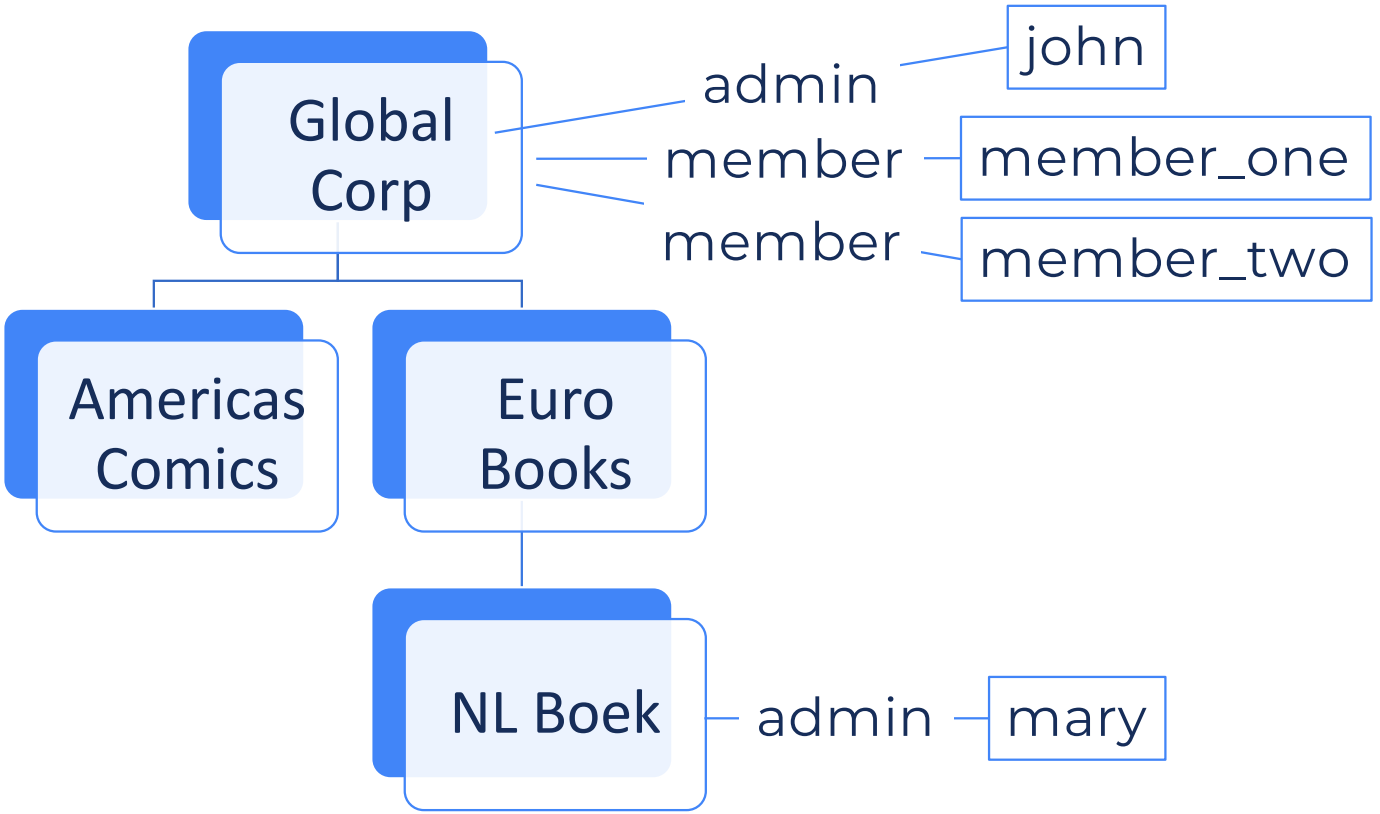
Relation Tuples examples

- `document:readme#owner@user:456`
- User `456` is the `owner` of document `readme`

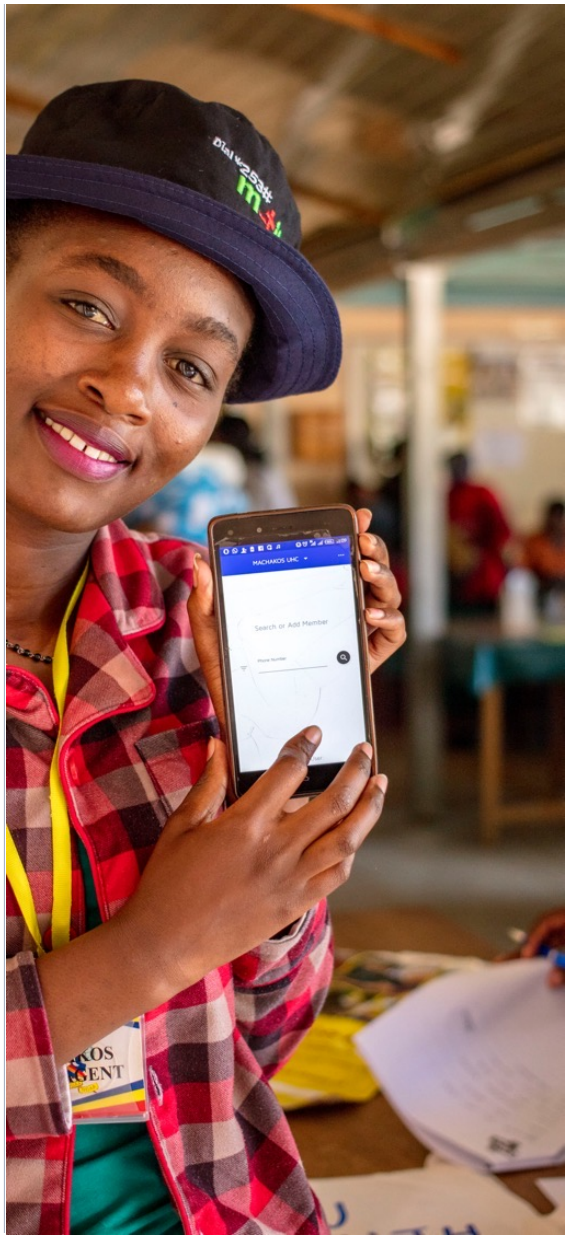
- `group:1#manager@group:leaders#member`
- any `member` of group `leaders` is `manager` for group `1`



Demo: Library with books



carepay



Defining the schema

```
definition user {}

definition library {
  relation member: user
  relation admin: user
  relation parent: library

  permission view =
       member + admin + parent->view


  permission manage =
       admin + parent->manage
}
```


carepay

Spring Security Authorization

1. `@Secured("ROLE_VIEWER")`
2. `@RolesAllowed("ROLE_VIEWER")`
3. `@PreAuthorize("hasRole('ROLE_VIEWER’)”)`

4.
```
@PreAuthorize("#user ==
authentication.principal.username")
String getMyRoles(String user)
{ … }
```

5.
```
@PreAuthorize("hasAnyAuthority('admin') or
hasPermission(#authorId, 'author', 'manage')")
Author updateAuthor(Long authorId,  String name)
```

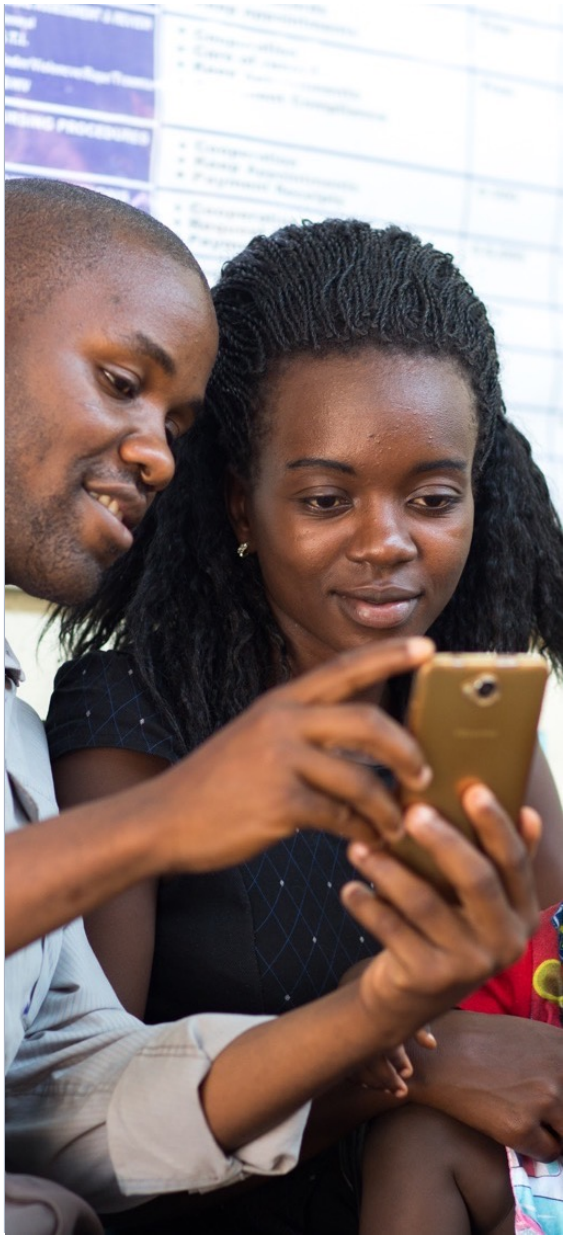

carepay

Spring Security Authorization

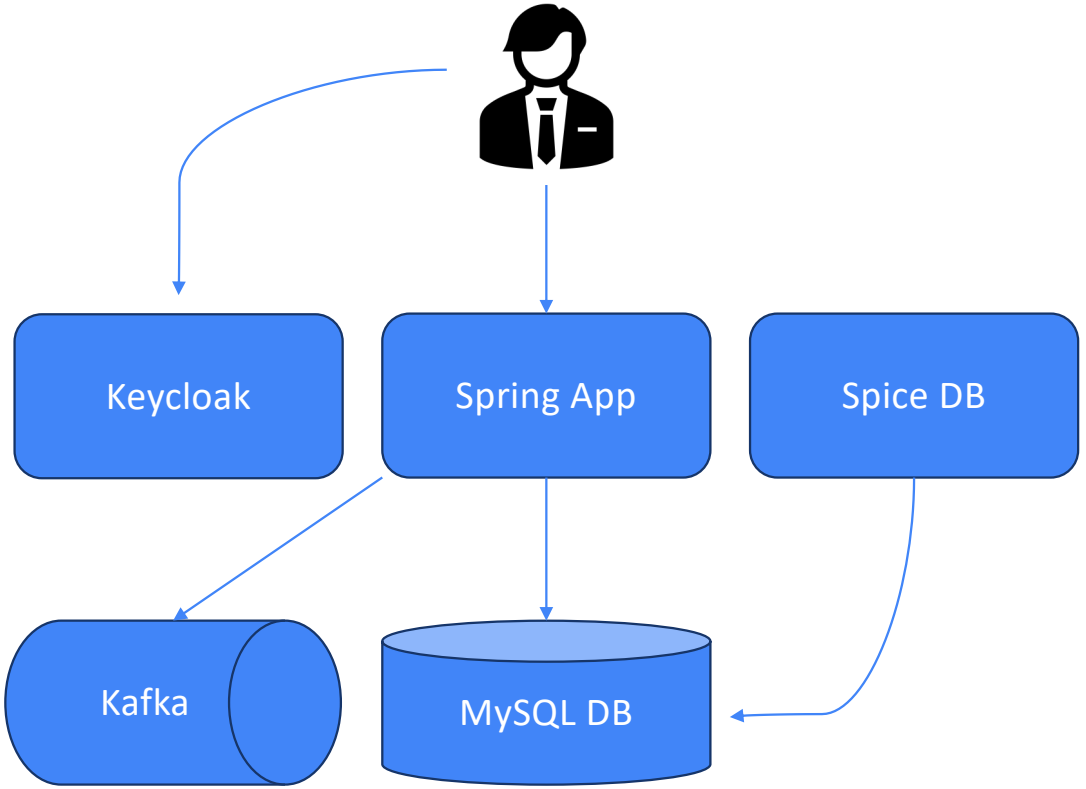
```
public interface PermissionEvaluator {

  boolean hasPermission(
      Authentication authentication,
      Object targetDomainObject,
      Object permission
  );

  boolean hasPermission(
      Authentication authentication,
      Serializable targetId,
      String targetType,
      Object permission
  );
}
```



Demo time!



carepay

Wrap up

- Zanzibar derivatives:
  - Ory Keto
  - SpiceDB
  - Himeji (Airbnb)
  - Permify
  - OpenFGA (Okta)
  - AuthZ (Carta)
- Policy based: OPA and AWS Cedar
- Caveats

- Playground: https://play.authzed.com
- Github: https://github.com/team-carepay


carepay