lstdefinelanguagejson basicstyle=, numbers=left, numberstyle=, stepnumber=1, numbersep=8pt, showstringspaces=false, breaklines=true, frame=lines, backgroundcolor=, literate= *001 111 221 331 441 551 661 771 881 991 ::1 ,,1 {{1 }}1 [[1]]1,

Multi-Tenancy in Cloud Applications on the Example of PROCEED

by

Felipe Trost

Matriculation Number 456129

A thesis submitted to

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

Bachelor's Thesis

October 7, 2024

Supervised by:
Prof. Dr. Axel Küpper

Assistant supervisor:
Kai Grünert

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, October 7, 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

In today's digital age, businesses heavily rely on cloud applications, software tools that are accessed and run entirely over the internet. These tools represent a paradigm shift from traditional software applications, where the majority of the workload happened on the user's device. Shifting some of the workload to the cloud offers many advantages:

- Accessibility: They can be accessed anywhere from anywhere with an internet connection.
- Cost-Efficient: Most cloud applications implement a payment structure, where users pay based on how much they use the application.
- Collaboration: Typically, collaboration is easier since everything can be found in one place, instead of having to send files back and forth.
- Data safety: All files are stored by the application in the cloud, and they don't have to be stored in the user's device, which could be lost, stolen or damaged.
- 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.

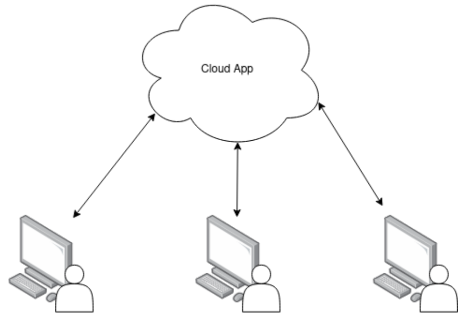

Figure 1.1: Users can access cloud applications from any device with an internet connection.

One very common feature that makes these benefits possible is called "multi-tenancy". Multi-tenancy is a software architecture in which one single instance of an application can be used by many different users or organizations at the same time. Without multi-tenancy, each user or organization would need to run the application on their own servers or computers, largely negating the numerous benefits listed earlier.

Think of it like a big apartment building. Each tenant (user or organization) has their own private space (their assets), but they're all using the same building (the cloud application).

Many popular cloud applications use this approach. For example, when you use Microsoft Teams, Slack, or Asana, you're sharing the application with many other companies, but you only see and interact with your own team.

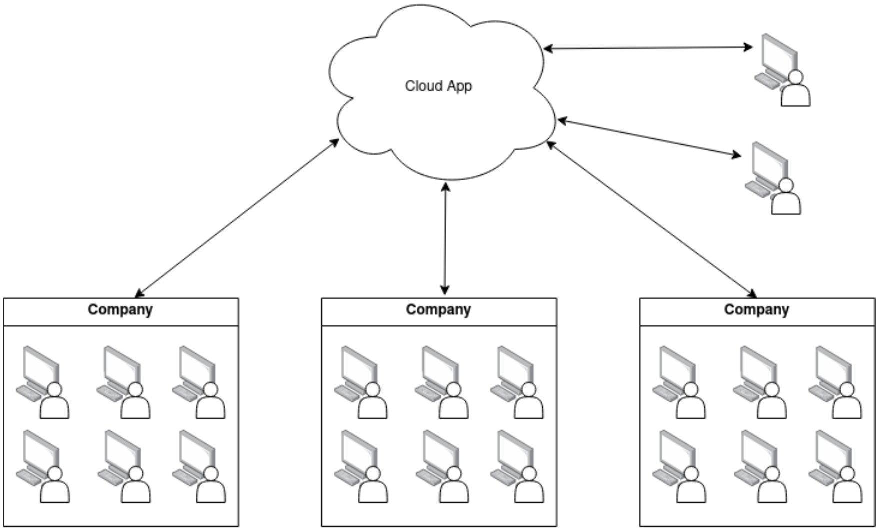

Figure 1.2: Multi tenancy in cloud applications: the same instance of the cloud application, can be used by different tenants, with different structures, without them knowing about each other.

PROCEEDis a 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 for short): the DPEs execute BPMN processes.
- Management System (MS for short): the MS is a cloud application that gives users a graphical interface to work on their BPMN processes and deploy these to the DPEs.

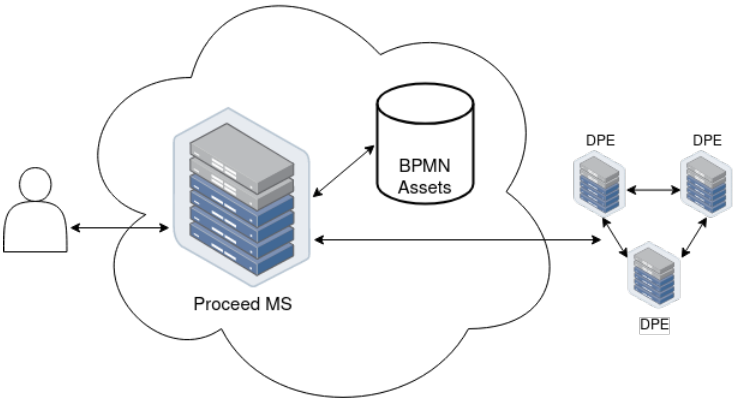

Figure 1.3: Overview of PROCEED: users interact with the Management System, to create and manage BPMN models. The Management System has the ability to deploy these models to the Distributed Process Engines.

Currently the MS lacks full multi-tenancy support, it only supports individual users and doesn't fully support organizations. For organizations to be supported, members of the organization need to be able to have a shared workspace, where they can work on the same assets. However, PROCEED only supports universal sharing, meaning that all users would be able to see the shared assets. Furthermore, even if it was possible to share assets only to a group of users, it would be very cumbersome and error-prone.

For this reason, this thesis implements multi-tenant functionality into the PROCEED MS by introducing the concept of Environments. The MS should be able to hold multiple isolated environments, where tenants can work on their assets. Each user, who signs in, should automatically have their own Environment, this allows users to work on personal projects. Organizations will be able to create environments where multiple members can work together. Additionally, environments should include a folder structure to improve the organization of assets.

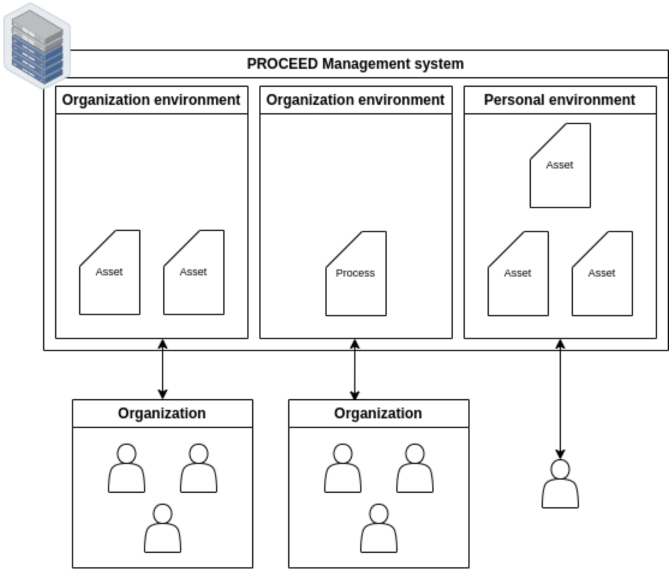

Figure 1.4: Goal of this thesis: tenants can work in their own isolated environments in the PROCEED Management System.

2 Research Questions

1. Environment Representation: How can we model environments within the existing PROCEED database schema to ensure the following:
    - Data integrity: find a schema that facilitates data consistency after updates.
    - Asset-Environment Association: find a schema that associates assets with their respective environments while maintaining a clear and consistent data model.
    - Efficient: find a schema that allows to efficiently query the database.
2. How can we extend the existing PROCEED role system to incorporate environment-specific permissions?
3. How can organizations model their hierarchical role structures within environments to allow members of the organization to have different levels of access to assets?
4. How can the Management System allow users to manage assets on different environments?
5. How can we minimize the impact on existing code while ensuring seamless integration?

3 Task List

1 Functional

   I The MS has to support two types of environments: personal and organization environments.

      A Every asset in the MS must be stored in one and only one environment.

      B Assets stored in one environment can only be accessed by members of that environment.

      C Every user has to have a personal environment, of which only he can be a member.

      D Organization environments must be able to have multiple members and roles that control what each member can do.

      E Environments must have a folder system to store assets.

         i Find a suitable abstraction to represent folders in a database.

         ii Ensure privacy between environments.

         iii 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.

            a Find a suitable inheritance model for roles based on the folder structure of an environment (e.g. a user with a role in a parent folder, can perform actions in all subfolders).

            b Ensure roles are always enforced in the backend.

            c The frontend UI must adapt to a user's roles, by only showing options that the user has permission to do.

   II The MS must be able to hold multiple environments and let users access them concurrently.

   III Users must be able to be members of multiple environments and carry out actions in each one of them.

2 Non functional

I keep changes to the existing codebase to a minimum.

II The same data structure should be used for both personal and organization environments.

III The user interface for navigating and managing folders and environments should be intuitive and easy to use.

IV Prioritize developer experience by creating clear abstractions and APIs.

   A Create simple abstractions for the backend code of the MS, that allow to acknowledge a user's environment with minimal effort.

   B Create a simple abstraction for the frontend, that facilitates adapting the Interface for each.

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 adresses 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. Insetead of handing his crendentials 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 succesfully 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 thmeselves and authorize the client. Afterwhich 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 typycally issued crendentials, 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. afterwhich 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.

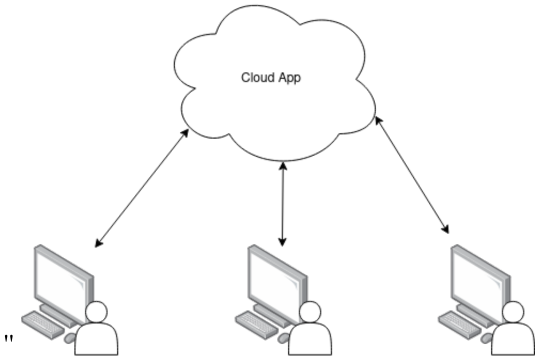

Figure 4.1: Cloud Application.

4.1.3 OpenID Connect

OpenID (OIDC for short) Connect is an identity layer built on top of the OAuth 2.0. While OAuth 2.0 focuses on authorization (granting clients access to resources), OIDC extends this to authentication. Since OIDC deals with authentication, I will call the resource owner the user from now on.

OIDC uses as its base either the Authorization Code flow 4.1.2.4 or the Implicit flow 4.1.2.1 and it introduces a new type of token called an ID Token. This ID Token is a JSON Web Token (JWT) that contains minimal information about the authenticated user Most importantly, the Id Token carries a Subject identifier, which is a unique identifier for the user in the <- finish here The Id Token is sent alongside the Access token to the client, on which step this happens depends on weather the Authorization Code flow or the Implicit flow is used.

In the OIDC flow, after the client obtains the Authorization Code, it exchanges it for both an Access Token (as in OAuth 2.0) and an ID Token. The client can then validate the ID Token to ensure it's genuine and extract the user information contained within.

In essence, OIDC allows clients to obtain information about the authenticated user in a standardized way.

4.2 PROCEED's Assets

Assets are Objects that users can create and manage through the MS's interface. When referring to assets, we are only talking about the core features of the MS, not objects that aid in the usage of the MS, like Roles .e.g., which only helps with managing access to assets. Currently the MS supports the following assets:

1. Processes, Project and Templates: These objects store BPNN at their core.

2. Task:
3. Machine: Object that represents a server running Distributed Process Engine. The machine is used to manage properties of the server.
4. Execution: An execution represents a process that is being executed distributedly.

4.3 PROCEED's role system

The PROCEED MS uses a Role-Based Access Control system to mange user authorization 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 and all the permissions of the roles are additively combined. That is, by adding a permission, a user can never do less than before. 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.

4.3.1 MS's Role System Terminology

The following terms are important to understand the role system in the PROCEED MS:

- Resource: A resource is any protected entity in the management system, that can be accessed by users.
- Action: An action is a specific operation that can be performed on a resource.
- Permission: A permission is a tuple of resource type and action, which specifies that a user can perform the action on the resource instances. Optionally a permission can have conditions that have to be met the by resource instances, for the user to be able to perform the action.
- Role: A role is a set of permissions. Roles can be assigned to users, which then inherit the role's permissions. Roles can have expiration dates, after which all permissions are revoked.

4.3.2 MS's resources and actions

The following are the resource types that are used in the PROCEED MS: `Process`, `Project`, `Template`, `Task`, `Machine`, `Execution`, `Role`, `User`, `Setting`, `EnvConfig`, `RoleMapping`, `Share`, `Folder`.

These are the actions that can be performed on these resources: `none`, `view`, `update`, `create`, `delete`.

4.3.3 MS's roles in CASL

The PROCEED MS uses [1]CASL to implement Rules. CASL is an isomorphic authorization JavaScript library. To enforce authorization CASL has abilities, which are assigned to users.

[1] https://casl.js.org/v6/en/

Abilities expose functions to check weather a user can perform an action on a resource. Abilities are defined by four parameters: user action, subject, fields, conditions. User actions and subjects are analogous to actions and resources 4.3.1.

CASL differentiates between subject type and subject instance. A subject instance is a specific instance of a subject type, e.g. a specific process users are working on, is an instance of the resource type "Process".

Fields are used to specify which fields of a resource instance an action can be performed on, e.g. a user can update a process's name, but not its id, or creation date.

Conditions are used to specify additional conditions that have to be met by a resource instance, for a user to be able to perform an action on it. E.g. a user can only update a process if he created it.

```
import { defineAbility } from '@casl/ability';

class User {
  constructor(id) {
    this.id = id;
  }
}

class Process {
  constructor(user, name) {
    this.authorId = user.id;
    this.createdOn = new Date();
    this.name = name
  }
}

function abilityForUser(user){
  return defineAbility((can, cannot) => {
    can('delete', 'User', {id: user.id});

    can('update', 'Process', ['name'], {authorId: user.id});
  });
}

const user1 = new User(1);
const user1Ability = abilityForUser(user1);
const user1Process = new Process(user1, 'some process');

const user2 = new User(2);
const user2Ability = abilityForUser(user2);

user1Ability.can('update', 'Process'); // true
user1Ability.can('update', user1Process, 'name'); // true
user1Ability.can('update', user1Process, 'createdOn'); // false

user1Ability.can('delete', user1); // true
user1Ability.can('delete', user2); // false

user2Ability.can('update', 'Process'); // true
user2Ability.can('update', user1Process); //false
```

Listing 4.1: CASL example

If there exist any possible resource instance, where the user has permission to perform an action, then the user has permission to perform the action on the resource type. E.g if a user has permission to view some process in the MS, then he has permission to view .

5 Concept and Design

This chapter outlines the key components of the implementation of environments in the MS. The core components are users, environments, roles, and assets. In essence the concept can be summarized as follows: users can be part of to multiple environments. Environments hold Assets. Users that are part of an environment can work on the assets that are stored in it. What a user can do with an asset is determined by the roles that the user has in the environment that the asset is stored in. All other components that will be introduced will help to manage and enforce these relationships.

5.1 Users

Users represent individual people utilizing PROCEED. A single person can have one or more users, but each user is intended for individual use. To facilitate the exploration of the MS, without creating an account, users can use the MS as guests. Thus, we differentiate between authenticated users and guest users. Guest users have the ability to transition to authenticated users whilst retaining their assets.

All users have a personal environment 5.4.1 in which they can create and manage assets freely. Authenticated users can also be part of organization environments 5.4.2 where they can collaborate with other users.

5.2 Folders

Folders are nodes in a rooted tree structure with a name and a description. Folders can contain other folders and assets. At the moment writing, folders only support processes, but they could be extended to support other types of assets like the ones described in 4.2.

Each environment will have a folder structure to store its processes. The root folder is created when the environment is created. Each root folder and all of its children are contained by one and only one environment 5.4.

Folders are intended to allow users to mirror the hierarchical structure of their organization and of its projects.

5.3 Assets

All assets within the MS 4.2 will be modified, so that each asset instance establishes a clear association with a single environment. Processes will be contained within folders, explicitly indicating to what environment they belong. Other asset types will store a direct reference to their environment, without being contained in a folder structure, this can be seen as a flat folder structure.

5.4 Environments

Conceptually environments are the data structure in which everything, other than users, is stored. Users aren't stored in environments as they can be a part of multiple environments, so they can't be contained in only one environment. Instead, the MS stores memberships that specify that a user is part of an environment. Everything else, assets and roles, belong to exactly one environment.

We distinguish between two types of environments: personal environments and organization environments.

5.4.1 Personal Environments

Personal environments are assigned to each user once they sign in. The user for which the environment is created, is the only member of this environment, and is therefore called the owner. No other users be a part of this environment. Personal environment only allow users to create and manage processes and folders, other Features that the MS offers are disabled for personal environments and can only be used in organization environments 5.4.2.

5.4.2 Organization Environments

Organization environments are intended to be used by organizations, thus they can have a name, description and a logo. Organization environments extend the feature set of personal environments, the enabled resources 4.3.2 for organization environments can be determined when deploying a MS instance.

Organization environments can also have multiple Users that are part of it, these are called members.

5.5 Roles

Before the implementation of this thesis, roles in the MS 4.3 were global, meaning that their permissions applied to all assets in the MS. With the introduction of the folder structure in organization environments, this no longer makes sense. Folders allow organizations to mirror their hierarchical structure, but this wouldn't be entirely useful if roles were still global. For this reason, roles can now be associated to a folder. Roles that are associated with a folder cascade down the folder structure, i.e. a role associated to a folder will also apply to all of its children. Roles that aren't associated to a folder will continue to apply to all assets in the environment. As roles are meant to mirror a users position in an organization, they're only available for organization environments.

If a user's role allows him to view assets and is associated to a folder, then the user also has the permission to view all parent folders of the folder the role is associated to. But this is only restricted to the parent folders, not the contents of the parent folders. This allows users to navigate the folder structure until they reach the assets they're allowed to view and manage.

5.5.1 Default roles

For each organization environment two roles will be created, which cannot be deleted and cannot be associated to a folder:

- `@admin`: This role has all permissions for all assets in the organization environment and it is first assigned to the user that creates the organization environment. Only users with the `@admin` role can add new users to this role.
- `@everyone`: The permissions in this role apply to for all the users that are part of the organization environment. The permissions in this role start out empty, but can be modified.

6 Implementation

6.1 Users

We authenticate users by storing a JWT [1] cookie in their browser. This cookie is then parsed by the MS backend and if the cookie is valid and it stores the id of an existing user, the user is considered authenticated. If the user couldn't be authenticated, he is redirected to the sign in page.

6.1.1 Guest Users

Users that aren't signed in can choose to try the MS out as a guest, this doesn't require the user to input any personal information.

For storing guest user data, one could take one of two approaches: storing the data in the user's browser or storing it in the MS's database, alongside the data of authenticated users. Storing the data locally has two great benefits: The MS doesn't have to store data of users who might never return and the MS would become less susceptible to an attack where the attacker tries to use up as much space as possible in the MS's database. However, this approach has one key downside, the MS would have to implement two storage solutions and accordingly switch between them. The added complexity of storing guest user's data locally isn't worth the benefits, so we decided to store a guest user's data in the MS's database.

If a user chooses to try the MS out as a guest, a new user entry is created in the

6.1.2 Authenticated Users

```
{
    isGuest: false;
    emailVerifiedOn: Date | null;
    firstName?: string | undefined;
    lastName?: string | undefined;
    username?: string | undefined;
    image?: string | null | undefined;
    favourites?: string[] | undefined;
```

[1] https://www.rfc-editor.org/rfc/rfc7519.html

```
    id?: string | undefined;
    email?: string | undefined;
}
```

Listing 6.1: CASL example

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

Appendices

Appendix 1

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