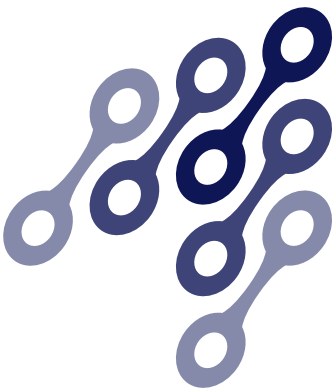
POLITECHNIKA CZĘSTOCHOWSKA

WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI



PRACA MAGISTERSKA

Implementacja usług z wykorzystaniem OpenLDAP

Implementation of services using OpenLDAP

Paweł Szubert

Nr albumu: 99137

Kierunek: Informatyka

Studia: stacjonarne

Poziom studiów: II

Promotor pracy: dr inż. Ireneusz Szcześniak

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, 2012

Spis treści

Abstract

In every company which is using computers and providing employee some intranet services, there is a problem with storing employees' data.

Usually, each application or service has its own database, so users' data are decentralized and located in separated databases, which can cause a lot of problems.

From administrator's perspective it is necessary to monitor many database services. When new employee is hired, it is necessary to create accounts for him in all services and assign permissions. When he is leaving company, administrator has to remove accounts one by one.

Solution for all those problems is to create central database of users, that can be used by all services. Tool that is the most suitable to that needs is LDAP server. Due to its flexibility and big scalability, there is a lot of projects that are using LDAP as authorization backend. Because of simple API, it is extremely easy to add LDAP support in our software.

Wstęp

W każdej firmie czy organizacji, której działalność wiąże się z wykorzystaniem komputerów, a pracownikowi udostępniane są jakieś zasoby intranetowe po pewnym czasie powstaje problem związany z zarządzaniem uprawnieniami do takich zasobów. Zasobami takimi są zarówno konta na lokalnych komputerach, serwisy działające w ramach lokalnego intranetu czy usługi przeznaczone dla programistów takie jak system kontroli wersji czy system zarządzania projektem. Każdy z tych zasobów zazwyczaj korzysta z własnej bazy użytkowników. Taka decentralizacja danych o użytkownikach jest kłopotliwa zarówno dla administratorów jak i dla samych użytkowników.

Patrząc z punktu widzenia użytkownika podstawową wadą jest konieczność pamiętania wielu haseł do różnych usług, a w przypadku potrzeby ich zmiany potrzeba skorzystania z wielu różnych narzędzi czy też stron intranetowych.

Dla administratora taka decentralizacja oznacza konieczność nadzorowania wielu rozproszonych baz danych o użytkownikach. W przypadku zatrudnienia nowego użytkownika konieczne jest założenie mu i skonfigurowanie konta w wielu różnych usługach, w przypadku odejścia takiego pracownika - zablokowanie lub usunięcie konta. Kiedy serwisów czy usług jest mało nie stanowi to jeszcze problemu, ale kiedy jest ich kilka czy kilkanaście i różne usługi wykorzystywane są przez różne działy firmy nie trudno o przeoczenie jednego serwisu. Przeoczenie takiej blokady może okazać się tragiczne w skutkach - znane są przypadki, kiedy zwolniony pracownik w ramach zemsty wykradał poufne dane firmowe, co było możliwe dlatego, że wciąż miał dostęp do firmowej poczty czy witryny intranetowej.

Rozwiązaniem, które pozwala na uproszczenie administracji takimi danymi jest stworzenie centralnego katalogu użytkowników. Narzędziem, które najlepiej nadaje się do tego celu jest serwer LDAP. Ze względu na swoje cechy takie jak duża skalowalność i możliwość dopasowania do własnych potrzeb, wiele oprogramowania posiada już wsparcie dla pobieranie danych autoryzacyjnych z katalogu LDAP. Dzięki prostemu API i dużej ilości

bibliotek dla wielu języków, możliwa jest także szybka implementacja autoryzacji z wykorzystaniem katalogu LDAP we własnym oprogramowaniu.

Cel pracy

Celem niniejszej pracy jest opracowanie konfiguracji serwera LDAP, tak aby mógł on służyć jako źródło danych autoryzacyjnych dla usług takich jak konta na serwerze, możliwość zarządzania własnym repozytorium GIT i udostępniania go innym studentom. Do przechowywania informacji o uprawnieniach do repozytoriów wykorzystany zostanie także katalog LDAP.

Aby użytkownicy mogli rejestrować się i zarządzać repozytoriami konieczne jest też stworzenie serwisu internetowego dokonującego modyfikacji wewnątrz katalogu.

Struktura pracy

Praca składa się z trzech rozdziałów. Pierwszy zawiera podstawowe informacje na temat tego co to jest LDAP, w jaki sposób odbywa się komunikacja pomiędzy klientem a serwerem i w jaki sposób przechowywane są dane wewnątrz katalogu. W rozdziale tym opisany jest także sposób konfiguracji serwera LDAP.

W rozdziale drugim zawarte są informacje na temat instalacji i konfiguracji serwera OpenLDAP. Znajdują się w nim także informacje na temat konfiguracji poszczególnych usług jakie zostały zaimplementowane. Omówione zostały zmiany, jakie należy wprowadzić zarówno w samym katalogu, jak i w systemie w celu udostępnienia możliwości lokalnego i zdalnego logowania użytkowników, korzystania z repozytoriów GIT oraz ich udostępniania.

Rozdział trzeci zawiera informacje na temat portalu służącego do rejestracji użytkowników oraz zarządzania repozytoriami. W pierwszej sekcji zaprezentowane zostały biblioteki jakie zostały wykorzystane do stworzenia portalu. Następnie przedstawiona została Pythonowa reprezentacja danych przechowywanych w katalogu. Kolejne dwie sekcje prezentują sposób implementacji sekcji portalu dotyczącej zarządzania użytkownikami i repozytoriami.

1. LDAP i OpenLDAP

Aby w pełni zrozumieć czym jest LDAP (*Lightweight Directory Access Protocol*) należy na początek zdefiniować pojęcie usługi katalogowej. W najprostszym ujęciu usługa katalogowa jest to wyspecjalizowana baza danych [1, str. 18]. Istnieją jednak pewne różnice pomiędzy katalogiem a bazą danych:

- nastawienie na odczyt i przeszukiwanie a nie zapisywanie danych,
- zazwyczaj brak wsparcia dla transakcji i zaawansowanych operacji,
- duża możliwość rozszerzania i skalowania.

LDAP jest protokołem dostępowym do usług katalogowych, jednak w szerszym znaczeniu jest to także:

- model opisujący rodzaj danych jakie można umieścić w katalogu, sposób w jaki będą one ułożone oraz model uprawnień zabezpieczający dane przed nieautoryzowanym dostępem,
- format LDIF (*LDAP Data Interchange Format*) - format wymiany danych katalogowych w postaci tekstowej,
- oprogramowanie serwera LDAP,
- narzędzia pozwalające uzyskać dostęp do danych zawartych w katalogu oraz narzędzia pozwalające na jego przeszukiwanie, [1, str. 63]
- API pozwalające na dostęp do funkcji katalogu.

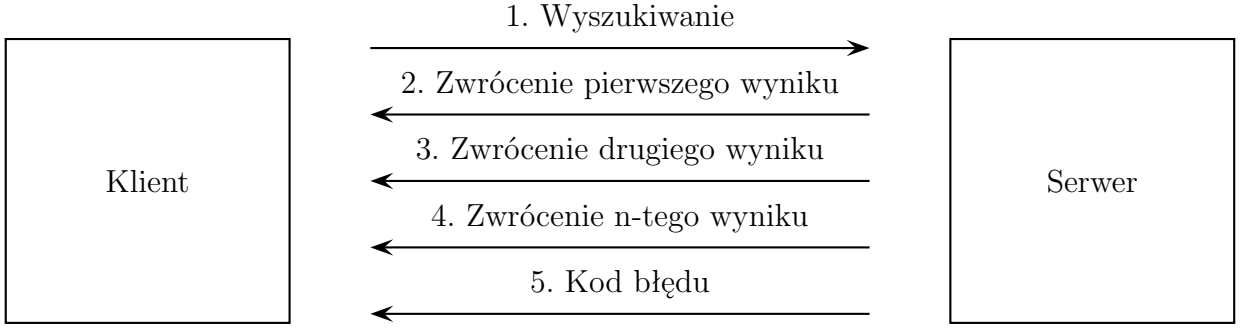
1.1. Protokół LDAP

1.1.1. Wiadomości

Protokół LDAP jest oparty o komunikaty (wiadomości). Klient LDAP po sformułowaniu zapytania przesyła je do serwera, który przetwarza daną wiadomość, a następnie

odpowiada klientowi przesyłając odpowiedź w postaci jednej lub serii wiadomości (rys. 1).

Kiedy klient LDAP chce dokonać wyszukiwania danych w katalogu wysyła do serwera wiadomość opatrzoną wygenerowanym przez siebie identyfikatorem. Serwer po wyszukaniu danych wysyła odpowiedź oraz, w kolejnej wiadomości, kod błędu. Wszystkie wiadomości wysyłane przez serwer zostają opatrzone tym samym identyfikatorem, który znajdował się w zapytaniu klienta. W przypadku, kiedy serwer odnajdzie wiele wyników, każdy z nich przesyłany jest w odrębnej wiadomości.



Rysunek 1. Przesyłanie komunikatów pomiędzy serwerem a klientem

1.1.2. Operacje

Każda wiadomość, jaka jest przesyłana do serwera musi zawierać informację o tym jaką operację serwer ma wykonać. LDAP definiuje kilka rodzajów operacji. Podstawowe z nich to:

- operacja wyszukiwania: search,
- operacje modyfikujące dane: add, delete, modify, modify DN,
- operacje uwierzytelnienia: bind, unbind.

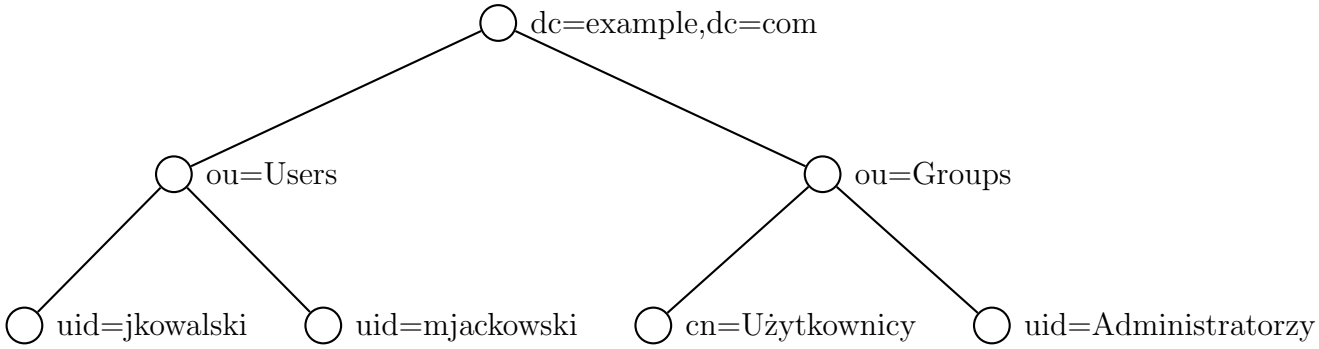
1.2. Struktura danych w LDAP

Dane w katalogu LDAP przechowywane są w postaci wpisów (*entry*). Wpisy umieszczone są w określonej hierarchii w formie drzewa (rys. 2). Mimo tego, że katalog LDAP ma strukturę hierarchiczną żadna konkretna hierarchia nie jest wymuszona. Administrator może sam określić gdzie w drzewie znajdowały będą się konkretne wpisy, tak aby

Tablica 1. Przykładowe atrybuty i ich wartości

Atrybut	Wartość
`cn:`	Jan Kowalski
`sn:`	Kowalski
`uid:`	jkowalski
`mail:`	jkowalski@example.com

zarządzanie i wykorzystanie takiego drzewa było jak najłatwiejsze i dopasowane do indywidualnych potrzeb.



Rysunek 2. Przykładowe drzewo LDAP

Każdy ze wpisów posiada jakieś atrybuty (*attributes*) oraz wartości tych atrybutów (*values*) (tabela 1).

To jakie atrybuty może posiadać dany wpis zależne jest od tego jakiej jest on klasy (*objectClass*). Wpis może posiadać jedną klasę podstawową (*structural*)[1] oraz kilka klas dodatkowych (*auxiliary*). Każda klasa definiuje listę atrybutów jaką obiekt musi posiadać (atrybuty z opcją `MUST`) oraz atrybuty opcjonalne (z opcją `MAY`). W klasie określone jest także czy dany atrybut może występować jedynie raz czy kilka razy. Kolejną rzeczą jaka definiowana jest w klasie jest typ atrybutu (tekstowy, numeryczny czy binarny) oraz to w jaki sposób ma przebiegać porównywanie atrybutów.

Każdy wpis musi zawierać atrybut `entryDN` (*Distinguished Name*) będący adresem danego wpisu w drzewie LDAP. Odczytując wartość `entryDN` jesteśmy w stanie jednoznacznie zidentyfikować dany wpis w drzewie LDAP. Atrybut ten tworzony jest na podstawie hierarchii wpisu w drzewie oraz jednego z obowiązkowych atrybutów wpisu. Adres

[1]Wpisów na temat klasy podstawowej może być kilka o ile klasy te tworzą hierarchię dziedziczenia.

DN wpisu użytkownika Jan Kowalski przedstawionego na rysunku 2 to `uid=jkowalski, ou=Users,dc=example,dc=com`.

Pozostałe atrybuty mogą określać różne właściwości danego obiektu.

1.3. OpenLDAP

Jedną z najpopularniejszych implementacji LDAP jest serwer OpenLDAP. Projekt ten powstał w roku 1998 na podstawie kodu źródłowego serwera LDAP z Uniwersytetu w Michigan. Od roku 2000 serwer OpenLDAP obsługuje protokół LDAP w wersji 3.

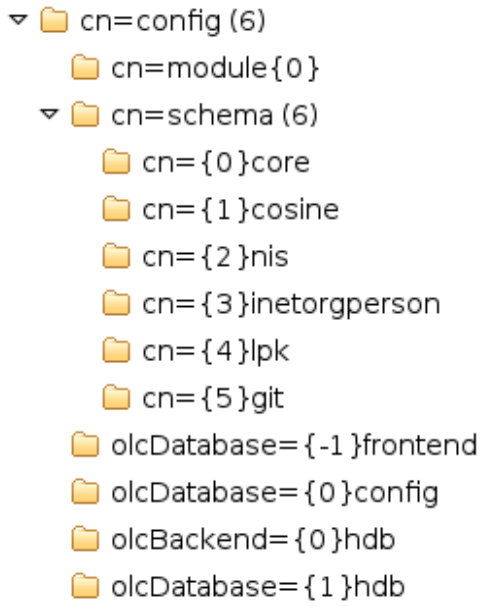
1.3.1. Konfiguracja serwera OpenLDAP

Konfiguracja serwera OpenLDAP może odbywać się na dwa sposoby: albo za pomocą pliku konfiguracyjnego `slapd.conf`, albo za pomocą katalogu `slapd.d`. Możliwość konfiguracji poprzez katalog `slapd.d` została dodana w wersji 2.3 programu. Konfiguracja za pomocą pliku konfiguracyjnego jest uznawana za przestarzałą i wypierana przez konfigurację poprzez katalog konfiguracyjny. Podstawową zaletą wykorzystania katalogu jest możliwość wprowadzania zamian w konfiguracji serwera bez jego restartowania, co nie było możliwe przy wykorzystaniu pliku.

Zawartość katalogu konfiguracyjnego jest mapowana na specjalny katalog LDAP `cn=config`3, tak więc możliwe jest konfigurowanie serwera za pośrednictwem dowolnego edytora danych LDAP, dzięki czemu administrator katalogu nie musi mieć dostępu do systemu plików serwera. Wewnątrz katalogu `slapd.d` znajduje się wiele plików w formacie LDIF, które konfigurują poszczególne składniki serwera, takie jak wykorzystane schematy, moduły czy bazy danych.

Główna gałąź `cn=config` zawiera opcje takie jak ścieżki do plików pid i args czy poziom logowania.

Gałąź `cn=module,cn=config` odpowiada za ładowanie modułów serwera. Atrybut `olcModulePath` określa ścieżkę w której przechowywane są moduły, a kolejne wpisy `olcModuleLoad` wskazują nazwę i kolejność w jakiej nastąpi załadowanie danego modułu. W przypadku modułów, które są od siebie zależne powinna zostać zachowana odpowiednia kolejność ładowania.



Rysunek 3. Katalog konfiguracyjny `cn=config`

We wpisach znajdujących się poniżej gałęzi `cn=schema,cn=config` przechowywane są definicje schematów danych jakie mogą być przechowywane w katalogu. Każdy z takich schematów przechowywany jest w osobnym pliku LDIF. Nazwy plików (czyli także nazwy `cn`) wskazują kolejność ładowania schematów. Zachowanie odpowiedniej kolejności jest ważne ze względu na możliwość dziedziczenia atrybutów i klas.

Gałąź `olcBackend=hdb,cn=config` zawiera konfigurację backendu (silnika bazy danych w jakiej serwer przechowywał będzie wpisy).

Kolejnymi pozycjami są trzy wpisy `olcDatabase`, które konfigurują poszczególne bazy danych (katalogi). `olcDatabase={-1}frontend,cn=config`, która jest ładowana jako pierwsza zawiera globalną konfigurację wszystkich katalogów, jednak opcje zawarte w tym pliku mogą być nadpisywane przez poszczególne katalogi osobno. Kolejnym zdefiniowanym katalogiem jest `olcDatabase={0}config,cn=config`, czyli właśnie katalog konfiguracyjny. Katalogiem, który jest definiowany jako ostatni jest `olcDatabase={1}hdb,cn=config`, czyli ten, który będzie przechowywał faktyczne dane.

W każdym wpisie definiującym katalog możliwe jest zawarcie wielu parametrów. Najważniejsze z nich to:

- `olcRootDN` - DN administratora,
- `oldRootPW` - hasło administratora,

- `olcDbDirectory` - katalog w którym przechowywane będą pliki danej bazy,
- `olcAccess` - listy ACL, definiujące dostęp użytkowników do poszczególnych wpisów lub atrybutów. Wpisów tego typu może być wiele.
- `olcDbIndex` - informacje na temat tego jakie atrybuty mają być indeksowane. Indeksowanie atrybutów przyspiesza ich przeszukiwanie.
- `olcDbConfig` - różne opcje konfiguracyjne bazy danych.

Podstawowa konfiguracja serwera tworzona jest podczas jego instalacji, jednak w celu uzyskania lepszej wydajności czy podniesienia bezpieczeństwa danych przechowywanych w katalogu konieczne jest dodanie indeksów czy list ACL.

2. Implementacja

2.1. Instalacja i konfiguracja oprogramowania

Spośród kilku dostępnych implementacji LDAP wybrany został serwer OpenLDAP. Jako platforma systemowa wybrana została dystrybucja Debian, dostarczająca w repozytorium pakiety OpenLDAP.

Instalacja serwera OpenLDAP dokonana została za pomocą narzędzia `apt`:

```
apt-get install slapd
```

Po instalacji należy wywołać kreator konfiguracji pakietu:

```
dpkg-reconfigure -plow slapd
```

W kreatorze tym wprowadzone zostały następujące dane:

- Omit OpenLDAP server configuration? No
- DNS domain name: `icis.pcz.pl`
- Organization name: `icis.pcz.pl`
- Administrator password: hasło administratora
- Database backend to use: HDB
- Do you want the database to be removed when slapd is purged? No
- Move old database? No
- Allow LDAPv2 protocol? No

Kreator ten utworzy podstawową strukturę katalogu oraz określi najważniejsze opcje takie jak hasło administratora oraz silnik bazy danych jaki będzie wykorzystywany przez serwer LDAP do przechowywania danych.

Po instalacji konieczne jest też ustawienie hasła administratora dla bazy konfiguracyjnej `cn=config`, poprzez dodanie atrybutu `olcRootPW` do pliku `/etc/ldap/slapd.d/`

`cn=config/olcDatabase=0config.ldif`. Wartość tego atrybutu to hasło zaszyfrowane algorytmem SHA z solą (SSHA) i zakodowane przy użyciu base64.

OpenLDAP dostarcza narzędzie `slappasswd` do szyfrowania haseł. Po jego uruchomieniu z parametrem `-h SSHA` należy podać hasło, które ma zostać zaszyfrowane wskazanym algorytmem.

```
root@debian:/# slappasswd -h {SSHA}
New password: [wprowadzenie hasła]
Re-enter new password: [powtórzenie hasła]
{SSHA}bcBEMy/5NHOjlUX4bVjPhrZzN8C9ZQIk
```

Hasło takie następnie zostało zakodowane narzędziem `base64`:

```
root@debian:/# base64
{SSHA}bcBEMy/5NHOjlUX4bVjPhrZzN8C9ZQIk
^D
e1NTSEF9YmNCRU15LzVOSDBqbFVYNGJWalBoclp6TjhDOVpRSWsK
```

W przypadku dodawania do plików LDIF danych kodowanych w base64 po nazwie parametru stosowane są dwa dwukropki.

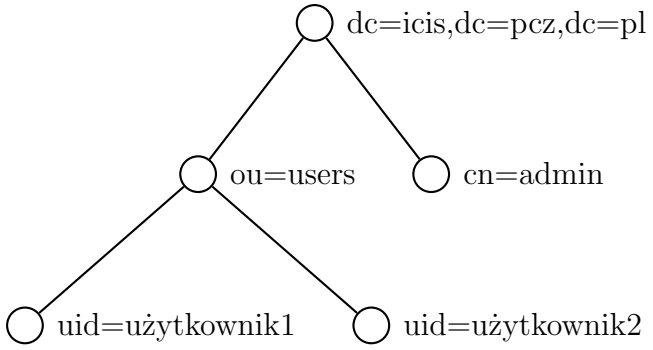
Po określeniu hasła administratora bazy konfiguracyjnej możliwe jest podłączenie się do niej za pomocą dowolnego narzędzia do zarządzania katalogiem LDAP. Dane jakie należy podać przy łączeniu się są następujące:

- user: `cn=admin,cn=config`,
- password: ustawione hasło,
- basedn: `cn=config`.

2.2. Podstawowa struktura katalogu

Podstawowym obiektem jaki będziemy przechowywali w katalogu są użytkownicy, dlatego w korzeniu drzewa `dc=icis,dc=pcz,dc=pl` wydzielona została jednostka organizacyjna (*organizational unit*) `ou=users` (rys. 4).

Jednostka organizacyjna `ou=users` służy wyłącznie do grupowania innych wpisów i zawiera jeden wymagany atrybut `ou`, który wykorzystany jest do budowy `DN` (tabela 2).



Rysunek 4. Podstawowa struktura stworzonego katalogu

Tablica 2. Wpis `ou=users`

Atrybut	Wartość
`objectClass:`	top
`objectClass:`	organizationalUnit
`ou:`	users

Wewnątrz jednostki `ou=users` dla każdego użytkownika został stworzony wpis klasy `inetOrgPerson` [19], dziedziczącej po `organizationalPerson` [18], która natomiast dziedziczy po klasie `person` [18].

Klasy te dostarczają pola opisane w tabeli 3. Do przechowywania danych wykorzystane zostały pola reprezentujące imię i nazwisko, nazwisko, nazwę użytkownika oraz adres email.

Tablica 3: Atrybuty dostępne w klasie `inetOrgPerson` oraz klasach z których ona dziedziczy.

Atrybut	Klasa	MUST/MAY	Opis
`sn`	person	MUST	Surname - nazwisko
`cn`	person	MUST	Common name - imię i nazwisko
`userPassword`	person	MAY	Hasło
`telephoneNumber`	person	MAY	Numer telefonu
`seeAlso`	person	MAY	`DN` obiektu powiązanego
`description`	person	MAY	Opis
`title`	organizationalPerson	MAY	Tytuł osoby
`x121Address`	organizationalPerson	MAY	Adres w formacie X.121
`registeredAddress`	organizationalPerson	MAY	Adres pocztowy

destinationIndicator	organizationalPerson	MAY	Kraj i miejscowość
preferredDeliveryMethod	organizationalPerson	MAY	Preferowany sposób dostarczania wiadomości
telexNumber	organizationalPerson	MAY	Numer teleksu
teletexTerminalIdentifier	organizationalPerson	MAY	Wycofany
telephoneNumber	organizationalPerson	MAY	Numer telefonu
internationalISDNNumber	organizationalPerson	MAY	Numer ISDN
facsimileTelephoneNumber	organizationalPerson	MAY	Numer faxu
street	organizationalPerson	MAY	Ulica
postOfficeBox	organizationalPerson	MAY	Numer skrytki pocztowej
postalCode	organizationalPerson	MAY	Kod pocztowy
postalAddress	organizationalPerson	MAY	Adres korespondencyjny
physicalDeliveryOfficeName	organizationalPerson	MAY	Odział pocztowy
ou	organizationalPerson	MAY	organizationalUnitName - nazwa jednostki organizacyjnej
st	organizationalPerson	MAY	stateOrProvinceName - stan lub prowincja
l	organizationalPerson	MAY	localityName - miasto lub inny region administracyjny
audio	inetOrgPerson	MAY	Nagranie dźwiękowe
businessCategory	inetOrgPerson	MAY	Kategoria biznesowa
carLicense	inetOrgPerson	MAY	Numer prawa jazdy lub tablic rejestracyjnych
departmentNumber	inetOrgPerson	MAY	Numer oddziału
displayName	inetOrgPerson	MAY	Wyświetlane imię
employeeNumber	inetOrgPerson	MAY	Identyfikator pracownika
employeeType	inetOrgPerson	MAY	Typ pracownika
givenName	inetOrgPerson	MAY	Imię
homePhone	inetOrgPerson	MAY	Telefon domowy
homePostalAddress	inetOrgPerson	MAY	Adres domowy
initials	inetOrgPerson	MAY	Inicjały
jpegPhoto	inetOrgPerson	MAY	Zdjęcie w formacie JPEG
labeledURI	inetOrgPerson	MAY	Adres URL
mail	inetOrgPerson	MAY	Adres mail
manager	inetOrgPerson	MAY	Nazwisko managera
mobile	inetOrgPerson	MAY	Telefon komórkowy

o	inetOrgPerson	MAY	Nazwa organizacji
pager	inetOrgPerson	MAY	Numer pagera
photo	inetOrgPerson	MAY	Zdjęcie w formacie G3
roomNumber	inetOrgPerson	MAY	Numer pokoju
secretary	inetOrgPerson	MAY	Nazwisko sekretarki
uid	inetOrgPerson	MAY	Nazwa użytkownika (login)
userCertificate	inetOrgPerson	MAY	Certyfikat użytkownika
x500uniqueIdentifier	inetOrgPerson	MAY	Identyfikator użytkownika
preferredLanguage	inetOrgPerson	MAY	Język
userSMIMECertificate	inetOrgPerson	MAY	Certyfikat PKCS#7
userPKCS12	inetOrgPerson	MAY	Certyfikat PKCS#12

2.3. Usługi

Podstawowa struktura katalogu przedstawiona w rozdziale 2.2 może już przechowywać niezbędne dane użytkowników, jednak jest niewystarczająca w celu udostępniania użytkownikom konkretnych usług. W zależności od tego, jakie usługi mają być udostępniane konieczne może być dodanie do wpisów użytkowników dodatkowych schematów a następnie atrybutów, które pochodzą z tych schematów. Możliwe jest jednak opracowanie podstawowego schematu, który umożliwi wprowadzenie podstawowych danych o użytkowniku do katalogu, a następnie dodawanie do niego kolejnych schematów wymaganych przez nowo wprowadzane usługi.

2.3.1. Logowanie do systemu

W celu umożliwienia użytkownikom LDAP logowania do systemu konieczne jest zarówno dostosowanie danych zawartych w katalogu, jak i wprowadzenie pewnych modyfikacji w systemie operacyjnym.

LDAP

Aby logowanie w systemie było możliwe do wpisu użytkownika należy dodać nową klasę `posixAccount` dostarczająca pola opisane w tabeli 4 [9] oraz klasę `shadowAccount` opisaną w tabeli 5 [9]. Klasa te jest zdefiniowana w pliku `nis.schema`, tak więc należy

ten plik załączyć w konfiguracji serwera LDAP. Po dodaniu wymaganej klasy konieczne jest uzupełnienie co najmniej atrybutów opisanych jako `MUST`.

Tablica 4: Atrybuty dostępne w klasie `posixAccount`.

Atrybut	MUST/MAY	Opis
`cn`	MUST	Common name - imię i nazwisko
`uid`	MUST	Nazwa użytkownika (login)
`uidNumber`	MUST	Identyfikator użytkownika
`gidNumber`	MUST	Identyfikator grupy
`homeDirectory`	MUST	Katalog domowy
`userPassword`	MAY	Hasło
`loginShell`	MAY	Powłoka logowania
`gecos`	MAY	Dane użytkownika (jak w pliku `/etc/passwd`)
`description`	MAY	Opis

Tablica 5: Atrybuty dostępne w klasie `shadowAccount`[13]

Atrybut	MUST/MAY	Opis
`uid`	MUST	Nazwa użytkownika (login)
`userPassword`	MAY	Hasło
`shadowLastChange`	MAY	Data ostatniej zmiany hasła
`shadowMin`	MAY	Minimalny wiek hasła, kiedy jego zmiana będzie możliwa
`shadowMax`	MAY	Wiek hasła, po którym użytkownik będzie musiał je zmienić
`shadowWarning`	MAY	Ilość dni przed wygaśnięciem hasła, kiedy użytkownik będzie ostrzegany
`shadowInactive`	MAY	Ilość dni po wygaśnięciu hasła, kiedy będzie ono jeszcze akceptowane
`shadowExpire`	MAY	Data wygaśnięcia konta[1]
`shadowFlag`	MAY	Pole zarezerwowane
`description`	MAY	Opis

Użytkownik powinien posiadać także grupę, tak więc w katalogu utworzona została nowa jednostka organizacyjna `ou=groups` (`ou=groups,dc=icis,dc=pcz,dc=pl`). Wewnątrz tej jednostki utworzone zostały wpisy klasy `posixGroup` dla każdego z użytkowników. Lista atrybutów dostarczanych przez klasę `posixGroup` znajduje się w tabeli 6.

[1] Liczone w dniach od 1.01.1970

Tablica 6: Atrybuty dostępne w klasie `posixGroup`.

Atrybut	MUST/MAY	Opis
`cn`	MUST	Nazwa grupy
`gidNumber`	MUST	Numer grupy
`userPassword`	MAY	Hasło
`memberUid`	MAY	Numery UID członków
`description`	MAY	Opis

System

Aby system operacyjny mógł skorzystać z danych zawartych w bazie LDAP konieczna jest instalacja odpowiednich bibliotek PAM (*Pluggable Authentication Modules*), NSS (*Name Service Switch*) oraz narzędzia `nslcd`.

NSS pozwala na zdefiniowanie baz danych dla różnych danych wykorzystywanych przez system. Lista baz oraz źródła danych określone są w pliku konfiguracyjnym `/etc/nsswitch.conf`[12]:

- passwd - baza dla danych użytkowników (domyślnie `/etc/passwd`),
- group - baza grup systemowych (domyślnie `/etc/group`),
- shadow - baza haseł (domyślnie `/etc/shadow`),
- hosts - baza hostów i ich adresów IP (domyślnie `/etc/hosts`),
- networks - baza adresów sieci (domyślnie `/etc/networks`),
- protocols - baza nazw i numerów protokołów (domyślnie `/etc/protocols`),
- services - baza usług i numerów portów (domyślnie `/etc/services`),
- ethers - baza hostów i ich adresów MAC (domyślnie `/etc/ethers`),
- rpc - baza numerów i nazw RPC (domyślnie `/etc/rpc`),
- netgroup - lista hostów i użytkowników w sieci (domyślnie pobierana z NIS).

Jak widać domyślnie większość tych danych przechowywana jest w plikach tekstowych. Aby umożliwić pobieranie tych danych z bazy LDAP konieczna jest instalacja modułu `libnss-ldapd`. Po jego instalacji do pliku konfiguracyjnego można dopisać bazę `ldap` jako kolejne źródło danych dla baz `passwd`, `group` oraz `shadow`:

```
passwd:         compat ldap
```

```
group:          compat ldap
shadow:         compat ldap
```

Kolejność źródeł danych na liście określa też w jakiej kolejności NSS będzie te źródła odpytywał. Po konfiguracji źródła danych możliwe jest już odpytanie bazy za pomocą polecenia `getent`[10]:

```
root@student:~# getent passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
pbm:x:1000:1000:pbm,,,:/home/pbm:/bin/bash
[...]
pszubert:x:20000:20000:Paweł Szubert:/home/pszubert:/bin/bash
ttestowy:x:20001:20001:Test Testowy:/home/ttestowy:/bin/bash
```

Dwóch ostatnich użytkowników o numerach UID zaczynających się od 20000 pochodzi z bazy LDAP.

Mechanizm PAM jest odpowiedzialny za autoryzację użytkowników. Aplikacje (w tym sam system) korzystające z PAM wysyłają zapytanie do podsystemu PAM, który na podstawie swojej konfiguracji przeprowadza odpowiednią autoryzację użytkownika, a do samej aplikacji przesyła jedynie informację o tym czy autoryzacja się powiodła.

PAM dzieli zadania związane z autoryzacją użytkownika na cztery grupy[15]:

- account - zadania związane z obsługa konta (wygasanie, itp),
- authentication - uwierzytelnienie - zweryfikowanie tożsamości osoby,
- password - zmiana hasła użytkownika,
- session - zadania, które muszą być wykonane po uwierzytelnieniu użytkownika.

Domyślnie w Linuksie wszystkie te funkcje spełnia moduł `pam_unix.so`, który przeprowadza uwierzytelnienie użytkownika na podstawie wprowadzonego przez niego loginu i hasła oraz bazy dostarczonej przez NSS[2].

Modułem odpowiedzialnym za autoryzację użytkowników LDAP jest `libpam-ldapd`, który także znajduje się w repozytorium dystrybucji Debian.

Kolejnym składnikiem, który jest wymagany jest program `nslcd`, odpowiadający za przesyłanie zapytań do serwera LDAP oraz za buforowanie wyników [14].

Po instalacji tych pakietów pojawia się kreator, który dokonuje podstawowej konfiguracji modułów. W kolejnych krokach kreatora należy podać:

- adres serwera LDAP,
- adres w bazie LDAP (DN) będący podstawą przeszukiwania,
- bazy NSS jakie mają być synchronizowane.

W tym przypadku będą to kolejno:

- `ldap://127.0.0.1`
- `dc=icis,dc=pcz,dc=pl`
- bazy `group`, `passwd`, `shadow`

Wszystkie opcje konfiguracyjne z których korzystają te moduły przechowywane są w pliku `/etc/nslcd.conf`[11].

Po wprowadzeniu tych zmian w systemie możliwe jest już lokalne i zdalne logowanie się użytkowników LDAP za pomocą hasła.

2.3.2. Dostęp zdalny

Zgodnie z założeniami projektu logowanie zdalne do serwera miało być możliwe przy użyciu kluczy SSH. Serwer OpenSSH pozwala na przeprowadzenie autoryzacji użytkownika na podstawie pary kluczy: publicznego i prywatnego. Klucz prywatny musi znajdować się na komputerze z którego się logujemy. Domyślne lokalizacje w których klient SSH szuka takiego klucza to:

- `~/.ssh/id_dsa` dla kluczy korzystających z algorytmu DSA,
- `~/.ssh/id_rsa` dla kluczy korzystających z algorytmu RSA,
- `~/.ssh/id_ecdsa` dla kluczy korzystających z algorytmu ECDSA,
- w przypadku protokołu SSH w wersji 1 `~/.ssh/identity`,

Możliwe jest także wskazanie klientowi SSH innego pliku klucza prywatnego wprowadzając jego ścieżkę po opcji `-i` (`ssh -i /sciezka/do/pliku/klucza user@host`) lub poprzez odpowiednie wpisy w pliku konfiguracyjnym klienta SSH.

Klucz publiczny użytkownika umieszczany jest na serwerze na który się logujemy w pliku ~/.ssh/authorized_keys. W pliku tym może znajdować się wiele kluczy publicznych.

Przechowywanie kluczy publicznych użytkowników w ich katalogach domowych jest zachowaniem domyślnym SSH, jednak w przypadku centralnej bazy użytkowników (takiej jak katalog LDAP) pożądana jest możliwość przechowywania ich także w tej samej bazie. W domyślnej konfiguracji SSH nie posiada integracji z katalogiem LDAP umożliwiającej odczyt kluczy z tego źródła. Istnieje jednak projekt openssh-lpk (OpenSSH LDAP Public Keys), dostarczający łatkę (*patch*), która umożliwia wykonanie takiej integracji.

Niestety pakiety OpenSSH dostępne w repozytoriach Debiana nie posiadają nałożonej tej łatki, tak więc konieczne jest jej ręczne nałożenie i rekompilacja pakietu. Debian dostarcza pakiet OpenSSH w wersji 5.5p1, tak więc najlepszym rozwiązaniem było przygotowanie pakietu z nałożoną łatką w tej samej wersji oraz z takimi samymi opcjami, jakie dostarcza domyślny pakiet.

W celu pobrania źródeł pakietu OpenSSH w takiej formie w jakiej dostarczane są poprzez programistów Debiana można posłuży się poleceniem apt-get source openssh. Po jego wykonaniu w aktualnym katalogu zostanie utworzony podkatalog openssh-5.5p1 zawierający źródła pakietu. Pierwszym koniecznym do wykonania krokiem jest dodanie wpisu informującego o nałożeniu łatki LPK w pliku debian/changelog:

```
openssh (1:5.5p1-7) unstable; urgency=low

  * OpenSSH LPK support

 -- Pawel Szubert <pawel.pbm@gmail.com>  Sat, 3 Sep 2011 19:09:29 +0000
```

Następnie należy pobrać plik łatki i nałożyć go za pomocą polecenia patch -p1 < ./plik.patch. Po jego nałożeniu można przystąpić do zbudowania pakietu za pomocą narzędzia dpkg-buildpackage: dpkg-buildpackage -us -uc.

W wyniku wykonania tego polecenia powstanie kilka pakietów deb. Na komputerze, który ma korzystać z danych LDAP należy dokonać instalacji pakietów openssh-client oraz openssh-server. Po wykonaniu tej operacji konieczne jest ponowne uruchomienie usługi serwera SSH (/etc/init.d/ssh restart).

Aby serwer SSH mógł pobierać dane z katalogu LDAP należy jeszcze dokonać jego odpowiedniej konfiguracji. W pliku konfiguracyjnym serwera (`/etc/ssh/sshd_config`) należy ustawić adres serwera LDAP, lokalizację (DN) użytkowników oraz grup oraz wskazać konto użytkownika za pomocą którego serwer SSH będzie łączył się do katalogu.

```
UseLPK yes
LpkServers          ldap://127.0.0.1
LpkUserDN           ou=users,dc=icis,dc=pcz,dc=pl
LpkGroupDN          ou=groups,dc=icis,dc=pcz,dc=pl
LpkBindDN           cn=admin,dc=icis,dc=pcz,dc=pl
LpkBindPw           haslo
LpkForceTLS         no
LpkSearchTimelimit 3
LpkBindTimelimit   3
```

Są to wszystkie zmiany jakie należy wprowadzić po stronie systemu. Kolejnym krokiem jest wprowadzenie odpowiednich danych, czyli kluczy publicznych do katalogu LDAP. W celu umożliwienia przechowywania kluczy publicznych w katalogu konieczne jest dodanie schematu zawierającego odpowiednie pola do konfiguracji LDAP. Schemat taki jest zawarty w pliku `openssh-lpk_openldap.schema` dostarczanym przez projekt `openssh-lpk`. W pliku znajduje się jedna klasa `ldapPublicKey` dostarczająca jeden atrybut `sshPublicKey`.

Po wprowadzeniu tych wszystkich zmian przy próbie zalogowania użytkownika przy użyciu kluczy SSH, serwer SSH dokona sprawdzenia czy klucz taki nie znajduje się w domyślnych lokalizacjach, a następnie jeśli odpowiedni klucz nie zostanie odnaleziony odwoła się do serwera LDAP.

2.3.3. Logowanie graficzne z innych komputerów

System umożliwia także logowanie użytkowników LDAP na komputerach działających na uczelni. W celu umożliwienia takiego logowania na każdym z komputerów należy dokonać odpowiedniej konfiguracji bibliotek PAM i NSS (zgodnie z instrukcjami z rozdziału 2.3.1).

Dodatkowo, aby na komputerze na którym użytkownik dokonuje logowania jako katalog domowy montowany był jego katalog domowy przechowywany na serwerze, konieczna jest instalacja i odpowiednia konfiguracja modułu PAM `pam-mount`. Jako, że katalog montowany będzie za pośrednictwem protokołu SSH, konieczna jest także instalacja pakietu `sshfs`.

```
apt-get install libpam-mount sshfs
```

Konfiguracja modułu `pam-mount` odbywa się poprzez plik `/etc/security/pam_mount.conf.xml` widoczny na listingu 1.

Listing 1. Plik konfiguracyjny modułu `pam-mount`

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE pam_mount SYSTEM "pam_mount.conf.xml.dtd">
3  <pam_mount>
4  <debug enable="0" />
5  <mntoptions allow="nosuid,nodev,loop,encryption,fsck,nonempty,
       allow_root,allow_other,user" />
6  <mntoptions require="nosuid,nodev" />
7  <fusemount>mount.fuse %(VOLUME) %(MNTPT) -o %(OPTIONS)</
       fusemount>
8  <logout wait="10000" hup="yes" term="yes" kill="yes" />
9  <mkmountpoint enable="1" remove="true" />
10 <volume fstype="fuse" path="sshfs#%(USER)@student.icis.pcz.pl:"
       mountpoint="~" options="user,reconnect,nonempty" ssh="1"/>
11 </pam_mount>
```

W linii 7 pliku konfiguracyjnego zdefiniowane jest polecenie jakie ma zostać wywołane przy montowaniu. Szczegóły dotyczące parametrów montowanego systemu plików znajdują się w linii 10. Kolejno wskazany jest typ systemu plików (`fuse`), ścieżka jaka ma zostać zamontowana (`sshfs#%(USER)@student.icis.pcz.pl:`, w miejscu zmiennej `%(USER)` wstawiona zostanie nazwa użytkownika, który się loguje), katalog montowania oraz dodatkowe opcje. Parametr `ssh="1"` wskazuje, że montowanie odbywa się z wykorzystaniem protokołu SSH, więc `pam-mount` musi skorzystać ze specjalnego narzędzia, aby

przekazać hasło do polecenia `ssh`, gdyż nie akceptuje ono haseł podawanych na standardowe wejście.

Opcja `mkmountpoint` znajdująca się w linii 9 wskazuje modułowi, aby w przypadku w którym katalog montowania nie istnieje automatycznie go utworzył, a po odmontowaniu zdalnego systemu plików usunął.

Aby odmontowywanie systemu plików przebiegało poprawnie przy wylogowywaniu się poprzez GDM3 należy zmodyfikować skrypt `/etc/gdm3/PostSession/Default` w poniższy sposób:

```
#!/bin/sh
fusermount -u -z ${HOME}
exit 0
```

Konieczne jest także dodanie linii zmieniających uprawnienia do plików FUSE w pliku `/etc/rc.local`:

```
chmod o+x /usr/bin/fusermount
chmod o+rw /dev/fuse
```

Ostatnim krokiem jest dodanie klucza SSH zdalnego komputera do pliku `/etc/ssh/ssh_known_hosts`. Po wykonaniu tych zmian możliwe jest już przeprowadzanie logowania z automatycznym montowaniem katalogu domowego. Aby uniknąć jednak komunikatu na temat błędu odczytu pliku `.ICEauthority`, konieczna jest zmiana jego lokalizacji. Zmiany takiej można dokonać poprzez wyeksportowanie zmiennej systemowej `ICEAUTHORITY`. Aby zmiana taka była automatycznie stosowana dla wszystkich użytkowników na serwerze w katalogu `/etc/skel/` stworzony został plik `.gnomerc` z następującą treścią:

```
mkdir "/tmp/.ICE-${USER}"
export ICEAUTHORITY="/tmp/.ICE-${USER}/.ICEauthority"
```

Podczas tworzenia użytkownika na serwerze zawartość katalogu `/etc/skel/` kopiowana jest do katalogu domowego użytkownika. Plik `.gnomerc` wykonywany jest przy uruchamianiu sesji GNOME, dzięki czemu zostaną wykonane zawarte w nim polecenia, czyli w katalogu `/tmp/` utworzony zostanie katalog `.ICE-nawa_użytkownika` i w nim utworzony zostanie plik `.ICEauthority`.

2.3.4. GIT

W celu udostępnienia użytkownikom repozytorium systemu kontroli wersji wybrany został pakiet GIT. Po instalacji oprogramowania GIT na serwerze użytkownik może z niego skorzystać uzyskując dostęp do repozytorium za pomocą protokołu SSH. Oprogramowanie to nie posiada jednak możliwości współdzielenia repozytoriów, co było jednym z założeń projektu. W celu uzyskania takiej funkcjonalności zdecydowano się na opracowanie skryptu, który na podstawie konfiguracji udostępniania repozytoriów przechowywanej w katalogu LDAP przyznałby dostęp danemu użytkownikowi do wybranego repozytorium lub takiego dostępu odmówił.

W celu umożliwienia użytkownikowi skorzystania z repozytorium należy do wpisu w katalogu LDAP użytkownika udostępniającego repozytorium dodać nowy klucz publiczny - klucz użytkownika chcącego skorzystać z repozytorium. Aby użytkownik taki miał dostęp jedynie do narzędzia GIT, a nie mógł dokonać za pomocą tego klucza zwykłego logowania do systemu, wykorzystany został mechanizm SSH pozwalający na określenie we wpisie klucza publicznego polecenia, jakie po logowaniu za jego pomocą zostanie wywołane.

Przed wpisem klucza publicznego została dodana opcja `command="/var/student/gitserver.py username"`, gdzie `username` jest nazwą użytkownika chcącego skorzystać z repozytorium, a tym samym właściciela klucza publicznego. Podczas logowania za pomocą tak opisanego klucza zostanie wywołany skrypt `/var/student/gitserver.py` i jako argument do niego zostanie przekazana nazwa użytkownika.

Dane dotyczące repozytoriów jakie są udostępniane oraz uprawnień do nich także przechowywane są w katalogu LDAP. W tym celu pod wpisem dla użytkownika udostępniającego repozytorium tworzony jest nowy wpis klasy `organizationalUnit ou=git`, mający na celu grupowanie wpisów dotyczących konkretnych repozytoriów.

Repozytoria przechowywane są w kolejnych wpisach klasy `repository`, która została przygotowana na potrzeby tego projektu. Klasa ta dostarcza trzy atrybuty:

- `repo` (lub `repositoryName`),
- `userRO`,
- `userRW`.

Atrybut `repo` jest atrybutem obowiązkowym (opcja `MUST`), pozostałe są opcjonalne (opcja `MAY`). Atrybuty `userRO` oraz `userRW` mogą występować kilka razy z różnymi wartościami.

Określają one osoby, które mają dostęp z uprawnieniami tylko do odczytu oraz do zapisu do danego repozytorium, którego dotyczy dany wpis, a które określone jest za pomocą atrybutu `repo`.

Skrypt `gitserver.py` nawiązuje połączenie z katalogiem LDAP i sprawdza czy dla repozytorium do którego użytkownik się łączy został zdefiniowany wpis klasy `repository` i czy użytkownik ten został wymieniony a atrybucie `userRO` lub `userRW`.

W czasie wykonania skryptu pierwszym krokiem jest pobranie zmiennej środowiskowej `SSH_ORIGINAL_COMMAND`. Zmienna ta zawiera oryginalne polecenie jakie zostało przekazane poprzez klienta SSH.

W przypadku nawiązywania połączenia przez klienta git zmienna ta zawiera jedno z poleceń `git-upload-pack` lub `git-receive-pack`, a następnie ścieżkę do repozytorium.

Jeśli polecenie zawarte w zmiennej jest inne połączenie musi być zerwane, gdyż oznacza to, że nie zostało ono nawiązane przez klienta git:

Listing 2. Konfiguracja backendów autoryzacyjnych

```
1 valid_commands = ('git-receive-pack', 'git-upload-pack',)
2 if command not in valid_commands:
3     exit()
```

Kolejnym krokiem weryfikacji jest sprawdzenie jakie uprawnienia ma użytkownik do danego repozytorium. W tym celu wywoływana jest funkcja `check_permissions` z argumentami `owner_name`, `guest_name` oraz `repository`. Pierwszy z parametrów określa nazwę właściciela repozytorium i jest pobierany ze zmiennej środowiskowej `USER`. Kolejny to nazwa użytkownika, który chce uzyskać dostęp do repozytorium. Nazwa ta jest zapisana w katalogu LDAP wraz z kluczem publicznym i jest przekazywana jako argument do skryptu `gitserver.py`. Wartość zmiennej `repository` pobierana jest z drugiej części zmiennej `SSH_ORIGINAL_COMMAND`.

Funkcja `check_persmissions` nawiązuje połączenie z serwerem LDAP, przeszukuje DN `repo=repository,ou=git,uid=owner_name,ou=users,dc=icis,dc=pcz,dc=pl` i pobiera z niego wpisy `userRO` i `userRW`. Jeśli nazwa użytkownika zostanie odnaleziona w atrybucie `userRO` funkcja zwraca wartość 1, jeśli nazwa ta będzie w atrybucie `userRW` zostanie zwrócona wartość 2. W przypadku nie odnalezienia nazwy użytkownika w żadnym z tych atrybutów zwrócone zostanie 0.

W przypadku kiedy zostanie zwrócona wartość 0 połączenie jest zakańczane. W pozostałych przypadkach następuje sprawdzenie czy polecenie przekazane w zmiennej `SSH_ORIGINAL_COMMAND` odpowiada poziomowi uprawnień danego użytkownika do repozytorium. W przypadku polecenia `git-receive-pack` (czyli zapisu do repozytorium) wymagany poziom uprawnień to 2, natomiast w przypadku `git-upload-pack`, które odpowiada za odczyt danych z repozytorium poziom to 1. Jeśli wartości te nie zgadzają się następuje rozłączenie użytkownika. Jeśli natomiast polecenie, które użytkownik chce wykonać zgodne jest z jego uprawnieniami następuje jego wykonanie.

3. Portal

W celu uproszczenia administracji danymi przechowywanymi w katalogu stworzony został portal umożliwiający użytkownikom rejestrację w serwisie, zarządzanie uprawnieniami do repozytoriów oraz wykonywanie podstawowych czynności administracyjnych w przypadku użytkownika z uprawnieniami administratora.

3.1. Wykorzystane biblioteki

Portal wykonany został w języku Python przy użyciu frameworka Django. Dodatkowo wykorzystane zostały takie aplikacje Django jak `django_auth_ldap` i `django-ldapdb`. Dodatkowo do stworzenia warstwy wizualnej portalu wykorzystana została biblioteka CSS i Javascript Twitter Boostrap.

3.1.1. Django

Django jest to framework do tworzenia aplikacji internetowych napisany w języku Python. Opiera się on na zbliżonym do MVC (*Model-View-Controller, Model-Widok-Kontroler*) wzorcu projektowym nazwanym MVT (*Model-View-Template, Model-Widok-Szablon*).

Model definiuje struktury danych z jakich korzystać będzie aplikacja. Utworzenie modelu polega na zdefiniowaniu klas dziedziczących po klasie `models.Model`. Jednym ze składników Django jest także system ORM (*Object-Relational Mapping, Mapowanie Obiektowo-Relacyjne*), który odpowiada za mapowanie zdefiniowanego modelu na strukturę bazy danych. ORM dostarcza także funkcje umożliwiające pobieranie i dodawanie danych do bazy danych bez konieczności pisania zapytań SQL. Wykorzystanie mechanizmu ORM pozwala na stworzenie aplikacji niezależnej od serwera bazodanowego. Odpowiednie moduły Django tłumaczą kod programu na zapytania SQL dostosowane do wykorzystywanego silnika bazodanowego. W chwili obecnej ORM Django (wersja 1.4) wspiera następujące silniki bazodanowe[4]:

- PostgreSQL
- MySQL
- SQLite
- Oracle

Widok jest odpowiedzialny za pobieranie danych od użytkownika, przetwarzanie ich i odsyłanie do użytkownika. Kiedy użytkownik wchodzi na konkretną stronę serwisu internetowego mechanizm Django o nazwie `URL dispatcher` mapuje wywołany adres na konkretną funkcję widoku. Do widoku przesyłane jest całe zapytanie użytkownika w postaci obowiązkowego parametru `request` zawierającego miedzy innymi parametry GET i POST. Po przetworzeniu takiego żądania funkcja widoku zwraca obiekt klasy `HttpResponse`, zawierający odpowiedź.

Szablony Django odpowiadają za generowanie strony internetowej. Szablon jest to zazwyczaj plik HTML zawierające specjalne znaczniki Django pozwalające na wstawianie do kodu HTML zmiennych języka Python. Język szablonów Django zawiera także podstawowe instrukcje sterujące.

3.1.2. `django_auth_ldap`

Django dostarcza mechanizm autoryzacji użytkowników z zaawansowanym systemem grup i uprawnień[8]. Domyślnie mechanizm ten jest obsługiwany przez backend ModelBackend, dla którego źródłem danych na temat użytkowników, grup i uprawnień są dane przechowywane w modelu `User`, który przechowywany jest w domyślnej bazie danych. Aplikacja `django_auth_ldap` dostarcza backend autoryzacji dla którego źródłem danych jest katalog LDAP [3].

Aby włączyć wykorzystanie tej aplikacji należy pliki aplikacji umieścić w jednym z katalogów zdefiniowanych w zmiennej systemowej `PYTHONPATH`. Następnie należy zmodyfikować plik ustawień aplikacji (`settings.py`), dodając ten backend jako kolejny w zmiennej `AUTHENTICATION_BACKENDS`:

Listing 3. Konfiguracja backendów autoryzacyjnych

```
1 AUTHENTICATION_BACKENDS = (
2     'django.contrib.auth.backends.ModelBackend',
3     'django_auth_ldap.backend.LDAPBackend',
```

```
4 )
```

Konfiguracji backendu dokonuje się poprzez ustawienie kilku zmiennych w tym samym pliku konfiguracyjnym:

Listing 4. Konfiguracja backendów autoryzacyjnych

```
1 LDAP_SUFFIX = 'dc=icis,dc=pcz,dc=pl'
2 AUTH_LDAP_SERVER_URI = 'ldap://127.0.0.1/'
3 AUTH_LDAP_BIND_DN = ""
4 AUTH_LDAP_BIND_PASSWORD = ""
5 AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users," + LDAP_SUFFIX,
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
```

W pierwszej ze zmiennych określamy adres serwera LDAP. Ustawienie pustej zmiennej `bind dn` oraz `bind password` oznacza, że będziemy dokonywali bindowania anonimowego. W ostatniej zmiennej określamy jak ma wyglądać zapytanie przeszukujące drzewo LDAP w poszukiwaniu kont użytkowników. W tym przypadku konta przechowywane są w jednostce organizacyjnej `ou=users` i definiowane są za pomocą atrybutu `uid`.

3.1.3. `djnago-ldapdb`

Aplikacja `django-ldapdb` rozszerza możliwości mechanizmu ORM o zarządzanie katalogiem LDAP w taki sam sposób w jaki zarządza on innymi bazami danych, czyli z wykorzystaniem mechanizmu modeli. Model korzystający z bazy LDAP musi dziedziczyć po klasie `ldapdb.models.Model`.

Oprócz standardowych zmiennych, które zostaną zmapowane na bazę danych model korzystający z biblioteki `django-ldapdb` musi posiadać ustawione dwie zmienne: `base_dn` oraz `object_classes`.

Pierwsza z nich określa DN danego wpisu, czyli pozycję w katalogu LDAP gdzie obiekty danej klasy będą się znajdowały. Druga zmienna odpowiada za ustawienie obiektom odpowiednich atrybutów `objectClass`. Jeśli w katalogu znajdują się już jakieś obiekty i chcemy aby były one traktowane jako obiekty danej klasy muszą one mieć ustawione identyczne wartości `objectClass` jak te podane w zmiennej `object_class`.

Konfiguracji bazy danych dokonuje się w pliku `settings.py`:

Listing 5. Konfiguracja biblioteki `django-ldapdb`

```
1 DATABASES = {
2       'ldap': {
3             'ENGINE': 'ldapdb.backends.ldap',
4             'NAME': 'ldap://127.0.0.1/',
5             'USER': 'cn=admin,dc=icis,dc=pcz,dc=pl',
6             'PASSWORD': 'HASŁO',
7       }
8  }
9 DATABASE_ROUTERS = ['ldapdb.router.Router']
```

Po dodaniu bazy i utworzeniu modelu każda funkcja odwołująca się do niego będzie działała już na danych z katalogu LDAP.

3.1.4. Twitter Bootstrap

Biblioteka Twitter Bootstrap dostarcza zestaw klas CSS oraz funkcji Javascript do tworzenia komponentów interfejsu użytkownika takich jak przyciski, okna czy menu.

Podstawowym składnikiem biblioteki jest dwunastokolumnowa siatka do tworzenia układu strony. Dzięki wykorzystaniu klas CSS `row` i `spanX` możliwe jest utworzenie wierszy i kolumn na stronie z elementów `div`. Upraszcza to tworzenie struktury strony oraz podział jej na części takie jak nagłówek, stopka, cześć główna czy menu.

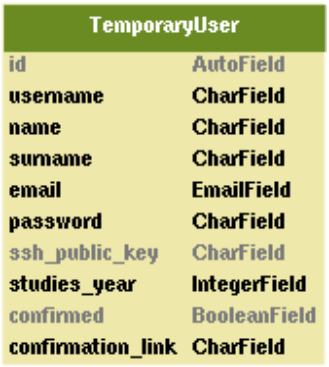
Kolejna część biblioteki odpowiada za odpowiednie ostylowanie elementów HTML takich jak odnośniki, tabele czy formularze.

Ostatnim komponentem jest zestaw skryptów Javascript wykorzystujących bibliotekę jQuery i pozwalających na proste tworzenie dynamicznych elementów takich jak okienka komunikatów, rozwijane menu czy alerty.

3.2. Model

Na potrzeby aplikacji zostały stworzone dwie klasy reprezentujące użytkownika oraz klasa reprezentująca grupę.

Pierwszą z klas jest `TemporaryUser`, czyli model przechowywany w bazie SQLite mający na celu przechowywanie danych użytkownika zanim zostaną one zaakceptowane przez administratora i przeniesione do katalogu LDAP. Klasa ta zawiera atrybuty przedstawione na rysunku 5.



Rysunek 5. Klasa `TemporaryUser`

Kolejną klasą jest `LdapUser`, która reprezentuje użytkownika po zapisaniu go do bazy danych. Atrybuty zdefiniowane w tej klasie zaprezentowane są na rysunku 6. Dodatkowo zmienne `base_dn` oraz `object_classes` zostały ustawione w sposób zaprezentowany na listingu 6.

Listing 6. Zmienne `base_dn` oraz `object_classes` w klasie `LdapUser`

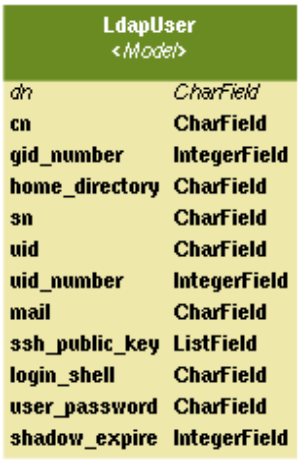
```
1 base_dn = "ou=users," + LDAP_SUFFIX
2 object_classes = ['top', 'posixAccount', 'person', '
    organizationalPerson', 'inetOrgPerson', 'ldapPublicKey', '
    shadowAccount']
```

Klasa reprezentująca grupy, czyli `LdapGroup`, została zdefiniowana w sposób widoczny na rysunku 7, natomiast sposób ustawienia zmiennych na listingu 7.

Listing 7. Zmienne `base_dn` oraz `object_classes` w klasie `LdapGroup`

```
1 base_dn = "ou=groups," + LDAP_SUFFIX
2 object_classes = ['posixGroup', 'top']
```

Dodatkowo oprócz tych modeli, które zdefiniowane są na stałe, w pliku `models.py` znajdują się dwie funkcje, które w locie generują klasy. Klasy te tworzą obiekty przechowujące wpisy dla repozytoriów oraz dla grupowania repozytoriów. Mechanizm dynamicznego



Rysunek 6. Klasa `LdapUser`

Rysunek 7. Klasa `LdapGroup`

generowanie klas zastosowany został ze względu na konieczność wypełnienia zmiennych wewnątrz klasy na podstawie tego dla jakiego użytkownika tworzymy model. W takiej sytuacji nie można było skorzystać z mechanizmu konstruktorów, gdyż modele Django często wykorzystują metody klas, które nie są wywoływane na instancji klasy, lecz na niej samej.

Funkcje te to `repo_entry()` oraz `git_entry()`. Obie z nich jako parametr przyjmują nazwę użytkownika, a zwracają kolejno klasę `RepoEntry_username` i `GitEntry_username`. Kod funkcji `git_entry` przedstawiony został na listingu 8.

Listing 8. Funkcja `GitEntry()`

```
1 def git_entry(username):
2     base_dn = "uid=" + username +",ou=users,"+ LDAP_SUFFIX
3     object_classes = ['top', 'organizationalUnit']
4
5     ou = LDAPCharField(db_column='ou', max_length=200,
          primary_key=True)
```

```
6
7       class Meta:
8           app_label = "label"
9           managed = False
10          verbose_name = 'GitEntry'
11          verbose_name_plural = 'GitEntry'
12      attrs = {}
13      attrs['__module__'] = 'management.models'
14      attrs['Meta'] = Meta
15      attrs['base_dn'] = base_dn
16      attrs['username'] = username
17      attrs['object_classes'] = object_classes
18      attrs['ou'] = ou
19
20      return type('GitEntry_' + username.encode('ascii','ignore'),
            (ldapdb.models.Model,), attrs)
```

W linii 2 widzimy inicjalizację zmiennej `base_dn` na postawie nazwy użytkownika. Następnie tworzony jest atrybut `ou` typu `LDAPCharField`. Parametry jakie są wypełniane przy tworzeniu tego atrybutu oznaczają kolejno:

- atrybut ten będzie przechowywany jako atrybut LDAP o nazwie `ou`,
- maksymalna długość tekstu to 200 znaków,
- atrybut ten jest kluczem publicznym, czyli w katalogu LDAP zostanie wykorzystany do stworzenia DN wpisu.

W linii 7 utworzona jest subklasa `Meta`[6]. Klasa ta definiuje ustawienia danego modelu. W tym przypadku określa ona nazwę aplikacji w jakiej model jest definiowany, to czy Django ma zajmować się zarządzaniem strukturą bazy danych oraz nazwę jaką dany model będzie miał w panelu administracyjnym.

W kolejnych linijkach tworzony jest słownik `attrs` i wypełniane są jego pola. Słownik ten posłuży do stworzenia atrybutów funkcji.

W ostatniej linii zwracany jest wynik wywołania funkcji `type`[17] pochodzącej ze standardowej biblioteki Pythona. Funkcja ta przyjmuje trzy atrybuty: nazwę, klasę nadrzędną

oraz słownik zawierający atrybuty i zwraca klasę utworzoną na podstawie tych danych. Jest ona dynamiczną wersją wyrażenia `class`.

3.3. Rejestracja i aktywacja użytkownika

3.3.1. Rejestracja

Użytkownik, aby otrzymać konto na serwerze musi dokonać rejestracji za pośrednictwem portalu. Podczas dokonywania rejestracji wprowadzane są następujące dane:

- nazwa użytkownika,
- imię,
- nazwisko,
- adres e-mail,
- hasło,
- klucz publiczny SSH,
- rok studiów.

Formularz rejestracji generowany jest na podstawie klasy `RegistrationForm`, dziedziczącej po dostarczanej przez Django klasie `forms.Form`. `RegistrationForm` posiada atrybuty definiujące poszczególne pola formularza. Przykładowo pole "Nazwisko" generowane jest na podstawie kodu przedstawionego na listingu 9. Pole to jest obiektem klasy `forms.CharField`. Parametry przekazane do konstruktora określają maksymalną ilość tekstu jaką dane pole może pomieścić oraz etykietę pola.

Listing 9. Definicja pola nazwisko

```
surname = forms.CharField(max_length=200, label='Nazwisko')
```

Kolejne pola formularza zostały zdefiniowane przy wykorzystaniu takich klas jak `EmailField`, `BooleadField` oraz `ChoiceField`. To jakiej klasy jest dane pole ma wpływ na to za pomocą jakiego elementu HTML zostanie ono wyrenderowane oraz to czy i w jaki sposób zostanie wykonana walidacja danych takiego pola. Przykładowo pole klasy `EmailField` zostanie wyświetlone jako element HTML `<input type=text>`, a klasy `BooleanField` jako `<input type=checkbox>`. Dodatkowo na polu `EmailField` zostanie przeprowadzone sprawdzenie czy wprowadzona wartość jest adresem e-mail.

W klasie `RegistrationForm` zdefiniowane zostały także funkcje `clean_NAZWA_POLA`[5]. Funkcje te pozwalają na stworzenie dodatkowych reguł walidacji pola. Są one wywoływane na obiekcie pola dla, którego zostały zdefiniowane, a po przetworzeniu lub dokonaniu walidacji zwracają wartość tego pola.

Przykładowo dla pola zawierającego klucz publiczny SSH zdefiniowana została funkcja `clean_ssh_public_key()` widoczna na listingu 10.

Listing 10. Funkcja `confirm()`

```
1  def clean_ssh_public_key(self):
2      ssh_public_key = self.cleaned_data.get('ssh_public_key')
3      try:
4          key_type, data, _ = ssh_public_key.split()
5          data = base64.decodestring(data)
6          int_len = 4
7          str_len = struct.unpack('>I', data[:int_len])[0]
8          if not data[int_len:int_len+str_len] == key_type:
9              self._errors["ssh_public_key"] = self.error_class(["
                   Nieprawidłowy klucz."])
10     except:
11         self._errors["ssh_public_key"] = self.error_class(["
               Nieprawidłowy klucz."])
12     return ssh_public_key
```

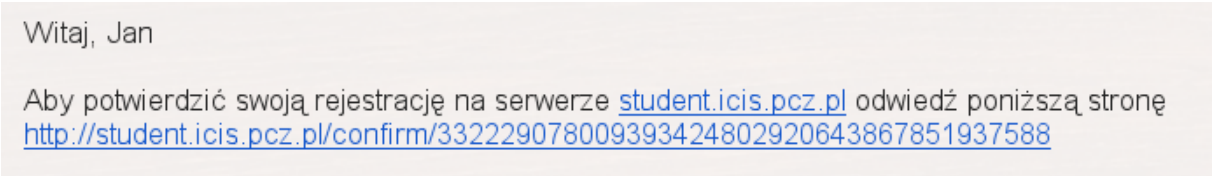
W drugiej linii funkcji pobieramy wartość pola a następnie w klauzuli `try` dzielimy wartość klucza na trzy części. Na listingu 11 widoczny jest przykładowy klucz publiczny przed podziałem i po podziale. Zmienna `key_type` zawiera typ klucza, `data` jego zawartość, a `_` komentarz, który jest ignorowany. Następnie w piątej linii dane klucza są dekodowane za pomocą funkcji `base64.decodestring()` i następuje sprawdzenie czy typ klucza zawarty w zmiennej `key_type` (czyli w pierwszym fragmencie klucza) zgodny jest z tym zakodowanym w danych klucza. Jeśli tak to klucz uznawany jest za prawidłowy. W przeciwnym przypadku wyświetlany jest błąd.

Listing 11. Przykładowy klucz publiczny SSH przed podziałem i po podziale w funkcji `clean_ssh_public_key()`

```
ssh_public_key = "ssh-dss AAAAB3NzaC1kc3MAAACBAIvRh9ALPHvkqXqCB
vO2PzKwBiLTCP40bevM4y1bOcdSFzF0BqRh16kkTyfCS6xNzEwZVjf0u29ig3tS
1zAZO7v40AM03UDeDPg3F+Z43Q2dSVgeURuA6f9xJKu8+OhbQasl4WZ6Qwi7Rsk
Bi9edIABmkBXuh7bLxzhccMWTkwIjAAAAFQDWiyOCUA1QdvHmX2CU+rwOLk5+Dw
AAAIBuiwkBAp7SIyZnRuMQG6+YJ/dy+i2aTocJ2ypLNfpCBrrwN1u77IY3iYrE6
yqLg1tOULCbe7sFJeiXzocaBEfbC71LCaDJ+ESsXB8Sa1aTq9vMH3bAuZqhc//C
XHvw17NWi1RcPsTwiFiODQoRlawd+Oryi/PX9wZWffGb1PkswwAAAIA0BDFUfHj
XMNPu227I3mR8/Amue91zkIbEIR+QEyesXYgqgN57C3sPZ7vohkqOCgTCfRqMSY
Bh+nXW6mxis9TSwg/lm22lvZ3rRuNr5aD500co25OFgagaACn0NGVmfOi+FJMFx
qki4xHcVivIoZN49PVfM8BbVr+kSORVX9yHbg== pbm@tauri"

key_type = "ssh-dss"

data = "AAAAB3NzaC1kc3MAAACBAIvRh9ALPHvkqXqCBvO2PzKwBiLTCP40bev
M4y1bOcdSFzF0BqRh16kkTyfCS6xNzEwZVjf0u29ig3tS1zAZO7v40AM03UDeDP
g3F+Z43Q2dSVgeURuA6f9xJKu8+OhbQasl4WZ6Qwi7RskBi9edIABmkBXuh7bLx
zhccMWTkwIjAAAAFQDWiyOCUA1QdvHmX2CU+rwOLk5+DwAAAIBuiwkBAp7SIyZn
RuMQG6+YJ/dy+i2aTocJ2ypLNfpCBrrwN1u77IY3iYrE6yqLg1tOULCbe7sFJei
XzocaBEfbC71LCaDJ+ESsXB8Sa1aTq9vMH3bAuZqhc//CXHvw17NWi1RcPsTwiF
iODQoRlawd+Oryi/PX9wZWffGb1PkswwAAAIA0BDFUfHjXMNPu227I3mR8/Amue
91zkIbEIR+QEyesXYgqgN57C3sPZ7vohkqOCgTCfRqMSYBh+nXW6mxis9TSwg/l
m22lvZ3rRuNr5aD500co25OFgagaACn0NGVmfOi+FJMFxqki4xHcVivIoZN49PV
fM8BbVr+kSORVX9yHbg=="

_ = "pbm@tauri"
```

Na postawie wprowadzonych danych w widoku `registration()` tworzony jest nowy obiekt klasy `TemporaryUser`. Ważnym elementem jest tu funkcja `strip_polish_letters()`, która odpowiada za oczyszczenie nazwy użytkownika (loginu) z polskich znaków. Dodatkowo w obiekcie tym zmienna `confirmed` ustawiana jest na wartość `False` oraz za pomocą

funkcji `generate_confirmation_link()` generowany jest losowy adres odnośnika potwierdzającego. Tak utworzony obiekt zapisywany jest w bazie danych SQLite, a następnie na adres e-mail wysyłana jest wiadomość z linkiem potwierdzającym (rys. 8).



Rysunek 8. Wiadomość e-mail z linkiem potwierdzającym

W momencie potwierdzenia przez użytkownika adresu e-mail, poprzez kliknięcie na odnośnik zawarty w wiadomości wywoływany jest widok `confirm()` do, którego jako parametr przekazywany jest klucz potwierdzający będący ostatnią częścią odnośnika. Na tej podstawie w bazie odnajdywany jest użytkownik, którego adres jest potwierdzany (listing 12, linia 4). W przypadku jeśli dane konto zostało już potwierdzone użytkownik jest o tym informowany odpowiednim komunikatem. Jeśli konto nie jest jeszcze potwierdzone zmienna `confirmed` ustawiana jest na wartość `True`, a użytkownikowi wyświetlany jest komunikat informujący, że właśnie dokonał potwierdzenia konta.

W przypadku kiedy funkcja `TemporaryUser.objects.get()` odnajdująca użytkownika w bazie zwróci wyjątek `ObjectDoesNotExist`, oznaczający, że użytkownik o takim linku potwierdzającym nie został znaleziony w bazie wyświetlany jest komunikat błędu.

Listing 12. Funkcja `confirm()`

```
1 def confirm(request, activation_key):
2     form = LogInForm()
3     try:
4         user = TemporaryUser.objects.get(confirmation_link=
              activation_key)
5         if user.confirmed:
6             return render_to_response('invitation.html', {'
                  confirmed': 1, 'form': form}, context_instance=
                  RequestContext(request))
7         user.confirmed = True
```

```
8           user.save()
9           return render_to_response('invitation.html', {'
                confirmed': 2, 'form': form}, context_instance=
                RequestContext(request))
10      except ObjectDoesNotExist:
11          return render_to_response('invitation.html', {'
                confirmed': 3, 'form': form}, context_instance=
                RequestContext(request))
```

Wyświetlanie komunikatów informacyjnych dotyczących potwierdzenia adresu zrealizowane jest na postawie parametru `confirmed`, jaki przekazywany jest do szablony strony głównej `invitation.html`. Parametr ten może przyjmować trzy wartości:

- 1 - kiedy konto zostało już wcześniej potwierdzone,
- 2 - kiedy konto zostało teraz potwierdzone,
- 3 - kiedy dany link aktywacyjny nie został odnaleziony w bazie,
- 4 - wykorzystywany przy rejestracji konta, w widoku `registration()`.

Przyczyną nieodnalezienia linka w bazie może być na przykład jego uszkodzenie przez program pocztowy, niepełne skopiowanie do przeglądarki przez użytkownika lub też usunięcie lub aktywowanie konta tymczasowego przez administratora portalu.

W kodzie szablonu `invitation.html` (listing 13) zawarty jest warunek sprawdzający czy parametr `confirmed` został przekazany (linia 1). Jeśli tak następuje wygenerowanie obiektu `div` z klasami `modal` oraz `fade` pochodzącymi z biblioteki Twitter Bootstrap. Wewnątrz tego obiektu zawarte są kolejne elementy `div` tworzące nagłówek, treść oraz stopkę okna. W części zawierającej właściwą treść okna znajdują się kolejne instrukcje `if` sprawdzające wartość zmiennej `confirmed` i wypisujące odpowiednią treść komunikatu.

Listing 13. Wyświetlanie komunikatu potwierdzenia

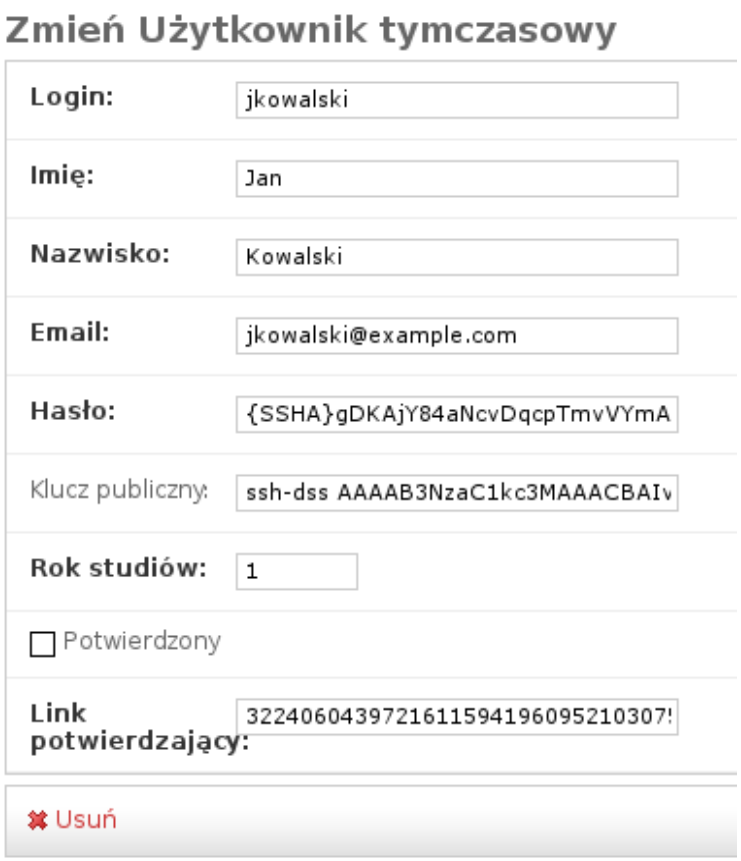
```
1 {% if confirmed %}
2     <div class="modal fade" id="activationModal">
3     <div class="modal-header">
4     <a class="close" data-dismiss="modal">x</a>
5     <h3>Potwierdzenie konta</h3>
```

```
6       </div>
7       <div class="modal-body">
8       <p>
9         {% if confirmed == 1 %}To konto zostało już potwierdzone.
              Prosimy poczekać na jego aktywację przez administratora
              . {% endif %}
10        {% if confirmed == 2 %}Dziękujemy za potwierdzenie konta.
              Prosimy poczekać na jego aktywację przez administratora
              . {% endif %}
11        {% if confirmed == 3 %}Odnośnik potwierdzający
              nieprawidłowy. Być może konto zostało już aktywowane.
              {% endif %}
12        {% if confirmed == 4 %}Dziękujemy za rejestrację. Na
              podany adres mailowy został przesłany odnośnik
              potwierdzający. {% endif %}
13
14      </p>
15      </div>
16      <div class="modal-footer">
17      <a href="#" class="btn" id="close-modal-btn">Zamknij</a>
18      </div>
19      </div>
20  {% endif %}
```

3.3.2. Aktywacja

Dane użytkowników tymczasowych można przeglądać poprzez Panel administratora (rys. 9).

Aby wybrane modele były dostępne w panelu administratora należy je w nim zarejestrować. Rejestracja taka odbywa się poprzez wywołanie w pliku `admin.py` funkcji `admin.site.register()`. Funkcja ta przyjmuje dwa argumenty. Pierwszym z nich jest nazwa modelu, który chcemy zarejestrować w panelu administracyjnym, a drugi to nazwa

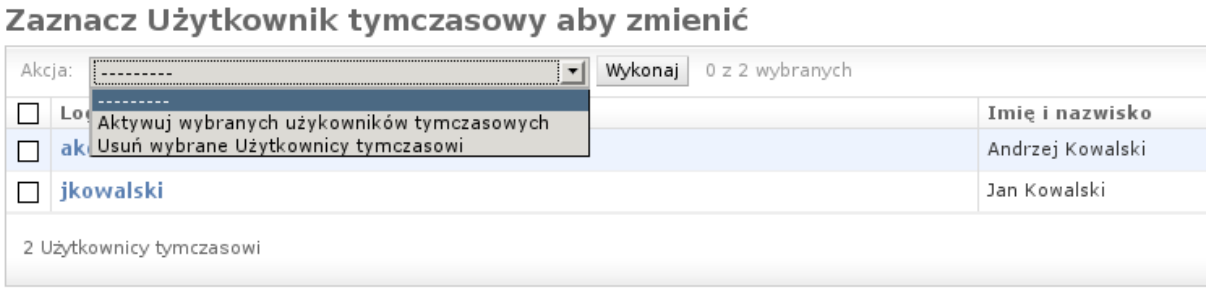


Rysunek 9. Dane użytkownika tymczasowego w panelu administracyjnym

klasy, które definiuje parametry wyświetlania takiego modelu. Wewnątrz klasy tej poprzez odpowiednie zmienne można na przykład wykluczyć wyświetlanie pewnych pól.

Również przez Panel administratora dokonuje się aktywacji konta użytkownika (rys. 10). W celu dodania opcji aktywacji do zestawu standardowych akcji Panelu administracyjnego w Klasie `TemporaryUserAdmin` zdefiniowanej w pliku `admin.py` stworzona została funkcja `activate()`. Funkcja ta jako parametry przyjmuje `request` (jak w przypadku widoku) oraz obiekt `queryset`, zawierający listę obiektów zaznaczonych na liście.

Pierwszym krokiem jest sprawdzenie czy w parametrach przesłanych jako POST znajduje się wartość `apply_update`. Jeśli tak, oznacza to, że należy dokonać aktywacji wszystkich dostępnych kont znajdujących się w parametrze `queryset`. Jeśli parametr ten nie występuje oznacza to, że przechodzimy do dalszej części kodu. Za sekcją `if` następuje rozbicie obiektu `queryset` na dwie zmienne - `queryset_without_confirmation` oraz `queryset_with_confirmation`. W pierwszej zmiennej znajdują się wszyscy użytkownicy



Rysunek 10. Aktywacja konta

tymczasowi zaznaczeni na liście, którzy nie dokonali potwierdzenia adresu e-mail, natomiast w drugiej użytkownicy, który potwierdzili adres.

Dla obiektów zawartych w zmiennej `queryset_with_confirmation` tworzony jest obiekt `LdapUser`. Do konstruktora przekazywane są kolejno następujące parametry:

- `cn` - imię i nazwisko,
- `gid_number` - numer wygenerowany w funkcji `generate_uid_number()` i zapisany w zmiennej `newUid`
- `sn` - nazwisko,
- `uid` - nazwa użytkownika,
- `uid_number` - numer zawarty w zmiennej `netUid`,
- `ssh_public_key` - klucz publiczny, jako jednoelementowa lista,
- `user_password` - hasło,
- `login_shell` - powłoka `/bin/bash`,
- `mail` - adres e-mail,
- `shadow_expire` - liczba dni do wygaśnięcia konta wygenerowana w funkcji `calculate_account_expiration()`.

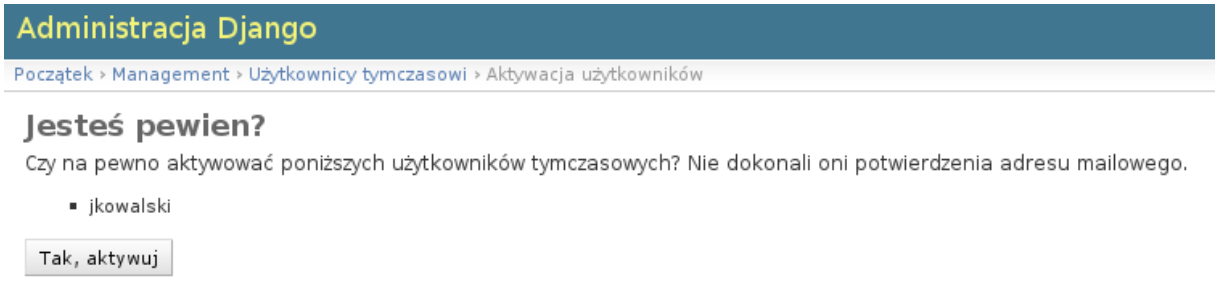
Funkcja `generate_uid_number()` służąca do generowania numerów UID i GID zdefiniowana jest w pliku `misc.py` (listing 14). W funkcji tej wykonywane jest zapytanie do katalogu LDAP w celu odnalezienia najwyższego numeru UID jakie w chwili obecnej znajduje się w katalogu. Następnie wartość ta jest inkrementowana i zwracana jako nowy numer UID. W przypadku nieodnalezienia w katalogu żadnego wpisu z numerem UID przyjmowana jest wartość określona w pliku `settings.py` w zmiennej `INITIAL_UID`.

Listing 14. Funkcja generująca numer UID

```
1 def generate_uid_number():
2     try:
3         uid = LdapUser.objects.order_by('-uid_number')[0].
              uid_number + 1
4     except IndexError:
5         uid = INITIAL_UID
6     return uid
```

W funkcji `activate()` tworzony jest również obiekt na podstawie modelu `LdapGroup` reprezentujący grupę.

W przypadku użytkowników bez potwierdzonych kont mailowych, czyli tych znajdujących się wewnątrz zmiennej `queryset_without_confirmation` za pośrednictwem szablonu `activate_confirmation.html` wyświetlana jest prośba o potwierdzenie aktywacji (rys. 11). Jeśli aktywacja zostanie potwierdzona przez administratora wywoływana jest ta sama funkcja, lecz tym razem wśród parametrów POST znajduje się ustawiona wartość `apply_update`, przez co wykonywana jest zawartość pierwszej klauzuli `if`, czyli aktywacja wszystkich pozostałych kont.



Rysunek 11. Prośba o potwierdzenie aktywacji

3.3.3. Wysyłanie powiadomień mailowych

Aby możliwe było wysyłanie powiadomień mailowych wykorzystany został wbudowany w Django mechanizm wysyłania wiadomości email. W celu skorzystania z niego konieczne jest dokonanie jego konfiguracji.

Konfiguracji dokonuje się poprzez ustawienie zmiennych w pliku `settings.py`. Przykładowa konfiguracja dla wysyłanie powiadomień poprzez konto w serwisie Gmail widoczna jest na listingu 15.

Listing 15. Konfiguracja powiadomień email

```
1 EMAIL_USE_TLS = True
2 EMAIL_HOST = 'smtp.gmail.com'
3 EMAIL_HOST_USER = 'student.icis.pcz.pl@gmail.com'
4 EMAIL_HOST_PASSWORD = 'haslo'
5 EMAIL_PORT = 587
```

Funkcje odpowiedzialne za wysyłanie powiadomień zdefiniowane są w pliku `mail.py`. Podstawowymi funkcjami, wywoływanymi z aplikacji są funkcje `send_confirmation_mail()` oraz `send_activation_mail()`. W funkcjach tych przygotowywana jest treść wiadomości w formacie HTML oraz w formie tekstu. Następnie każda z tych funkcji wywołuje funkcję `send_html_mail()`, która z kolei wywołuje metodę klasy `EmailThread().start()`. Klasa `EmailThread` jest klasą dziedziczącą po `threading.Thread` i tworzącą nowy wątek tak, aby działanie portalu nie było przerywane na czas potrzebny na wysłanie wiadomości.

3.4. Zarządzanie repozytoriami

W celu udostępnienia repozytorium innemu użytkownikowi należy w swoim katalogu domowym utworzyć podkatalog `git`, w którym mieściły się będą udostępniane repozytoria. Jako, że zarządzania repozytoriami odbywa się poprzez stronę internetową użytkownik systemowy `www-data` musi mieć uprawnienia do odczytu danych z tego katalogu. Najprostszym rozwiązaniem tego problemu jest zmiana grupy do której należy podkatalog `git` na grupę `www-data`. Ze względu na to, że użytkownik nie może dokonać tego samodzielnie, ze względu na to, że nie posiada takich uprawnień utworzone zostały dwa skrypty. Pierwszym z nich jest `/usr/bin/fixgitperm`, który poprzez narzędzie `sudo` wywołuje właściwy skrypt zmieniający uprawnienia, czyli `/usr/bin/gitfix`. Treść tego skryptu przedstawiona została na listingu 16.

Listing 16. Skrypt `/usr/bin/gitfix`

```
chown $SUDO_USER:www-data /home/$SUDO_USER/git
```

W skrypcie tym została użyta zmienna systemowa `SUDO_USER`[16] określająca użytkownika, który wywołał polecenie `sudo`.

Poza stworzeniem tych dwóch skryptów została odpowiednio zmodyfikowana konfiguracja narzędzia `sudo` w taki sposób, aby użytkownik mógł uruchomić za jego pośrednictwem jedynie skrypt `/usr/bin/gitfix`. Konfiguracji narzędzia `sudo` dokonuje się za pomocą narzędzia `visudo`, które uruchamia domyślny edytor i otwiera w nim plik konfiguracyjny `/etc/sudoers`. W pliku tym dodana zastała linia przedstawiona na listingu 17.

Listing 17. Konfiguracja `sudo`

```
ALL     ALL=(ALL) NOPASSWD: /usr/bin/gitfix
```

Kolejne pola pliku `/etc/sudoers` określają:

- Nazwę użytkownika, który może korzystać z `sudo`. W tym przypadku są to wszyscy użytkownicy.
- Nazwę komputera na którym reguła będzie działała.
- Nazwę użytkowników z uprawnieniami jakich użytkownik uruchamiający `sudo` może wykonać polecenie.
- Polecenie jakie może zostać wykonane. W tym przypadku jest to `/usr/bin/gitfix`, które może zostać wykonane bez podawania hasła.

Po wprowadzeniu takiej konfiguracji użytkownik, który chce udostępniać repozytoria musi wykonać poniższe polecenia:

```
cd ~
mkdir git
fixgitperm
cd git
git --bare init NAZWA_REPOZYTORIUM
```

Polecenia te są prezentowane użytkownikowi po przejściu przez niego na podstronę udostępniania repozytoriów jeśli serwer www nie ma możliwości odczytania listy stworzonych repozytoriów.

3.4.1. Lista repozytoriów

Po zalogowaniu się użytkownika ma on dostęp do listy uprawnień do repozytoriów (rys. 12). Lista ta jest generowana w widoku `git_repos()`. Pierwszym krokiem jest stworzenie klasy `RepoEntry`, a następnie za pomocą metody klasy (*class method*) `Repos.objects.all()` pobranie wszystkich repozytoriów jakie dany użytkownik udostępnia. Repozytoria te dodawane są do listy `repos`, która w funkcji `render_to_response()` przekazywana jest do szablonu.

Rysunek 12. Lista uprawnień

3.4.2. Nadawanie uprawnień

Za dodawanie uprawnień do repozytoriów odpowiedzialny jest widok `git_repo_add()`. W pierwszej linii tej funkcji wywoływana jest funkcja `list_user_repos()`, która przeszukuje katalog `/git/` i pobiera nazwy podkatalogów (czyli repozytoriów) jakie są w nim stworzone. Lista ta zapisywana jest w zmiennej `repos`. Jeśli lista ta jest pusta użytkownikowi wyświetlana jest instrukcja utworzenia repozytorium.

W sytuacji, w której na liście znajdują się repozytoria, prezentowany jest oparty na klasie `RepoForm` formularz dodawania uprawnień (rys. 13). W klasie tej zdefiniowane są trzy pola: pole tekstowe na nazwę użytkownika, któremu udostępniamy repozytorium oraz dwa pola wyboru zawierające listę dostępnych repozytoriów i uprawnień. Lista repozytoriów przekazywana jest do konstruktora klasy 18.

Rysunek 13. Formularz dodawania uprawnień

Listing 18. Konstruktor klasy `RepoForm`

```
1 def __init__(self, *args, **kwargs):
2     repos = kwargs.pop('repos', tuple(""))
3     super(RepoForm, self).__init__(*args, **kwargs)
4     self.fields['reponame'].choices = repos
```

W drugiej linii kodu z listy nazwanych argumentów pobierany jest argument `repos`. Funkcja `pop` wywoływana na słowniku `kwargs` usuwa jednocześnie pobrany argument z tego słownika. Dzięki temu możliwe jest przekazanie go do konstruktora klasy nadrzędnej względem `RepoForm` (linia 3). W linii czwartej następuje inicjalizacja listy wyboru dla pola `reponame`.

Po przesłaniu takiego formularza do aplikacji w tym samym widoku pobierane są obiekty `LdapUser` osoby udostępniającej repozytorium (`user`) oraz osoby dla której takie repozytorium jest udostępniane (`guest`). Następnie do pola zawierającego listę kluczy użytkownika `user` dodawany jest klucz publiczny użytkownika `guest` odnaleziony przez funkcję `find_primary_key` wraz z opcją `command` (zobacz rozdział 2.3.4).

Dla użytkownika `user` tworzona jest także klasa `Git` (przez wywołanie funkcji `git_entry()` zdefiniowanej w pliku `models.py` (zobacz rozdział 3.2)), a następnie obiekt tej klasy, który zapisywany jest do katalogu LDAP. Kiedy jednostka organizacyjna jest utworzona w podobny sposób tworzony jest wpis dla repozytorium z odpowiednim atrybutem LDAP

`userRO` lub `userRW`.

3.4.3. Usuwanie uprawnień

Usuwanie uprawnień do repozytoriów odbywa się w widoku `delete()`. Do widoku oprócz obiektu `request` przekazywana jest nazwa repozytorium, nazwa użytkownika `guest` oraz uprawnienie jakie ma być usunięte.

Kod odpowiedzialny za usuwanie wpisu z katalogu LDAP przedstawiony jest na listingu 19.

Listing 19. Fragment widoku `delete()` odpowiedzialny za usuwanie repozytoriów

```
1  try:
2      if permissions == 'ro':
3          entry = Repos.objects.get(repo=reponame,
               userRO__contains=username)
4          entry.userRO.remove(username)
5          entry.save()
6      elif permissions == 'rw':
7          entry = Repos.objects.get(repo=reponame,
               userRW__contains=username)
8          entry.userRW.remove(username)
9          entry.save()
10 except ObjectDoesNotExist:
11     return HttpResponse("Brak obiektu do usunięcia.")
```

W zależności od tego czy uprawnienie, które ma być usunięte to `ro` czy `rw` pobierany jest odpowiedni obiekt z katalogu za pomocą metody `Repos.objects.get()`. Do metody tej przekazane są dwa nazwane argumenty: `repo` oraz `userRO__contains`. Argumenty te stanowią warunki wyszukania obiektów w katalogu. Zwrócony obiekt musi mieć argument `repo` równy `reponame` oraz w argumencie `userRO` musi zawierać nazwę użytkownika `username`[7].

4. Podsumowanie

Wykorzystanie centralnej bazy użytkowników znacznie upraszcza zarządzanie danymi użytkowników. W środowiskach Linuksowych LDAP jest jednym z najpopularniejszych sposobów centralizacji tych danych. Ze względu na to, że istnieje duża ilość oprogramowania, która posiada obsługę katalogów LDAP możliwe jest wdrożenie usługi katalogowej niewielkim nakładem kosztów. Jeśli natomiast dana aplikacja nie posiada możliwości integracji z serwerem LDAP to dzięki prostemu API oraz istniejącym bibliotekom dla wielu języków programowania bardzo proste jest stworzenie odpowiedniego modułu dostarczającego taką funkcję.

Dzięki elastyczności i brakowi narzuconej struktury katalogu może on zostać dostosowany do przechowywania różnego typu danych w sposób najlepiej odwzorowujący strukturę przechowywanych danych, czyli na przykład strukturę firmy.

W stworzonym systemie katalog LDAP przechowuje zarówno dane użytkowników, jak i informacje o repozytoriach GIT oraz uprawnieniach użytkowników do tych repozytoriów. Wraz z rozwojem potrzeb użytkowników systemu może on zostać rozszerzony przed dodanie kolejnych elementów takich jak na przykład system zarządzania projektami. Redmine oraz Trac, które są jednymi z najpopularniejszych systemów tego typu dostępnych na wolnej licencji posiadają wsparcie dla autoryzacji na podstawie danych z katalogu LDAP.

Wraz z rosnącą ilością danych jakie katalog przechowuje konieczne staje się także zapewnienie odpowiedniego bezpieczeństwa tych danych, czyli zapewnienia ochrony przed dostępem osób niepowołanych jak i również przed ich utratą.

Dzięki mechanizmowi ACL (*Access Control List*) wbudowanemu w serwer OpenLDAP możliwe jest stworzenie zaawansowanych list dostępu do danych wewnątrz katalogu. Listy takie mogą kontrolować dostęp użytkowników do poszczególnych gałęzi drzewa LDAP, a także pojedynczych wpisów czy atrybutów. Poprzez odpowiednią konfigurację list ACL możliwe jest także wydzielenie grup użytkowników o podwyższonych uprawnieniach dostępu do danych, czyli osób, które będą mogły administrować pewnymi fragmentami ka-

talogu.

W celu zabezpieczania przed utratą danych konieczne jest oczywiście wykonywanie okresowych kopii katalogu z wykorzystaniem narzędzia `slapcat`. Serwer OpenLDAP udostępnia także możliwość replikacji danych w czasie rzeczywistym do serwera podrzędnego poprzez mechanizm `syncrepl`. Rozwiązanie takie może być szczególnie przydane w razie awarii sprzętowej.

Literatura

[1] Gordon S. Good Timothy A. Howes Mark C. Smith. *Understanding and Deploying LDAP Directory Services*. Second. Addison Wesley, 2003.

[2] Thorsten Kukuk Andrew G. Morgan. In: Aug. 2010. Chap. pam_unix - traditional password authentication. URL: `http://www.linux-pam.org/Linux-PAM-html/sag-pam_unix.html` (data dostępu 05/20/2012).

[3] *django-auth-ldap 1.1*. URL: `http://pypi.python.org/pypi/django-auth-ldap/1.1` (data dostępu 05/21/2012).

[4] *Django documentation: Databases*. URL: `https://docs.djangoproject.com/en/1.4/ref/databases/` (data dostępu 05/21/2012).

[5] *Django documentation: Form and field validation*. URL: `https://docs.djangoproject.com/en/1.4/ref/forms/validation/#cleaning-a-specific-field-attribute` (data dostępu 05/21/2012).

[6] *Django documentation: Model Meta options*. URL: `https://docs.djangoproject.com/en/1.4/ref/models/options/` (data dostępu 05/21/2012).

[7] *Django documentation: QuerySet API reference*. URL: `https://docs.djangoproject.com/en/1.4/ref/models/querysets/` (data dostępu 05/21/2012).

[8] *Django documentation: User authentication in Django*. URL: `https://docs.djangoproject.com/en/1.4/topics/auth/` (data dostępu 05/21/2012).

[9] L. Howard. *An Approach for Using LDAP as a Network Information Service*. RFC 2307 (Experimental). Internet Engineering Task Force, Mar. 1998. URL: `http://www.ietf.org/rfc/rfc2307.txt`.

[10] *man 1 getent*. URL: `http://man7.org/linux/man-pages/man1/getent.1.html` (data dostępu 05/20/2012).

[11] *man 5 nclcd.conf*. URL: `http://linux.die.net/man/5/nslcd.conf` (data dostępu 05/20/2012).

[12] *man 5 nsswitch.conf*. URL: `http://linux.die.net/man/5/nsswitch.conf` (data dostępu 05/20/2012).

[13] *man 5 shadow*. URL: `http://linux.die.net/man/5/shadow` (data dostępu 05/20/2012).

[14] *man 8 nclcd*. URL: `http://linux.die.net/man/8/nslcd` (data dostępu 05/20/2012).

[15] *man 8 pam*. URL: `http://linux.die.net/man/8/pam` (data dostępu 05/20/2012).

[16] *man 8 sudo*. URL: `http://linux.die.net/man/8/sudo` (data dostępu 05/20/2012).

[17] *Python v2.7.3 documentation - The Python Standard Library - Built-in Functions*. URL: `http://docs.python.org/library/functions.html#type` (data dostępu 05/21/2012).

[18] A. Sciberras. *Lightweight Directory Access Protocol (LDAP): Schema for User Applications*. RFC 4519 (Proposed Standard). Internet Engineering Task Force, June 2006. URL: `http://www.ietf.org/rfc/rfc4519.txt`.

[19] M. Smith. *Definition of the inetOrgPerson LDAP Object Class*. RFC 2798 (Informational). Updated by RFCs 3698, 4519, 4524. Internet Engineering Task Force, Apr. 2000. URL: `http://www.ietf.org/rfc/rfc2798.txt`.

Zawartość płyty

Do pracy załączona została płyta CD-ROM, która zawiera:

- pracę magisterską w formacie PDF (w katalogu `praca`)
- kod źródłowy wykonanego programu (w katalogu `program`)
- tekstową wersję pracy w formacie LaTeX (w katalogu `praca/latex`)