Single Sign-On Service Provider

Выпуск 1.1.0

БАРС Груп

27 May 2015

Оглавление

SSO - Технология единого входа в комплекс приложений (Single Sign-On) может реализовываться различными способами. Один из них - спецификация SAML 2.0.

За основу была взята реализация [1] сервис провайдера SAML 2 для Django (которая в свою очередь является наследником другой реализации [2]).

Содержание:

Глава 1

Описание

1.1 Назначение

Модуль SSOSP предназначен для более простой интеграции *SSO* в веб-приложения на Django по спецификации *SAML*. Реализуется функционал поставщика услуг (*SP*) по взаимодействию с поставщиком идентификации (*IdP*) для обеспечения технологии единого входа и выхода (*SSO*). Поддерживается *SAML* с цифровой подписью сообщений.

В качестве протокола взаимодействия используется спецификация SAML версии 2.0.

Примечание: Терминология

SSO - Технология единого входа в комплекс приложений (Single Sign-On).

SAML - Язык разметки подтверждения безопасности (Security Assertion Markup Language) используется для обеспечения сквозной аутентификации при работе через браузер.

SP - Поставщик услуг (Service provider в терминах *SAML*), защищенный сервис предоставляющий ресурсы. Обычно, это веб-приложение или веб-сервисы.

IdP - Поставщик идентификации (Identity provider в терминах *SAML*), выполняющий аутентицикацию пользователя, запрощенную поставщиком услуг.

ACS - Сервис обработки утверждений (Assertion Consumer Service в терминах *SAML*) на стороне поставщика услуг, получающий запросы со стороны поставщика идентификации.

Single Logout - подразумевает *Единый выход* из приложений, в которые вошел пользователь через *IdP*. Т.е. при выходе из одного приложения, автоматически происходит выход из остальных.

Согласно спецификации SAML, *SP* обращается к *IdP* для аутентификации пользователя, обратившегося к *SP* с каким-либо запросом.

Примерные схема взаимодействия между участниками по спецификации SAML на примере единого входа:

Модуль реализует функционал *SP* и *ACS*. Согласно схеме выше, реализуются шаги №2, №3, №7. Шаги №8 и №9 реализуются непосредственно приложением, в которое встраивается данный модуль.

Дополнительно, реализуется механизм *Single Logout*, который при выходе пользователя позволяет осуществить корректный выход из всех приложений, в которые он заходил.

В этом взаимодействии модуль реализует шаги №2, №3, №7 и №9.

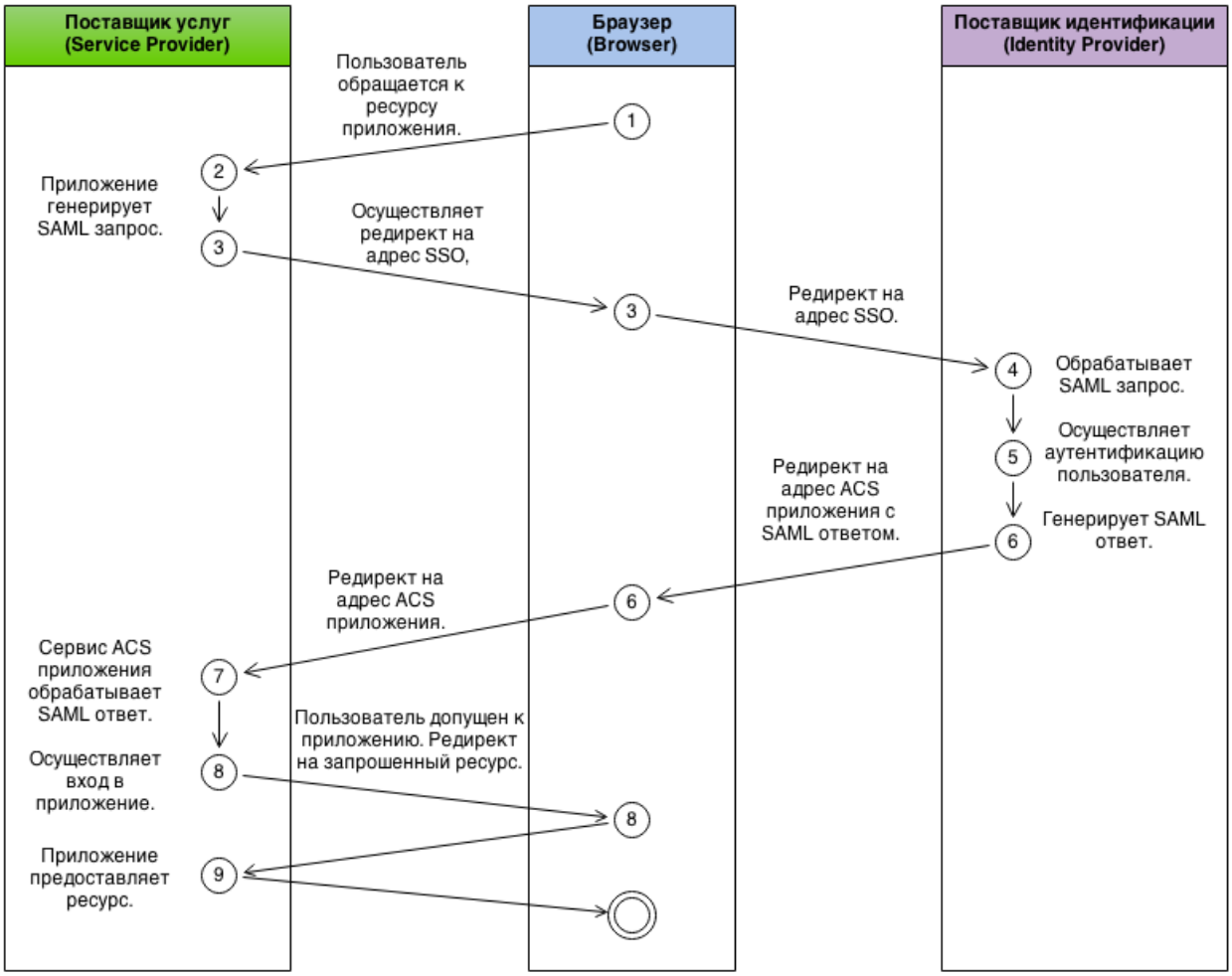


Fig. 1.1: Шаги единого входа в приложение

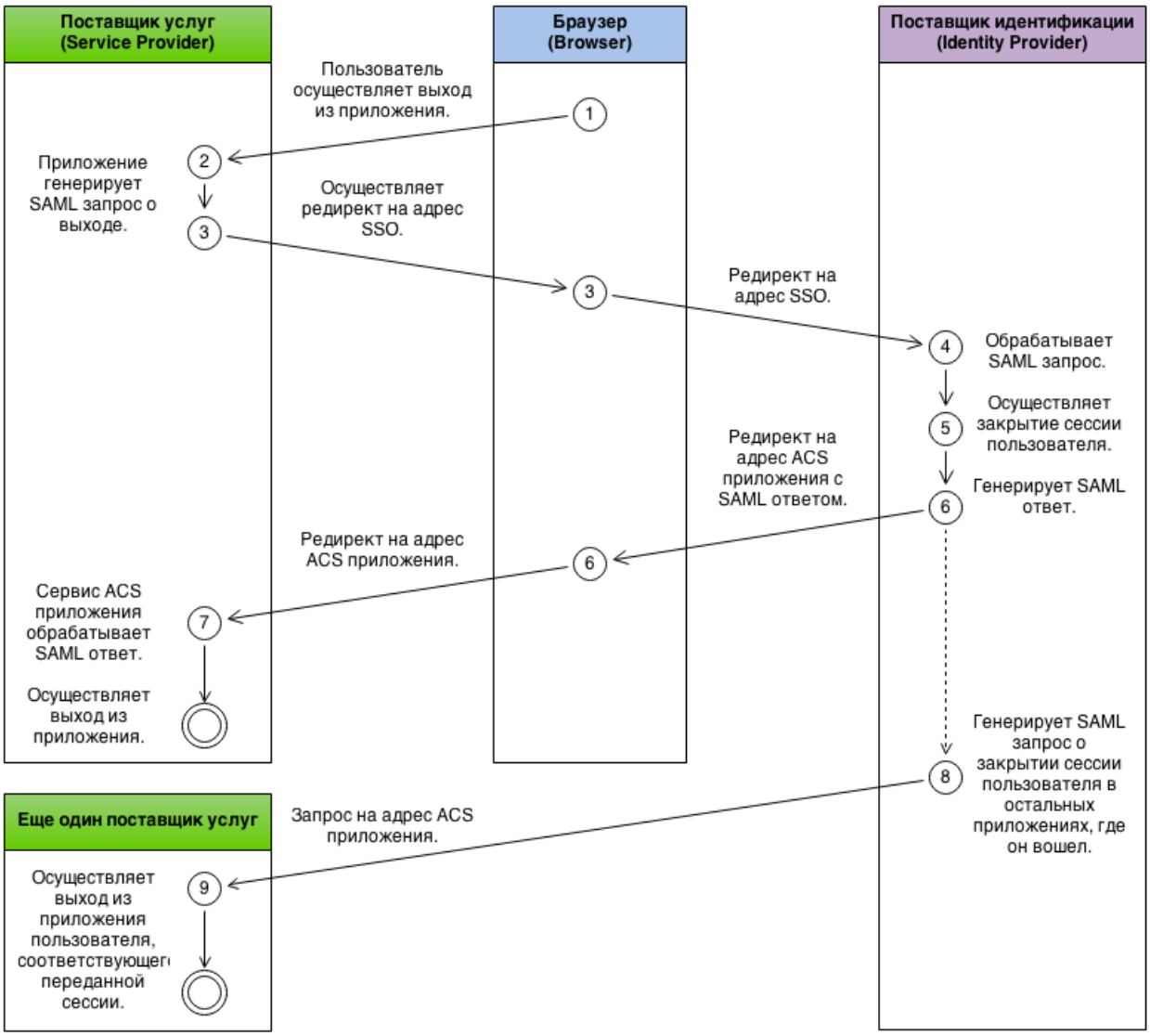


Fig. 1.2: Шаги единого выхода из приложения

1.2 Структура

Модуль содержит реализацию обработчиков:

- Входа в приложение
- Выхода из приложения
- Сервиса обработки утверждений (*ACS*)

Работа с *SAML* реализована через классы SAML-запросов: AuthRequest, LogoutRequest. И SAML-ответов: AuthResponse, LogoutResponse.

Поддержка механизма *Single Logout* требует хранения соответствия Django-сессии и сессии на *IdP*, чтобы при завершении сессии *IdP* закрывать соответствующую Django-сессию. Хранение соответствия сессий реализовано через механизм включения необходимого бэкенда. В модуль входят 2 вида бэкенда:

- Хранение соответствия в БД (модель SSOSession).
- Хранение соответствия в кэше Django.

Глава 2

Установка

Модуль SSOSP подключается как приложение Django. Поддерживается Django версии 1.3 и старше.

Для функционирования, модулю SSOSP требуется установка дополнительных библиотек.

2.1 Необходимые библиотеки

- lxml >= 3.0
- rsa >= 3.1.2
- xmldsig == 1.0.0

Их можно установить одной командой:

```
pip install lxml rsa xmldsig -i https://<PyPi_сервер_БАРС_Груп>
```

Подключение PyPi сервера необходимо для пакета xmldsig.

2.2 Используя pip

Установите пакет ssosp из репозитория пакетов компании `БАРС Груп`

```
pip install ssosp -i https://<PyPi_сервер_БАРС_Груп>
```

В этом случае будут установлеты также все необходимые пакеты.

Теперь можно приступать к *настройке Django-приложения*.

2.3 Скачать архив

Скачайте и распакуйте архив модуля https://bitbucket.org/barsgroup/ssosp/downloads

2.4 Используя Mercurial

Клонируйте исходный код модуля из репозитория

```
hg clone https://bitbucket.org/barsgroup/ssosp
```

2.5 Установка из каталога

Установка

```
python setup.py install
```

2.6 Настройка

Основная настройка модуля осуществляется в settings.py приложения. В нем надо создать словарик парметров настройки *SSO_ CONFIG*.

Пример заполнения словаря настройки:

```
SSO_CONFIG = {
    'idp': 'https://localhost:9443/samlsso', # адрес Identity Provider
    'issuer': 'saml2.demo2', # код связи между IdP и SP
    'acs': 'http://localhost:9000/sso/acs/', # адрес сервиса ACS
    'index': '1906473741',
    'session_map': 'ssosp.backends.cache', # бэкенд соответствия сессий
    'get_user': 'demo2.views.get_or_create_user', # получение пользователя
}
```

Параметры настройки:

- **idp - адрес *IdP*. Именно на этот адрес будут отправляться SAML-запросы. Обязательный параметр**
- **issuer - код приложения, под которым оно зарегистрировано в *IdP*. Обязательный параметр**
- **acs - адрес сервиса *ACS*. Этот адрес должен быть зарегистрирован в *IdP*** для этого приложения и также указывается в SAML-запросах. Именно по этому адресу будут приходит SAML-запросы от *IdP* в приложение и происходить редирект. Т.е. это должен быть внешний адрес! **Обязательный параметр**
- **index - код набора атрибутов (*Claims*), передаваемых при аутентификации** пользователя на *IdP*. При изменении набора передаваемых атрибутов на *IdP* изменяется и этот код. По умолчанию, этот параметр отсутствует и атрибуты не передаются.
- **session_map - путь к бэкенду, отвечающему за хранение соответствий** django-сессии и сессии на *IdP*. В состав *SSOSP* входят 2 бэкенда:
  - 'ssosp.backends.db' - хранение в БД (модель SSOSession)
  - 'ssosp.backends.cache' - хранение в кэше django

  Значение по умолчанию: 'ssosp.backends.db'.
- **cache_timeout - длительность хранения сессий в кэше django в случае использования** 'ssosp.backends.cache'. Значение по умолчанию: 2592000 (30 дней).
- **get_user - путь к функции поиска пользователя в приложении, по переданным** данным из *IdP* (передаются userid и список claims). На выходе функции должен быть соответствующий пользователь django, либо *None*, если не нашли. Если параметр не указан, то пользователь ищется в модели User по соответствию username = userid.

- **login - путь к функции входа пользователя в приложение. В функцию передается** request и user. Этот параметр предназначен для обработки входа в приложение.

  По умолчанию, используется функция *login* из модуля *django.contrib.auth*.

- **logout - путь к функции выхода пользователя из приложения. В функцию** передается request. Этот параметр предназначен для обработки выхода из приложения.

  По умолчанию, используется функция *logout* из модуля *django.contrib.auth*.

- **zipped - логический параметр, определяющий, необходимоть распаковки данных** при получении SAML-сообщений. (Предыдущий версии *WSO2IS* отправляли запросы с упаковкой - теперь нет)

  Значение по умолчанию: *False*.

- **validate - логический параметр, определяющий, необходимость проверки цифровой** подписи поступающих SAML-сообщений, подписанных по спецификации XMLDSIG.

  Значение по умолчанию: *False*.

- **public_key - строка публичного ключа для проверки цифровой подписи** SAML-сообщений. Используется совместно с параметром *validate*.

  Значение по умолчанию: *None*.

- **signing - логический параметр, определяющий, необходимость цифровой подписи** отправляемых SAML-сообщений, по спецификации SimpleSign.

  Значение по умолчанию: *False*.

- **private_key - строка закрытого ключа для цифровой подписи SAML-сообщений.** Используется совместно с параметром *signing*.

  Значение по умолчанию: *None*.

Глава 3

Демо проект

Быстренько напишем демонстрационный проект и проверим его работу.

Для проверки приложения возьмем, например, бесплатный *IdP* WSO2 Identity Server.

3.1 Разворачиваем WSO2 Identity Server

1. Скачиваем дистрибутив https://svn.wso2.org/repos/wso2/people/dulanja/scratch/wso2is-4.6.0.zip
2. Распаковываем wso2is-4.6.0.zip в папку wso2is-4.6.0
3. Запускаем *WSO2IS*

```
/wso2is-4.6.0/bin/wso2server.sh (или *bat* для Windows)
```

4. Заходим в *WSO2IS* https://localhost:9443/carbon/ (пользователь *admin* пароль *admin*)
5. Создадим нового пользователя.

   Идем Home > Configure > Users and Roles > Users > Add New User

   Создадим пользователя *demo* с паролем *demo123*.
6. Добавим право входа для пользователя.

   Идем Home > Configure > Users and Roles > Roles

   Редактируем права (Permission) для роли *Internal/everyone*.
7. Зарегистрируем новый Service Provider (который напишем чуть позже).

   Идем Main > Manage > SAML SSO > Register New Service Provider

   Создадим поставщика услуг:

```
Issuer: saml2.demo

Assertion Consumer URL: http://127.0.0.1:8000/sso/acs/
```

3.2 Пишем приложение

1. Подготавливаем виртуальное окружения для приложения:

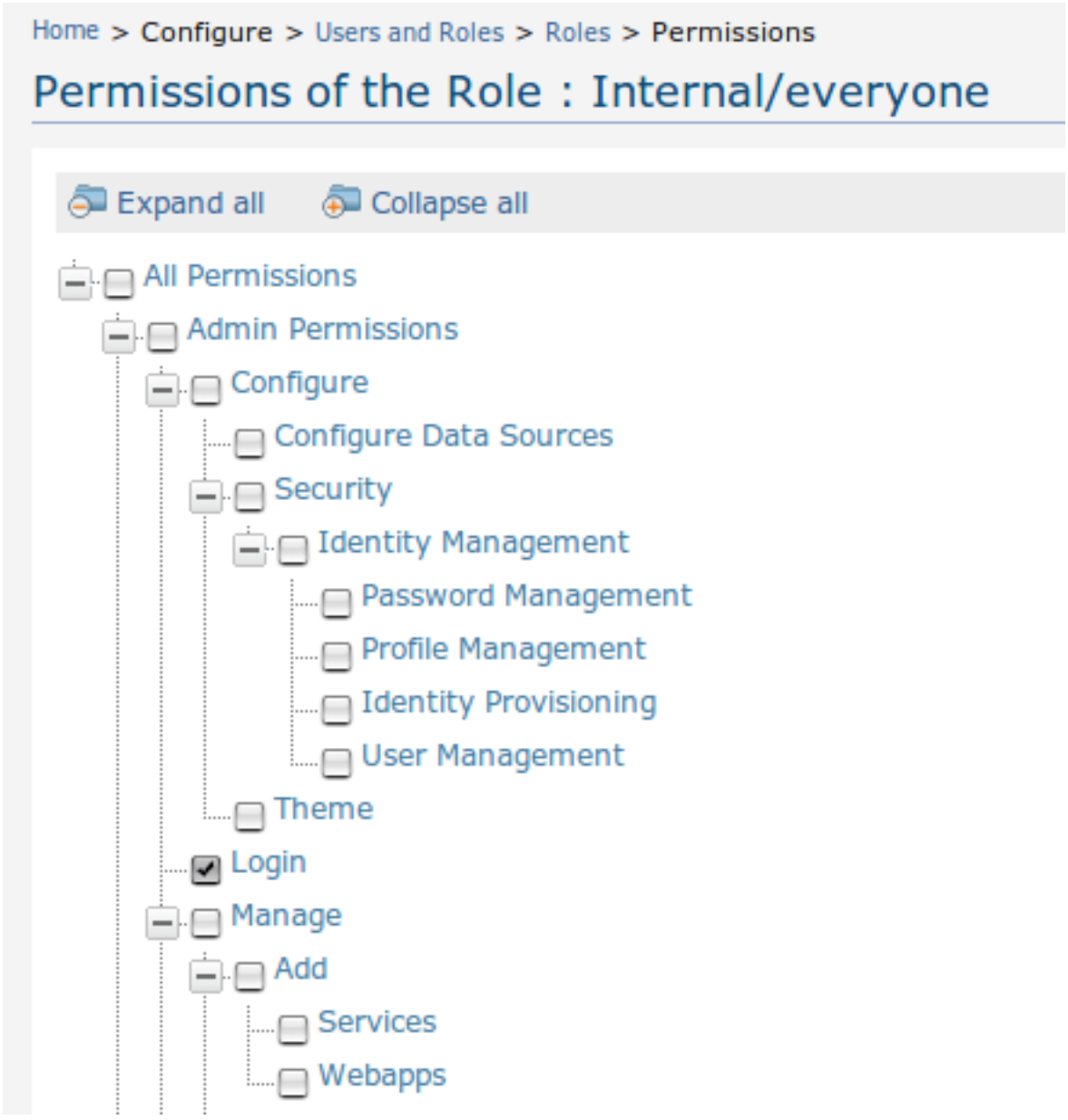
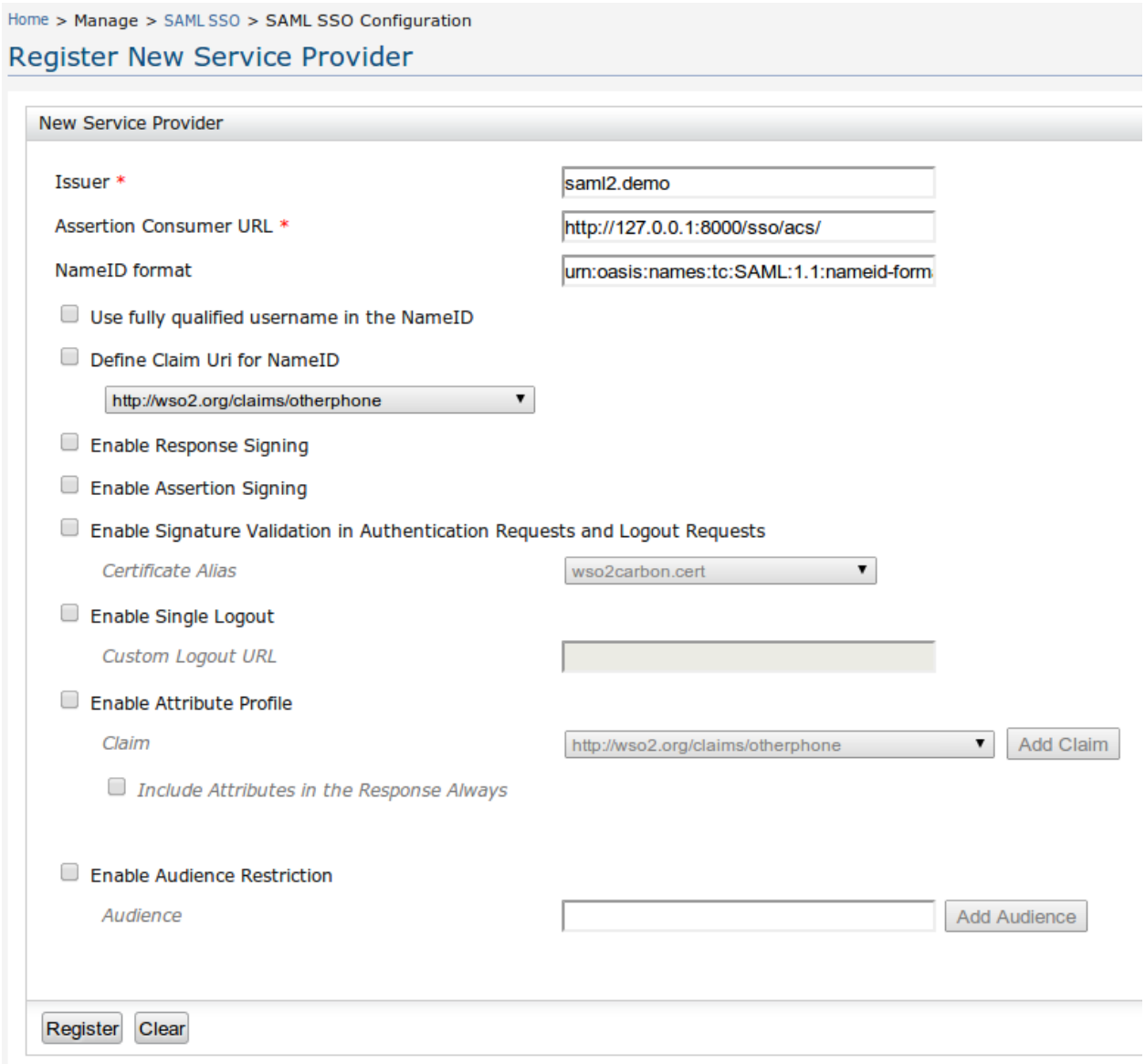


Fig. 3.1: Выставим право *Login*


Home > Manage > SAML SSO > SAML SSO Configuration
Register New Service Provider
New Service Provider
Issuer *
saml2.demo
Assertion Consumer URL *
http://127.0.0.1:8000/sso/acs/
NameID format
urn:oasis:names:tc:SAML:1.1:nameid-form
Use fully qualified username in the NameID
Define Claim Uri for NameID
http://wso2.org/claims/otherphone
Enable Response Signing
Enable Assertion Signing
Enable Signature Validation in Authentication Requests and Logout Requests
Certificate Alias
wso2carbon.cert
Enable Single Logout
Custom Logout URL
Enable Attribute Profile
Claim
http://wso2.org/claims/otherphone
Add Claim
Include Attributes in the Response Always
Enable Audience Restriction
Audience
Add Audience
Register
Clear

```
mkdir ssospdemo

cd ssospdemo

virtualenv --no-site-packages ./venv

source ./venv/bin/activate

pip install lxml rsa xmldsig django ssosp -i http://pypi.bars-open.ru/simple
```

2. Создаем проект:

```
django-admin.py startproject demo
```

3. В settings.py добавляем блок настройки SSO:

```
SSO_CONFIG = {
    'idp': 'https://localhost:9443/samlsso', # адрес Identity Provider
    'issuer': 'saml2.demo', # код связи между IdP и SP
    'acs': 'http://127.0.0.1:8000/sso/acs/', # адрес сервиса ACS
    'session_map': 'ssosp.backends.cache', # бэкенд соответствия сессий
    'get_user': 'demo.views.get_or_create_user', # получение пользователя
}
```

Также в INSTALLED_APPS добавляем наш модуль 'ssosp':

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ssosp',
)
```

Указываем какую-нибудь базу данных для проекта:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'demo',
    }
}
```

И укажем где искать шаблоны приложения:

```
import os
PROJECT_ROOT = os.getcwd()

TEMPLATE_DIRS = (
    '%s/templates' % PROJECT_ROOT,
)
```

В некоторых случаях может быть подключен *CsrfViewMiddleware* - для демо приложения его надо отключить.

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    #'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)
```

4. Создаем файл view.py, в котором будет функция отображения главной страницы (default) и метод поиска пользователя по переданным атрибутам (get_or_create_user).

   Функция *get_or_create_user* ищет пользователя по переданному от *IdP* userid и если не находит, то создает нового пользователя.

```
#coding:utf-8
from django.shortcuts import render_to_response
from django.contrib.auth.models import User
from django.conf import settings


def default(request):
    tv = {
        'user': request.user,
    }
    return render_to_response('default.html', tv)


def get_or_create_user(userid, attributes):
    try:
        user = User.objects.get(username=userid)
    except User.DoesNotExist:
        user = User.objects.create_user(userid, userid)
    # возьмем первый попавшийся бэкенд
    user.backend = settings.AUTHENTICATION_BACKENDS[0]
    return user
```

5. В url.py добавляем ссылки на SSO и главную страницу:

```
from views import default

urlpatterns = patterns('',
    url(r'^sso/', include('ssosp.urls')),
    url(r'^', default, name="default"),
)
```

6. В папке ‘templates’ создаем файл шаблона главной страницы default.html:

```
<html>
<head><title>Django SAML 2.0 SP</title></head>
<body>
{% if not user.username %}
I don't recognize you! Please login:<br />
<a href="{% url login %}?next={% url default %}">Login</a>
{% else %}
Welcome, {{ user.username }}!<br />
<hr>
<a href="{% url logout %}?next={% url default %}">Logout</a>
{% endif %}
</body>
```

Примечание: Для Django 1.5 и старше, в тэге *url* следует указывать имя функции в кавычках: {% url ‘login’ %}

7. Создаем базу и запускаем пример:

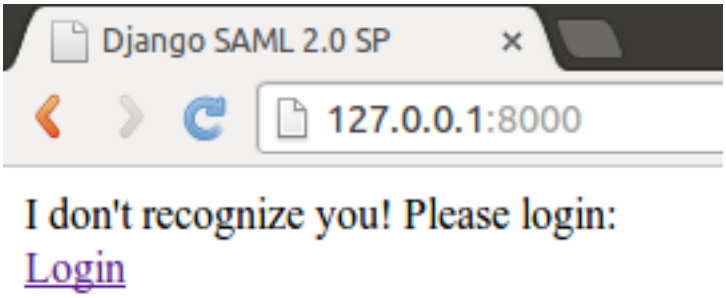
```
cd demo

python manage.py syncdb

python manage.py runserver
```

3.3 Проверяем

Открываем страницу приложения http://127.0.0.1:8000



и нажимаем на *Login* для входа в наше приложение.

Произойдет редирект на адрес *IdP*, который мы указали в настройках https://localhost:9443/samlsso

Так как мы еще не авторизованы, то *WSO2IS* запросит у нас имя пользователя и пароль.

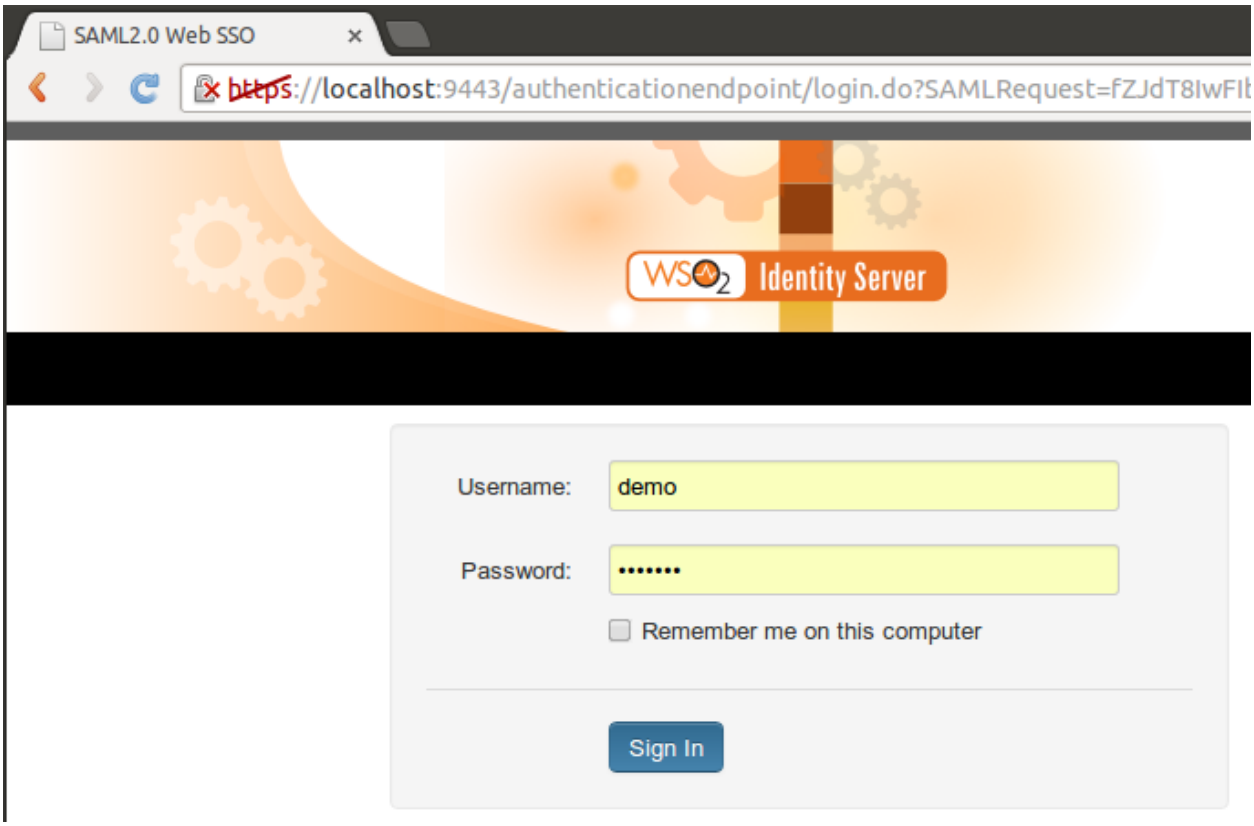
После успешной аутентификации, *WSO2IS* редиректит нас обратно в наше приложение на адрес *ACS*, где приложение обрабатывает результаты аутентификации и осуществляет вход.

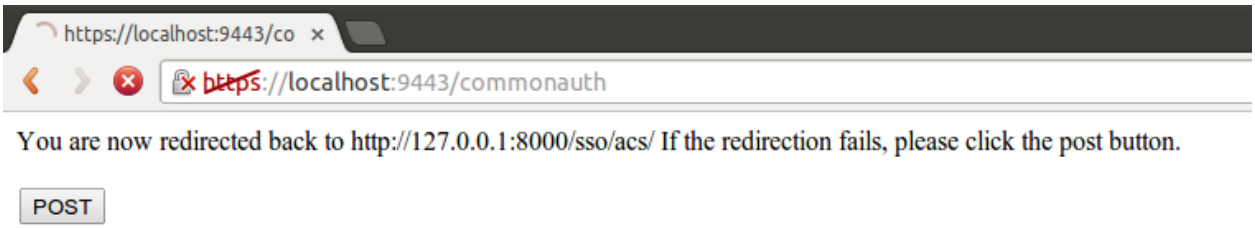
Это происходит достаточно быстро, поэтому можно не заметить как промелькнет эта страница:

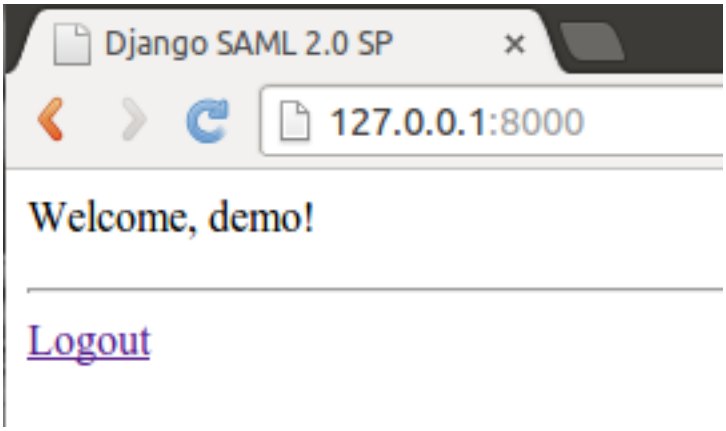
Всё! Теперь наше приложение успешно пустило нас.

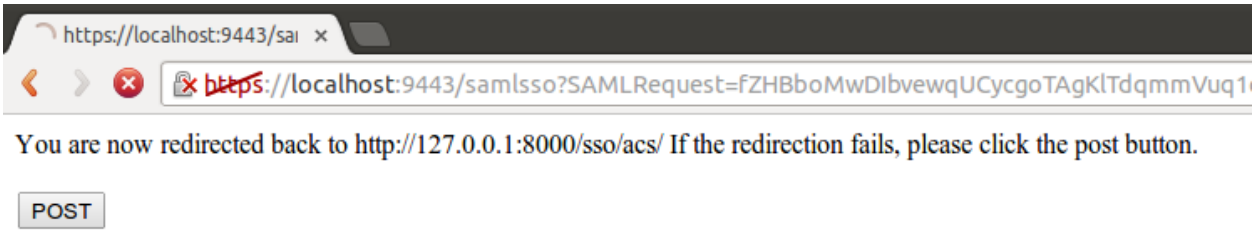
Теперь выходим из приложения нажав *Logout*.

После серии редиректов мы возвращаемся к первоначальному состоянию.


SAML2.0 Web SSO
https://localhost:9443/authenticationendpoint/login.do?SAMLRequest=fZJdT8IwFIb
WSO2 Identity Server
Username:
demo
Password:
Remember me on this computer
Sign In


https://localhost:9443/co
https://localhost:9443/commonauth
You are now redirected back to http://127.0.0.1:8000/sso/acs/ If the redirection fails, please click the post button.
POST


Django SAML 2.0 SP
127.0.0.1:8000
Welcome, demo!
Logout


https://localhost:9443/sa
https://localhost:9443/samlsso?SAMLRequest=fZHBboMwDIbvewqUCycgoTAgKlTdqmmVuq1
You are now redirected back to http://127.0.0.1:8000/sso/acs/ If the redirection fails, please click the post button.
POST

Глава 4

Простое использование

Примеры простого использования будут основаны на демо-приложении.

4.1 Получение дополнительных сведений о пользователе

Сразу после установки WSO2IS и регистрации нашего приложения не передается никаких дополнительных атрибутов пользователя.

1. Заполним атрибуты у созданного ранее пользователя WSO2IS.

   Идем Home > Configure > Users and Roles > Users

   Находим нашего пользователя *demo*, открываем и редактируем его *User Profile*.

2. Затем укажем, какие атрибуты нужно передавать при успешной аутентификации.

   Идем Main > Manage > SAML SSO

   Находим наше приложение *saml2.demo*, открываем и редактируем его.

   Укажем признак *Enable Attribute Profile* и добавим несколько *Claim*. Например:

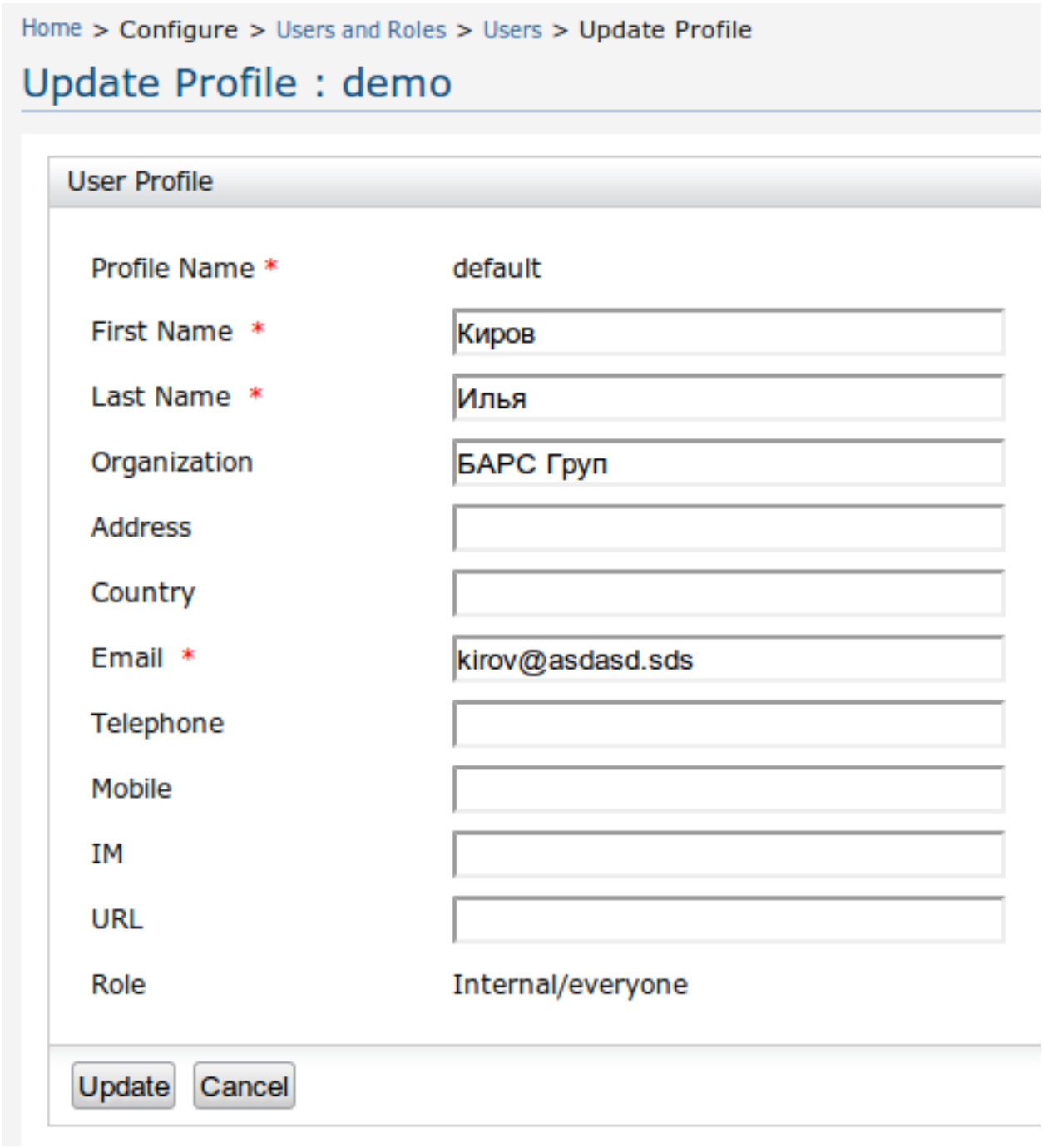
Примечание: Обращаю внимание, что после создания профиля атрибутов у зарегистрированного поставщика услуг появляется значение *Consumer Index*. Которое нужно указывать в настройках нашего приложения. Вероятно, можно создать несколько разных профилей атрибутов для одного и того же приложения. Но это пока не поддерживается модулем *SSOSP*.

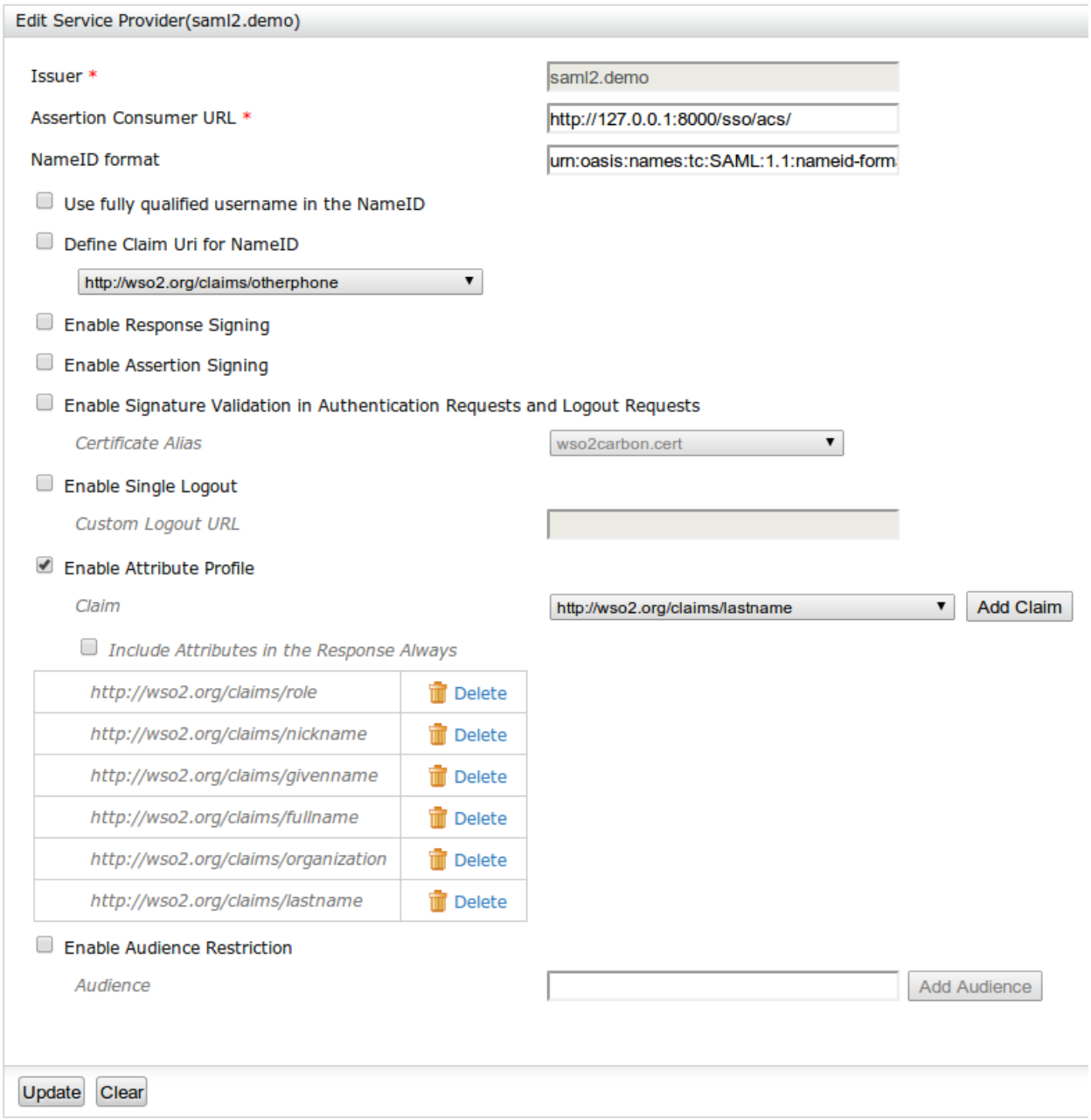
3. Укажем полученный *Consumer Index* в настройках settings.py:

```
SSO_CONFIG = {
    'idp': 'https://localhost:9443/samlsso', # адрес Identity Provider
    'issuer': 'saml2.demo', # код связи между IdP и SP
    'acs': 'http://127.0.0.1:8000/sso/acs/', # адрес сервиса ACS
    'session_map': 'ssosp.backends.cache', # бэкенд соответствия сессий
    'get_user': 'demo.views.get_or_create_user', # получение пользователя
    'index': '1537824998', # индекс профиля атрибутов (Consumer Index)
}
```

4. Атрибуты, после входа пользователя в систему, сохраняются в сессии пользователя. Получим их при отображении страницы приложения в функции default (view.py) и передадим в шаблон:

```
def default(request):
    attributes = request.session.get('attributes', {})
    tv = {
```


Home > Configure > Users and Roles > Users > Update Profile
Update Profile : demo
User Profile
Profile Name *
default
First Name *
Киров
Last Name *
Илья
Organization
БАРС Груп
Address
Country
Email *
kirov@asdasd.sds
Telephone
Mobile
IM
URL
Role
Internal/everyone
Update
Cancel


Edit Service Provider(saml2.demo)
Issuer *
saml2.demo
Assertion Consumer URL *
http://127.0.0.1:8000/sso/acs/
NameID format
urn:oasis:names:tc:SAML:1.1:nameid-form
Use fully qualified username in the NameID
Define Claim Uri for NameID
http://wso2.org/claims/otherphone
Enable Response Signing
Enable Assertion Signing
Enable Signature Validation in Authentication Requests and Logout Requests
Certificate Alias
wso2carbon.cert
Enable Single Logout
Custom Logout URL
Enable Attribute Profile
Claim
http://wso2.org/claims/lastname
Add Claim
Include Attributes in the Response Always
http://wso2.org/claims/role
Delete
http://wso2.org/claims/nickname
Delete
http://wso2.org/claims/givenname
Delete
http://wso2.org/claims/fullname
Delete
http://wso2.org/claims/organization
Delete
http://wso2.org/claims/lastname
Delete
Enable Audience Restriction
Audience
Add Audience
Update
Clear

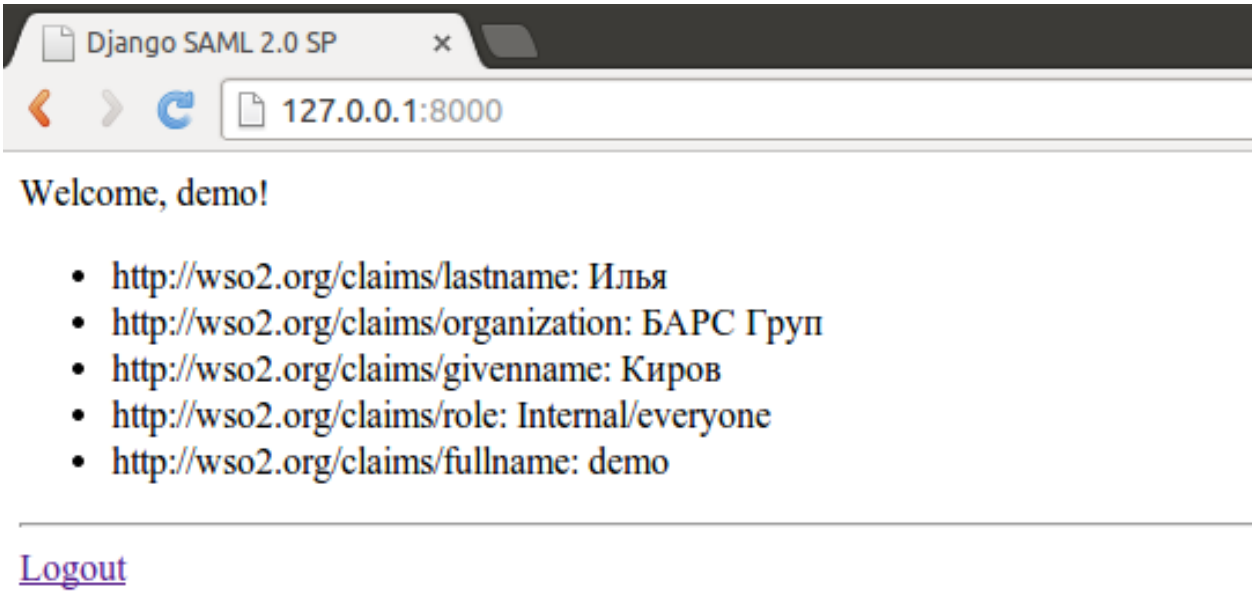
```
            'user': request.user,
            'attributes': attributes,
        }
        return render_to_response('default.html', tv)
```

4. Добавим отображение атрибутов на странице приложения.

   В шаблоне default.html добавим вывод списка атрибутов.

```
<html>
<head><title>Django SAML 2.0 SP</title></head>
<body>
{% if not user.username %}
I don't recognize you! Please login:<br />
<a href="{% url login %}?next={% url default %}">Login</a>
{% else %}
Welcome, {{ user.username }}!<br />
<ul>
{% for name, value in attributes.items %}
<li>{{ name }}: {{ value }}
{% endfor %}
</ul>
<hr>
<a href="{% url logout %}?next={% url default %}">Logout</a>
{% endif %}
</body>
```

5. Теперь, после входа пользователя, в приложении отображаются атрибуты пользователя из *WSO2IS*:

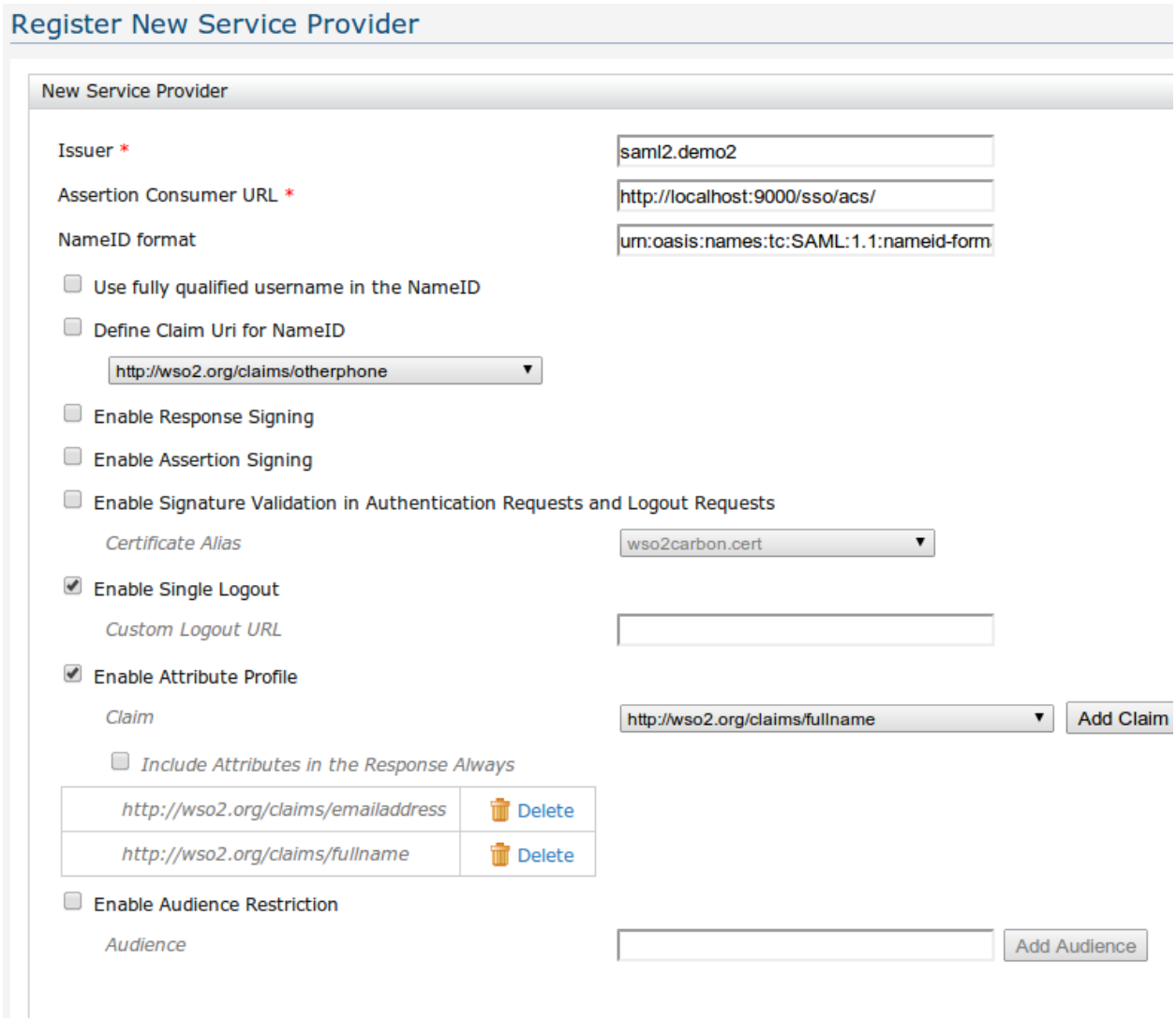


4.2 SSO и Single Logout

При регистрации нашего приложения или позже, можно настроить возможность *Единого выхода* для этого приложения. Для этого в описании сервисов *SSO WSO2IS* нужно отметить соответствующий параметр *Enable Single Logout*.

В самом нашем приложении дополнять ничего не нужно. Выход из приложения обрабатывается модулем *SSOSP* через сервис *ACS*.

Для проверки работы достаточно поднять копию демо-приложения, например, на порту 9000. Также, необходимо зарегистрировать в *WSO2IS* эту вторую копию, но соответственно по другому адресу и с другим параметром *Issuer*.



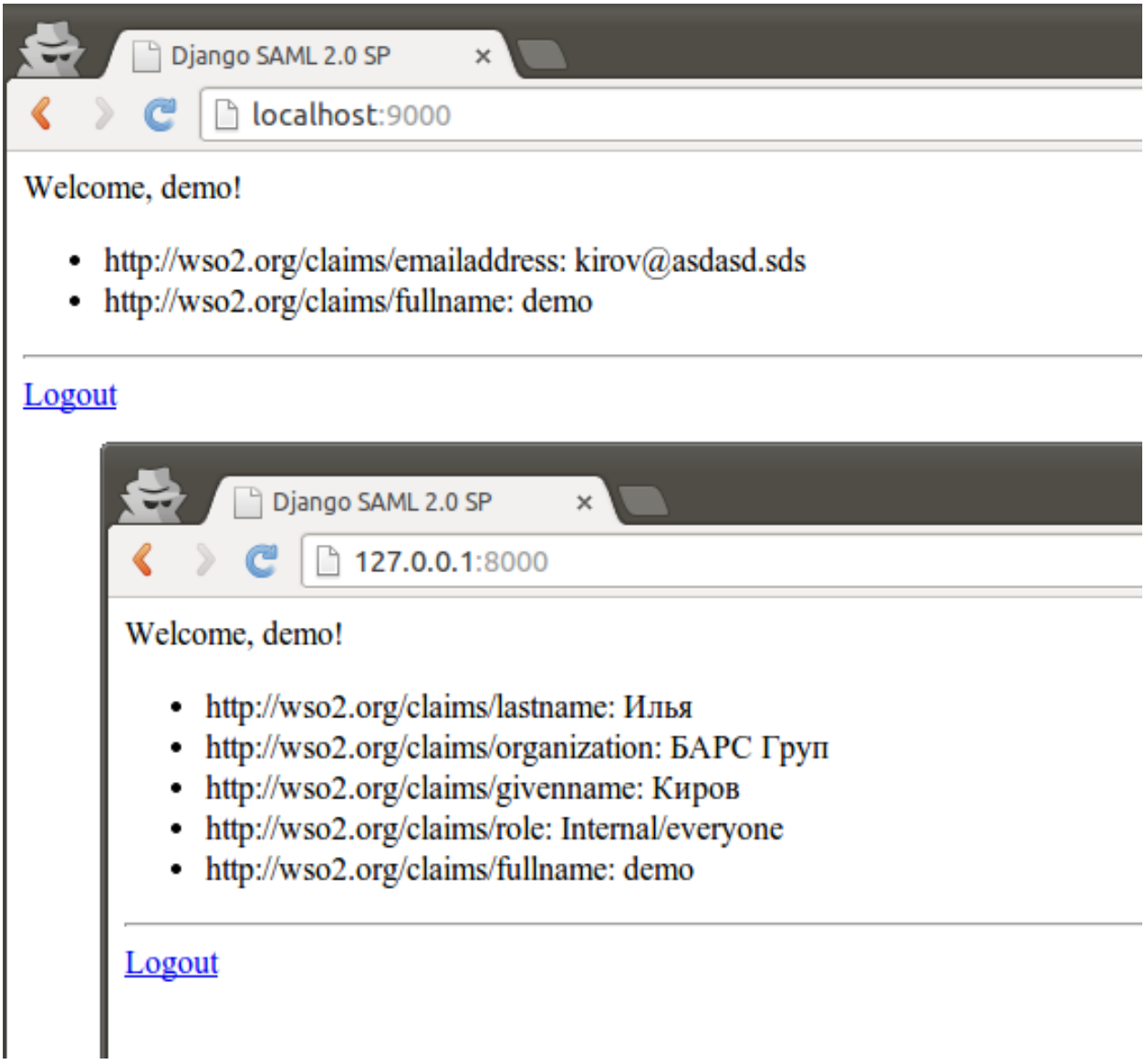
Во втором приложении необходимо соответственно поменять настройки *issuer*, *acs* и *index* (если использовался профиль атрибутов) в settings.py.

Если изменилось имя приложения, то надо поправить ссылки *get_user* и *ROOT_URLCONF* (у нас стало приложение demo2):

```
ROOT_URLCONF = 'demo2.urls'

SSO_CONFIG = {
    'idp': 'https://localhost:9443/samlsso', # адрес Identity Provider
    'issuer': 'saml2.demo2', # код связи между IdP и SP
    'acs': 'http://localhost:9000/sso/acs/', # адрес сервиса ACS
    'index': '1906473741',
    'session_map': 'ssosp.backends.cache', # бэкенд соответствия сессий
    'get_user': 'demo2.views.get_or_create_user', # получение пользователя
}
```

После запуска двух приложений, при входе в первое из приложений будет запрошен логин и пароль пользователя *WSO2IS*. При входе во второе приложение уже не потребуется вводить логин и пароль (при условии, что это происходит в одном браузере). **Вот оно SSO!**



Теперь, при выходе из одного из приложений, на второе приложение придет запрос о завершении сессии на адрес *ACS*. В результате, при обновлении страницы второго приложения пользователь окажется не

авторизован. Т.е. пользователь выйдет из обоих приложений.

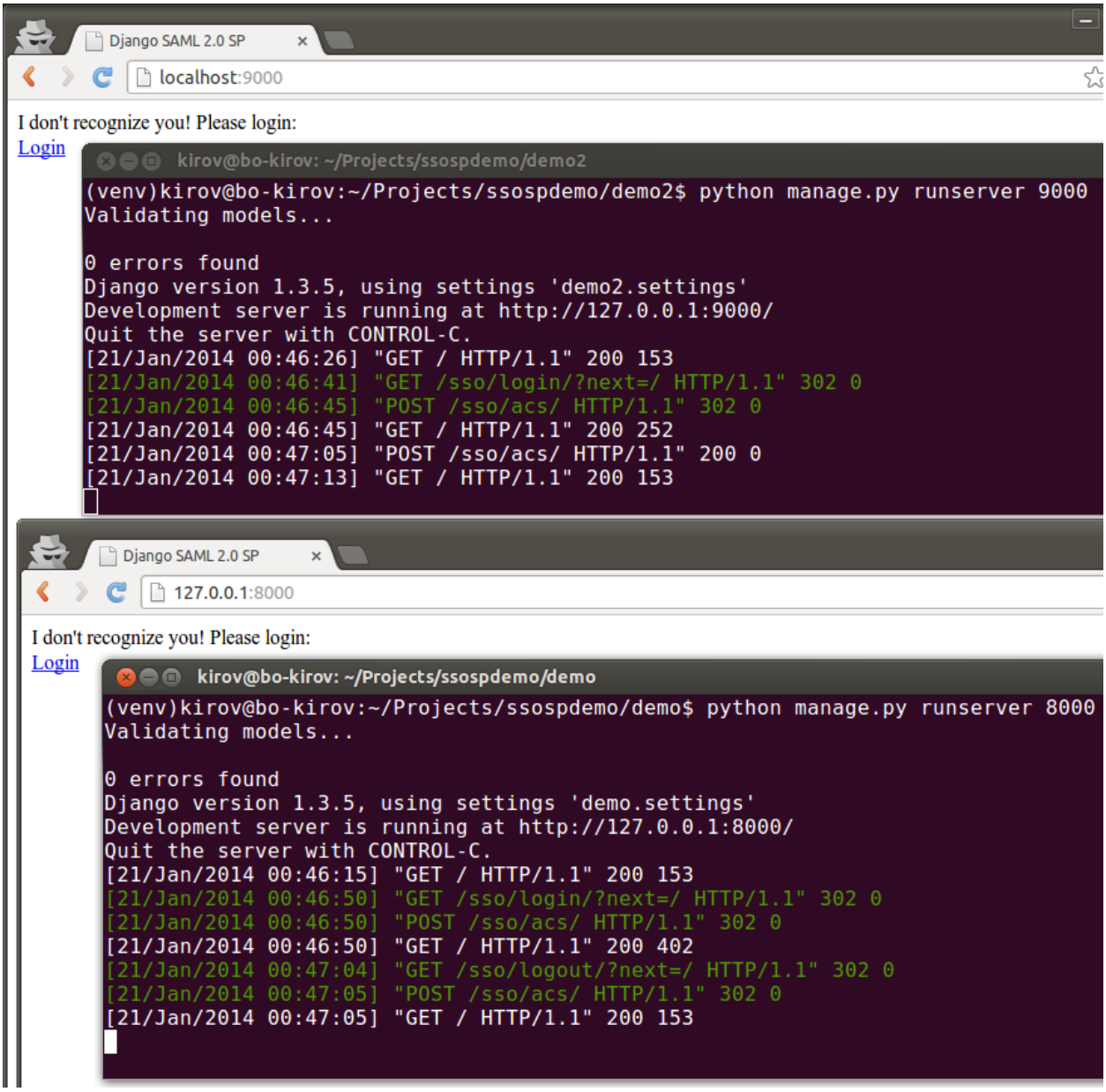


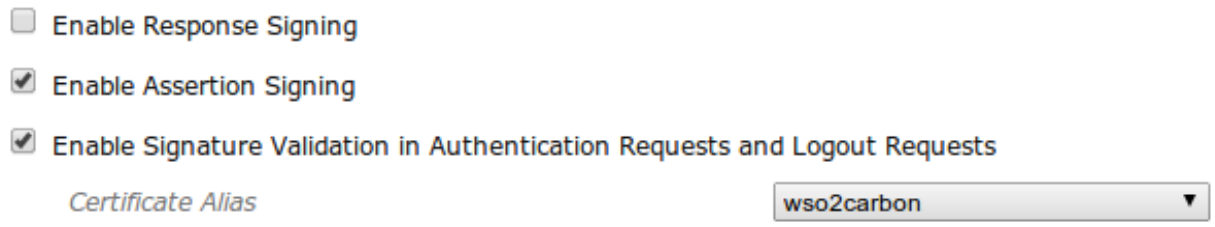
Fig. 4.1: В истории запросов второго приложения нет обращения к /logout

4.3 Подписывание сообщений SAML

В *IdP* сервере *WSO2IS* есть настройка, которой можно установить возможность использования цифровой подписи SAML-сообщений.

Идем Main > Manage > SAML SSO

Находим наше приложение *saml2.demo*, открываем и редактируем его.



Enable Assertion Signing - признак говорит о том, что исходящие от *IdP* сообщения будут **внутри** подписаны цифровой подписью формата XMLDSIG. Наше приложение может проверять корректность этой подписи.

Enable Signature Validation in Authentication Requests and Logout Requests - признак говорит о том, что приходящие на *IdP* запросы на вход и выход, будут проверяться по сигнатуре SimpleSign.

Certificate Alias - наименование сертификата, через который будут проверяться входящие запросы. Для проверки поставим “wso2carbon”.

Теперь выгрузим этот сертификат из хранилища сертификатов *WSO2IS*

```
keytool -importkeystore -srckeystore ~/wso2is-4.6.0/repository/resources/security/wso2carbon.jks -storepass wso2car
```

Достанем из сертификата закрытый ключ. При выгрузке потребуется ввести пароль *wso2carbon*.

```
openssl pkcs12 -in wso2carbon.p12 -nocerts -nodes | openssl rsa > privkey.pem
```

Достанем также сертификат, чтобы получить из него публичный ключ. При выгрузке потребуется ввести пароль *wso2carbon*.

```
openssl pkcs12 -in wso2carbon.p12 -clcerts -nokeys | openssl x509 -pubkey -noout > pubkey.pem
```

Теперь для проверки работы с подписями добавим выгруженные ключи в настройки settings.py и укажем признаки необходимость подписи (‘signing’) и проверки подписи (‘validate’):

```
SSO_CONFIG = {

....

   'signing': True,
   'validate': True,
   'public_key': '''-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCUp/oV1vWc8/TkQSiAvTousMzO
M4asB2iltr2QKozni5aVFu818MpOLZIr8LMnTzWllJvvaA5RAAdpbECb+48FjbBe
0hseUdN5HpwvnH/DW8ZccGvk53I6Orq7hLCv1ZHtuOCokghz/ATrhyPq+QktMfXn
RS4HrKGJTzxaCcU7OQIDAQAB
-----END PUBLIC KEY-----''',
   'private_key': '''-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCUp/oV1vWc8/TkQSiAvTousMzOM4asB2iltr2QKozni5aVFu81
8MpOLZIr8LMnTzWllJvvaA5RAAdpbECb+48FjbBe0hseUdN5HpwvnH/DW8ZccGvk
53I6Orq7hLCv1ZHtuOCokghz/ATrhyPq+QktMfXnRS4HrKGJTzxaCcU7OQIDAQAB
AoGAS/+ooju4a9po67zIGTEkqrQmsJC1HAPZo0bOmQK38LRzcps8Bmao9tjjbuVq
ogEj2xgjtHyNPSn3oBUA3v33usJ6YqwVrWsC6FwmZhq8Avsf94qm4hiTHe1AdxWm
ZGTs1eSYc6JnPIp0iVjHEfssIlGN+7LX1Q6kdbCf482dTnUCQQDvLwmtjlUASW84
zL5PEnNCorlcJ8qjGKlbcur2Lrn3vSCyX4cIWMxPNsCGvS2IO1Ctmz7yssnobhX6
iOaFOZVPAkEAnxuSwN4Kdw9Zku8cc7aifnJuEjzuEemM1cmwGSqilL0xUijVeeyq
fyy+1o7VFDa/nWPmmEZSqPNR6utcvLQU9wJAIycmpPtmQsSINDDjR3vOtNx1obW3
```

```
coENYwNgxQ3ZBzAkvhKMJg3m+T1yzlq/dmZBVUKb3c+pHSAQ2uGD/9CWwQJAVRy4
6ndc/ce2UQWcIMJINoAcJaF2cRqQfiTAERZfllWGtr6lQ+24XwOeqsQJdCC9bAJu
7nJf8YUIAzUYjNGAjQJBAKskkwcdhzvVcs7llm3+wWEzbMXzvNBmkZGRhDX6jtUI
J4U9RTHivqMeym4vpOmggaD4zc8qzG1NPDOpOp5AxBg=
-----END RSA PRIVATE KEY-----''',
}
```

Приложение должно работать как прежде.

4.4 Бэкенд соответствия сессий

В зависимости от необходимости, соответствие django-сессий и SSO-сессий можно хранить разными способами. За это отвечает настройка 'session_map' в *SSO_CONFIG*. В ней указывается бэкенд, реализующий хранение. В модуле доступны два бэкенда:

- 'ssosp.backends.cache' - хранение в кэше
- 'ssosp.backends.db' - хранение в базе данных

Бэкенд соответствия сессий используется только при входе в систему и при выходе из системы. Поэтому, на к нему не предъявляются требования по скорости работы - эти события не частые. Но к бэкенду предъявляется требование по обеспечению единого хранилища в случае масштабирования приложения. Поэтому, если приложение распределенное, то следует выбирать и настраивать бэкенд соответствия учитывая это.

При хранении соответствия в кэше, используется настроенное django.core.cache хранилище. Соответствие идентификаторов сессий хранятся в ключах с префиксом 'ssosessionmap.cache.sso' для SSO-сессии и 'ssosessionmap.cache.django' для django-сессии.

При хранении соответствия в базе данных, используется django-модель SSOSession.

Состав модуля

5.1 ssosp.assertion_parser

Низкоуровневая работа с SAML-утверждениями (Assertion) в xml виде с использованием lxml.etree

`ssosp.assertion_parser.assertion_to_xml(`*assertion*`)`
Переобразование утверждения в xml-строку

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат Утверждение представленное строкой

Тип результата basestring

`ssosp.assertion_parser.build_assertion(`*assertion_struct*`)`
Создание нового утверждения по структуре

Параметры `assertion_struct` (*dict*) – словарь структуры из которого строится утверждение

Результат Сформированное утверждение (Assertion, xml)

Тип результата etree.ElementTree

`ssosp.assertion_parser.get_attributes_from_assertion(`*assertion*`)`
Получение атрибутов из утверждения

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат список атрибутов

Тип результата list

`ssosp.assertion_parser.get_session_from_request_assertion(`*assertion*`)`
Получение ID сессии из запроса на выход (утверждения)

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат идентификатор сессии

Тип результата basestring

`ssosp.assertion_parser.get_session_from_response_assertion(`*assertion*`)`
Получение ID сессии из ответа (утверждения) на запрос

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат идентификатор сессии

Тип результата basestring

ssosp.assertion_parser.get_userid_from_assertion(*assertion*)
Получение ID пользователя из ответа (утверждения) на запрос

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат идентификатор пользователя

Тип результата basestring

ssosp.assertion_parser.is_logout_request(*assertion*)
Проверка того, что это запрос (утверждение) на выход из системы

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат True, если запрос на выход

Тип результата bool

ssosp.assertion_parser.is_logout_response(*assertion*)
Проверка того, что это ответ (утверждение) на запрос на выход из системы

Параметры `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат True, если ответ на запрос на выход

Тип результата bool

ssosp.assertion_parser.sign_request(*message*, *private_key_str*)
Цифровая подпись SAML-сообщения. Получение сигнатуры по алгоритму SHA1

Параметры

- `message` (*basestring*) – Сообщение для подписи
- `public_key_str` (*basestring*) – закрытый ключ для подписи представленный в виде строки

Результат строка сигнатуры подписи закодированная в base64

Тип результата basestring

ssosp.assertion_parser.verify_assertion(*assertion*, *public_key_str*)
Проверка цифровой подписи утверждения по публичному ключу

Параметры

- `assertion` (*etree.ElementTree*) – Утверждение (Assertion, xml)
- `public_key_str` (*basestring*) – публичный ключ подписи представленный в виде строки

Результат признак успешной проверки подписи

Тип результата bool

Raise XMLSigException - ошибка при проверке подписи

ssosp.assertion_parser.xml_to_assertion(*xml_string*)
Переобразование xml-строки в утверждение для дальнейшей работы

Параметры `xml_string` (*basestring*) – Утверждение, представленное строкой

Результат Преобразованное утверждение (Assertion, xml)

Тип результата etree.ElementTree

5.2 ssosp.models

Модель хранения соответствия SSO-сессии и django-сессии

class ssosp.models.SSOSession(**args*, ***kwargs*)
Модель хранения соответствия ключа SSO-сессии и django-сессии

5.3 ssosp.request_response

Классы SAML-запросов и ответов

class ssosp.request_response.AuthRequest(*request*)
SAML-запрос на вход в систему

get_login(*next_url*)
Получить GET-запрос на вход в систему.

Параметры next_url (*basestring*) – адрес, на который вернуться после входа

Результат редирект на адрес SSO с SAML-запросом на вход в качестве параметра

Тип результата django.http.HttpResponseRedirect

get_request()
Получить SAML-запрос. Формируется утверждение AuthnRequest и преобразовывается в строку.

Результат Утверждение для входа в систему, представленное в виде строки

Тип результата basestring

class ssosp.request_response.AuthResponse(*request*)
SAML-ответ на запрос аутентификации

do_login(*request*, *next_url*)
Выполнить вход в систему. Атрибуты пользователя сохраняются в сессию. Сохраняется соответствие SSO-сессии и django-сессии.

Параметры

- request (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие
- next_url (*basestring*) – адрес, на который вернуться после входа

Результат ответ на запрос - редирект на адрес возврата

Тип результата django.http.HttpResponseRedirect

from_assertion(*assertion*)
Заполнить из утверждения. Загрузить. Определяется сессия, пользователь, атрибуты пользователя.

Параметры assertion (*etree.ElementTree*) – Утверждение, из которого надо получить данные

class ssosp.request_response.LogoutRequest(*request*)
SAML-запрос на выход из системы

do_logout(*request*)
Осуществить выход из систему по текущей сессии django. Сессия получается из текущего запроса.

Параметры `request` (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие

`do_logout_by_session(request)`
Осуществить выход из систему по сессии SSO.

Параметры `request` (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие

`from_assertion(assertion)`
Заполнить из утверждения. Загрузить. Просто проверим цифровую подпись, если надо. И вытащим сессию, если ее передали в утверждении.

Параметры `assertion` (*etree.ElementTree*) – Утверждение, из которого надо получить данные

`get_logout(username, next_url)`
Получить GET-запрос на выход из системы.

Параметры

- `username` (*basestring*) – пользователь, который должен выйти из системы. (Похоже, что уже не надо использовать)
- `next_url` (*basestring*) – адрес, на который вернуться после входа

Результат редирект на адрес SSO с SAML-запросом на выход в качестве параметра

Тип результата django.http.HttpResponseRedirect

`get_request(username)`
Получить SAML-запрос на выход. Формируется утверждение LogoutRequest и преобразовывается в строку. Используется текущая сессия: либо загруженная, либо определенная из request.

Параметры `username` (*basestring*) – пользователь, который должен выйти из системы. (Похоже, что уже не надо использовать)

Результат Утверждение для выхода из системы, представленное в виде строки

Тип результата basestring

class `ssosp.request_response.LogoutResponse(request)`
SAML-ответ на запрос выхода из системы

`do_logout(request, next_url)`
Выполнить выход из системы. Удаляется соответствие SSO-сессии и django-сессии.

Параметры

- `request` (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие
- `next_url` (*basestring*) – адрес, на который вернуться после выхода

Результат ответ на запрос - редирект на адрес возврата

Тип результата django.http.HttpResponseRedirect

`from_assertion(assertion)`
Заполнить из утверждения. Загрузить. Просто проверим цифровую подпись, если надо.

Параметры `assertion` (*etree.ElementTree*) – Утверждение, из которого надо получить данные

class ssosp.request_response.SAMLObject(*request*)
Базовый класс SAML-объекта

exception ssosp.request_response.SSOException
Класс исключений работы с SSO

ssosp.request_response.get_method(*method_str*)
Получение функции, представленной строкой с полным путем

Параметры method_str (*basesting*) – полный путь к функции

Результат указатель на функцию или None, если строка пустая

ssosp.request_response.get_request_from_data(*request*, *xml_string*)
Получить утверждение-запрос из xml-строки. Используется для определения объекта в сервисе ACS.

Параметры

- request (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие
- xml_string (*basestring*) – xml-строка, содержащая утверждение

Результат объект Request, преобразованный из xml-строки

Тип результата LogoutRequest | None

ssosp.request_response.get_response_from_data(*request*, *xml_string*)
Получить утверждение-ответ из xml-строки. Используется для определения объекта в сервисе ACS.

Параметры

- request (*django.http.HttpRequest*) – запрос, в рамках которого выполняется действие
- xml_string (*basestring*) – xml-строка, содержащая утверждение

Результат объект Response, преобразованный из xml-строки

Тип результата либо LogoutResponse, либо AuthResponse

ssosp.request_response.get_session_map()
Получение бэкенда хранения соответствия сессий, указанного в настройке SSO_CONFIG['session_map']

Результат Экземпляр бэкенда, наследника BaseSSOSessionMap

Тип результата *ssosp.backends.base.BaseSSOSessionMap*

ssosp.request_response.get_str_from_assertion(*assertion*)
Преобразовать утверждение в строку перекодированную и упакованную.

Параметры assertion (*etree.ElementTree*) – Утверждение (Assertion, xml)

Результат xml-строка, содержащая утверждение

Тип результата basestring

5.4 ssosp.urls

Регистрация обработчиков url-адресов для django.

Регистрируются адреса:

- acs - ssosp.views.sso_acs
- login - ssosp.views.sso_login
- logout - ssosp.views.sso_logout

5.5 ssosp.utils

Вспомогательные утилиты

Часть утилит взята отсюда: http://stackoverflow.com/questions/1089662/python-inflate-and-deflate-implementations

`ssosp.utils.decode_base64(`*b64string*`)`
Разкодировать из base64

Параметры `b64string` (*basestring*) – исходная строка в формате base64

Результат раскодированная строка

Тип результата basestring

`ssosp.utils.decode_base64_and_inflate(`*b64string*`)`
Разкодировать из base64 и разжать zip

Параметры `b64string` (*basestring*) – исходная строка в формате base64

Результат раскодированная и распакованная строка

Тип результата basestring

`ssosp.utils.deflate_and_base64_encode(`*string_val*`)`
Сжать zip и закодировать в base64

Параметры `string_val` (*basestring*) – исходная строка

Результат запакованная и закодированная в base64 строка

Тип результата basestring

`ssosp.utils.get_random_id()`
Генерация случайного идентификатора UUID. Начинается с символа "_".

Результат идентификатор в виде строки

Тип результата basestring

`ssosp.utils.get_time_string(`*delta=0*`)`
Представить текущее время в виде строки с учетом дельты.

Параметры `delta` (*int*) – дельта времени

Результат текущее время с дельтой в виде строки

Тип результата basestring

5.6 ssosp.views

Обработчики адресов сервисов по умолчанию

ssosp.views.sso_acs(*request*)
Assertion Consumer Service

Приемник ответов и запросов при взаимодействии для SSO посредством SAML 2.0

Параметры request (*django.http.HttpRequest*) – входящий запрос

Результат ответ на запрос - обычно редирект на адрес SSO или адрес переданный через POST-параметр "RelayState"

Тип результата django.http.HttpResponseRedirect | django.http.HttpResponse

ssosp.views.sso_login(*request*, *next_url=None*)
Подготовка запроса на Identity Provider для входа в систему

Параметры

- request (*django.http.HttpRequest*) – входящий запрос
- next_url (*basestring*) – следующий адрес после обработки

Результат ответ на запрос - обычно редирект на адрес SSO или адрес переданный next_url

Тип результата django.http.HttpResponseRedirect

ssosp.views.sso_logout(*request*, *next_url=None*)
Подготовка запроса на Identity Provider для выхода из системы

Параметры

- request (*django.http.HttpRequest*) – входящий запрос
- next_url (*basestring*) – следующий адрес после обработки

Результат ответ на запрос - обычно редирект на адрес SSO или адрес переданный next_url

Тип результата django.http.HttpResponseRedirect

5.7 ssosp.backends

Базовый класс для бэкенда соответствия

class ssosp.backends.base.BaseSSOSessionMap
Интерфейс бэкенда соответствия сессий

delete_by_django_session(*django_session_key*)
Удалить соответствие по идентификатору django-сессии

Параметры django_session_key (*basestring*) – идентификатор django-сессии

delete_by_sso_session(*sso_session_key*)
Удалить соответствие по идентификатору SSO-сессии

Параметры sso_session_key (*basestring*) – идентификатор SSO-сессии

exists_django_session(*django_session_key*)
Проверить существование идентификатора django-сессии

Параметры django_session_key (*basestring*) – идентификатор django-сессии

Результат признак существования идентификатора

Тип результата bool

exists_sso_session(*sso_session_key*)
Проверить существование идентификатора SSO-сессии

Параметры sso_session_key (*basestring*) – идентификатор SSO-сессии

Результат признак существования идентификатора

Тип результата bool

get_django_session_key(*sso_session_key*)
Получить идентификатор django-сессии по идентификатору SSO-сессии

Параметры sso_session_key (*basestring*) – идентификатор SSO-сессии

Результат идентификатор django-сессии

Тип результата basestring

get_sso_session_key(*django_session_key*)
Получить идентификатор SSO-сессии по идентификатору django-сессии

Параметры sso_session_key (*basestring*) – идентификатор django-сессии

Результат идентификатор SSO-сессии

Тип результата basestring

set_session_map(*sso_session_key*, *django_session_key*)
Установить соответствие идентификатора SSO-сессии и django-сессии

Параметры

- sso_session_key (*basestring*) – идентификатор SSO-сессии
- django_session_key (*basestring*) – идентификатор django-сессии

class ssosp.backends.cache.SSOSessionMap
Кэш-бэкенд хранения соответствия сессий

class ssosp.backends.db.SSOSessionMap
Бэкенд хранения соответствия сессий в базе данных

Глава 6

Дополнительно

- genindex
- modindex
- search

S

A

B

D

E

F

G

I

L

S

V

X