

A solar powered flux capacitor

by

Felipe Trost

Matriculation Number 0456129

A thesis submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Service-centric Networking

Master's Thesis

May 19, 2024

Supervised by:
Prof. Dr. Axel Küpper

Assistant supervisor:

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, May 19, 2024

Chuck Norris' son

Abstract

In this thesis, we show that lorem ipsum dolor sit amet.

Zusammenfassung

Hier kommt das deutsche Abstract hin. Wie das geht, kann man wie immer auf Wikipedia nachlesen <http://de.wikipedia.org/wiki/Abstract...>

Contents

1	Introduction	1
2	Research Questions	5
3	Task List	7
4	Related Work	9
4.1	OAuth 2.0 and OpenID Connect	9
4.1.1	OAuth 2.0 Roles	9
4.1.2	Authorization Grants	10
4.1.2.1	Implicit	10
4.1.2.2	Resource Owner Password Credentials	10
4.1.2.3	Client Credentials	10
4.1.2.4	Authorization Code	10
4.2	PROCEED's role system	11
5	Concept and Design	13
6	Implementation	17
7	Evaluation	19
8	Conclusion	21
	List of Tables	23
	List of Figures	25
	Appendices	27
	Appendix 1	29

1 Introduction

"I'm awesome!"

—Barney Stinson, *WIRED* magazine, 19.1.2009

Nowadays cloud tools have become very popular in professional settings. Cloud applications are those, which are served and accessed over the internet, they offer many benefits:

- **Accessibility:** they can be accessed anywhere from anywhere with an internet connection.
- **Cost-Efficient:** most cloud applications offer a subscription model, where you have to pay a monthly price, instead of paying for the whole software upfront.
- **Collaboration:** typically, collaboration is easier since all assets are stored in a central place.
- **Data safety:** users don't have to implement their own data storage solutions, which may be vulnerable.
- **Device agnostic:** many cloud applications can be accessed through different device types.
- **No IT overhead:** users don't have to setup the application on their own, which would require technical knowledge.

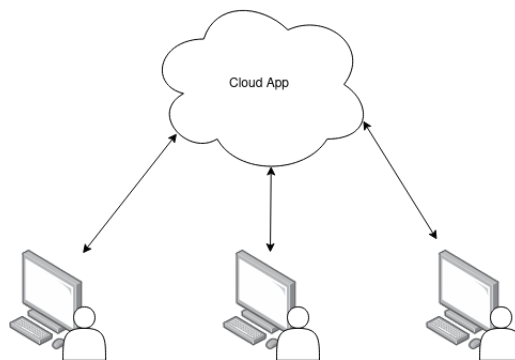


Figure 1.1: Cloud Application.

Most of these benefits only apply if the same instance of the application is able to serve every user. This is called multi-tenancy, and it is implemented by most popular cloud applications, e.g. by Teams, Slack, Asana, and many more. A tenant can be seen as an entity that uses the application, which can be either a single user or an organization made up of many users.

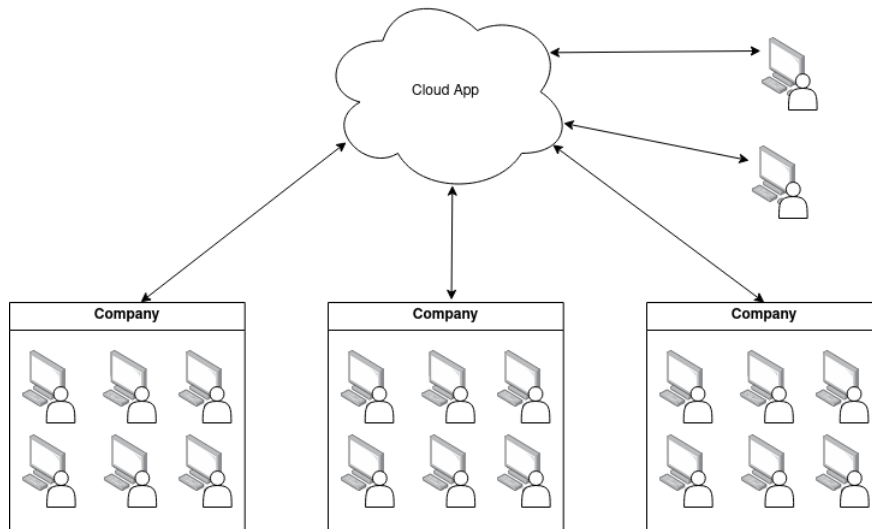


Figure 1.2: Multi tenancy in cloud applications.

PROCEED (short for PROcess EnginE in a Distributed environment) is a cloud application, it is a decentralized Business Process Management System. PROCEED uses BPMN at its core to model and execute business processes. BPMN (Business Process Model and Notation) is a standardized graphical notation used for documenting business processes. BPMN is typically used inside of organizations to illustrate sequences of tasks, decision points, and interactions within various business processes, providing a standardized visual representation.

PROCEED offers two products:

- Distributed Process Engine (DPE): the DPEs execute bpmn processes in a distributed fashion.
- Management System (MS): the MS is a cloud application that gives users a graphical interface to manage their processes and deploy these to the DPEs.

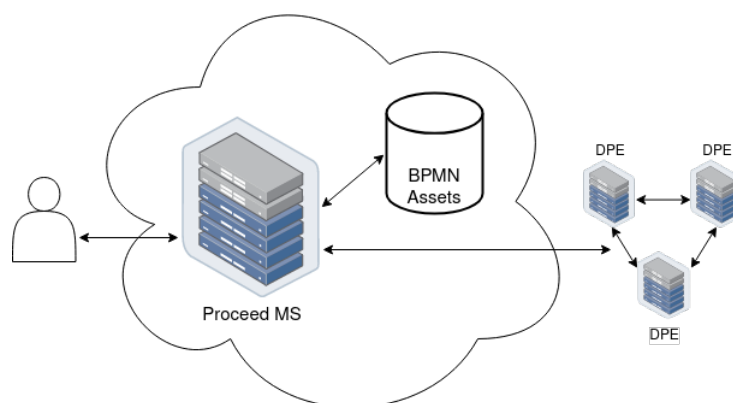


Figure 1.3: Overview of PROCEED.

Currently, the MS doesn't support multi-tenancy, it only supports individual users, which means that each user has only access to his assets. For this reason, it presents many difficulties for organizations, as there is no convenient way for multiple members of the organization to work together on assets in the MS. Furthermore, any admin user in the MS has the right to view and modify every asset, which presents privacy risks for organizations.

For these reasons, the only viable option for organizations to use PROCEED, is to spin up an instance of the MS on their own computers, which renders most benefits of cloud applications void.

This thesis implements multi-tenant functionality into the PROCEED Management System by introducing the concept of Environments.

Each tenant will be able to work in their isolated environment, which lives in a central instance of the Management System. Additionally, each account implicitly has its own Environment allowing users to work on personal projects, keeping these separated from any other users.

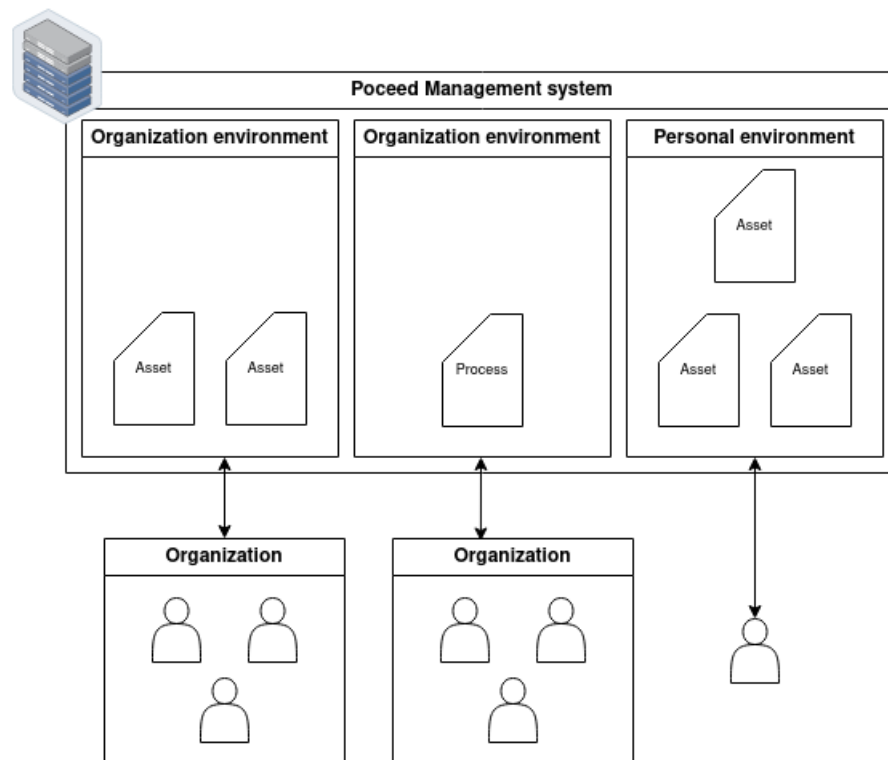


Figure 1.4: Environments in PROCEED.

Additionally, environments will provide a more sophisticated way to manage assets. Currently, the MS only provides labels for organizing assets, users cannot add, delete nor modify these labels. Environments will include a folder structures to organize BPMN assets, which provides more flexibility than labels.

2 Research Questions

3 Task List

1. Functional

- a) The MS must be able to hold multiple environments and let users access them concurrently.
- b) The MS has to support two types of environments: personal and organization environments.
 - i. Every user has to have a personal environment, of which only he can be a member.
 - ii. Organization environments must be able to have multiple members and roles that control what each member can do.
 - iii. Environments must have a folder system to store assets.
 - A. Find a suitable abstraction to represent folders in a database.
 - B. Ensure privacy between environments.
escribir aca ya que tiene que haber herencia reescribir should be modified no es tan fuerte/
 - C. Existing access managemnt has to be adapted to fit folders: Currently in the MS, conceptually, all Assets are stored in one folder, and access rules apply to each element of this folder. With a new folder structure, how roles act on different resources should be modified, depending on what folder the resources are in.
- c) Environments must be isolated from each other, so that users can only access assets in environments they are members of.
- d) Users must be able to be members of multiple environments and carry out actions in each one of them.
- e) The MS's preexisting role system must be adapted to fit environments: The proceed management system already has a role system in place to manage user's access to resources, these roles need to be modified for them to work with environments.
 - i. Ensure roles are always enforced in the backend.
 - ii. The frontend UI must adapt to a user's roles, by only showing options that the user has permission to do.

2. Non functional

- a) Try to keep changes to the existing codebase to a minimum.
- b) The same data structure should be used for both personal and organization environments.

- c) The user interface for navigating folders should be intuitive.
- d) Implement environments in a way that enables a good developer experience.
 - i. Create simple abstractions for the backend code of the MS, that allow to acknowledge a user's environment with minimal effort.
 - ii. Create a simple abstraction for the frontend, that makes it so, that the frontend doesn't need to know what environment the user belongs to.
- 3. Environments should be easy to create, manage and delete.
 - a) The frontend should provide intuitive interfaces for creating and deleting organization environments. // not functional
 - b) Provide well documented APIs to create, manage and delete environments both for the frontend and the backend. needs more description. ldap
 - c) Environments should provide the option to import an existing user database.

4 Related Work

4.1 OAuth 2.0 and OpenID Connect

OAuth is an open standard for access delegation, commonly used as a way for users to grant client applications access to their information on other applications. OAuth was born as a necessary security measure, to avoid sharing plaintext credentials between applications. Plaintext credential sharing, as outlined in [?] has many security risks:

1. Applications are forced to implement password authentication, to support the sharing of plaintext credentials.
2. Third party applications gain overly broad access to the user's account.
3. Users cannot revoke access to specific third party applications.
4. If any of the third party applications are compromised, the user's account is at risk.

OAuth addresses these issues by decoupling the client application from the role of the resource owner, meaning that the client application will not get a full set of permissions to the user's account. Instead of handing his credentials to the third party application, the resource owner signs in, in the application's website which then issues an access token to the client application. This method avoids the user having to share his credentials with third party applications.

4.1.1 OAuth 2.0 Roles

OAuth 2.0 defines four roles for participants in the protocol flow:

1. Resource owner: The entity that can grant access to a protected resource, typically this would be an end user of a web application.
2. Resource server: The server hosting the protected resources.
3. Client: The application requesting access to the protected resources. OAuth 2.0 distinguishes between two types of clients: confidential and public clients. Confidential clients are capable of keeping their credentials confidential, while public clients, like browser-based applications, cannot.
4. Authorization server: The server that issues access tokens to the client after the resource owner has been successfully authenticated.

The resource server and the authorization server can be the same entity, but they are not required to be.

4.1.2 Authorization Grants

Authorization Grants are credentials that are issued to clients, which can be exchanged for an access token. This access token can be used to access the protected resources on the resource server. OAuth 2.0 defines four authorization grants with different flows.

4.1.2.1 Implicit

The implicit grant is very helpful for public clients, as it doesn't require confidential client credentials. This is very helpful for browser-based clients, as they can't store confidential credentials securely. In the implicit grant users are redirected to the authorization server, where they authenticate themselves and authorize the client. After which the authorization server issues an access token directly to the client, this is done so with a HTTP redirect, where the access token is embedded in the redirect URL, this way the client can extract the access token from the URL.

In this flow the resource owner only authenticates with the authorization server, thus never having to share his credentials with the client.

Implicit grants have many security risks, as the access token is exposed in the URL and can be intercepted by a malicious attacker. This is why PKCE (Proof Key for Code Exchange) was later introduced as an addition to the implicit grant [?].

4.1.2.2 Resource Owner Password Credentials

This grant type requires the resource owner to share his password credentials with the client. The resource owner's password credentials represent an authorization grant, which the client can exchange them for an access token. Even though this grant type requires the resource owner to share his credentials with the client, these are only used for one request and don't have to be stored.

4.1.2.3 Client Credentials

The client credentials grant is used when the client is the resource owner. Clients are typically issued credentials, which they can use to authenticate themselves. Clients send these credentials to the authorization server and are issued an access token.

4.1.2.4 Authorization Code

The Authorization Code grant is the most common grant type used in OAuth 2.0, it is similar to the implicit grant 4.1.2.1, as it also uses HTTP redirects and it doesn't require the resource owner to share his credentials with the client. In the authorization code grant, the client redirects the resource owner to the authorization server. There the resource owner authenticates himself and authorizes the client. After which the authorization server redirects the resource owner back to the client with an authorization code. The client then authenticates itself with his confidential credentials on the authorization server and exchanges the authorization code

for an access token. As the client needs confidential credentials, this flow is only suitable for confidential clients. The exact steps are shown in figure 4.1.

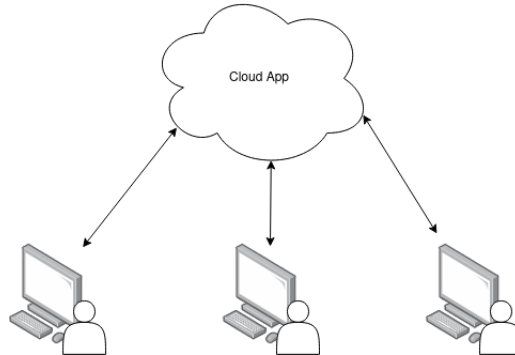


Figure 4.1: Cloud Application.

4.2 PROCEED's role system

The PROCEED MS uses a Role-Based Access Control (RBAC) system to manage user authorization, i.e. to determine what actions a user can perform. Roles can be seen as bundles of permissions, which are granted to users. A user can have multiple roles. Typically roles are assigned to users based on their job function. RBAC can be advantageous since they can be assigned to multiple users and don't change often, making them easier to manage than individual permissions.

5 Concept and Design

At the moment the Proceed Management system stores and manages all user assets the same for all users. There is no option to create assets such that only a specific group of people have access to it, meaning it is unsuitable for companies and large teams for which collaboration is important. This behaviour will be changed by the implementation of the beforementioned workspaces.

Workspaces are abstractions that enable asset isolation. A fundamental principle of this system is that every asset belongs exclusively to one workspace. Workspaces contain hierarchical structures, similar to folders, in which assets can be stored. The existing implementation of the Management System supports Role-based access control (RBAC). Which means that access to resources is determined by a user's roles. The implementation of workspaces will transform the existing roles, so that each role only belongs to one workspace, and so that it only affects the assets in that workspace. Roles will also be adapted for BPMN assets, they will be applied on the folders that the BPMN assets are stored in, instead of being defined globally, where they apply to each asset. Roles will cascade down the folder structure and be applied to all the assets in these folders as before. Furthermore users will now be able to see BPMN assets

We distinguish between two types of workspaces: company workspaces and personal workspaces. Personal workspaces will offer a more simplified feature set compared to company workspaces.

Every user is automatically provided with a personal workspace, offering an individualized space within the system. Within his workspace, a user will be granted the admin role. Roles in a personal workspace will not be able to be modified. Personal workspaces also cannot have members, the only way to enable others to work on it's assets will be to share them with the proper permissions.

Company workspaces will introduce roles that are associated with a hierarchical structure and cascade down through it, enabling role-based access control at multiple levels. Company workspaces can accommodate multiple members in contrast to personal workspaces.

In order to facilitate cooperation, the implementation will also allow sharing assets to users outside or inside the asset's workspace and granting temporary access to user's that aren't members of the workspace to the workspace.

Company workspaces can be created by normal users, for this, they will have to input data about their company. After the workspace is created, the user is granted the role of administrator and can add new members.

The frontend will include a interface for creating company workspaces, for adding members to the workspace, for managing the structure and roles of the workspace. In the context of company workspaces, when a user is created through the company workspace's interface, the user will be a regular Proceed user, with the only distinction being, that he will automatically be part of the company workspace. The Management System should also have the ability to

import or interact with existing user Databases. This will facilitate team managers, to create all the accounts they need for their team at once. The Management system frontend will show users only the contents of one workspace at the time. It will also include a simple way to view the workspaces a user belongs to and switch between them. The frontend will include an interface for sharing assets and managing the permissions of the shared assets.

Checklist

1. Interface for creating and deleting company workspaces.
2. Adapt roles to workspaces.
 - Each role has to belong to one workspace
 - Roles have to cascade down the folder structure of workspaces
 - Roles have to roughly keep the same functionality as before.
3. Adapt asset storage for workspaces.
 - Find a suitable solution to store assets.
 - Ensure privacy between workspaces.
4. Adapt the Management System's API to take into account workspaces.
 - Make sure that a the requester specifies the workspace he is referring to.
 - Make sure that the requester belongs to the workspace where the asset he is trying to access is located.
 - Implement an endpoint for users to get the workspaces they belong to.
 - Implement an endpoint for users to manage the workspaces they belong to.
5. The Management System frontend will only have one active workspace at a time.
 - All actions that manipulate assets, roles or create users, will be performed on the active workspace.
 - Only the assets of the active workspace will be shown.
 - There will be a clear indication of what workspace is active
 - There will be an easy way to switch between workspaces.
6. Interfaces for managing different aspects of company workspace's members.
 - Option for creating and deleting users.
 - Basic creation with manual input.
 - Import existing user Databases.
 - Option for inviting users to the workspace.
 - Option to manage users' roles.
7. Users should be able to share assets.
 - Allow users to share assets with other users.
 - A person who shares something will be referred to as a sharer and the recipient as a sharee
 - Users will only be able to share assets if their roles allow it.

- Each shared object has associated permissions with it, which restrict the sharee's ability to access and manipulate it.
- Sharees will be able to view and select all workspaces where they have at least one shared asset.
 - Sharees will see the folder structure of the workspace, but they won't see any of the assets in them, besides the ones shared with him
- The frontend will implement a Interface to share assets.
- The frontend will implement a Interface to manage shares.
- The frontend will indicate to the sharer, in the asset overview that an asset is shared.
- Users that are allowed to view an asset will also be able to see the users that it has been shared to.

6 Implementation

Describe the details of the actual implementation here...

7 Evaluation

The evaluation of the thesis should be described in this chapter

8 Conclusion

Describe what you did here

List of Tables

List of Figures

1.1	Cloud Application.	1
1.2	Multi tenancy in cloud applications.	2
1.3	Overview of PROCEED.	2
1.4	Environments in PROCEED.	3
4.1	Cloud Application.	11

Appendices

Appendix 1

```
1 for($i=1; $i<123; $i++)  
2 {  
3     echo "work harder! ;)";  
4 }
```