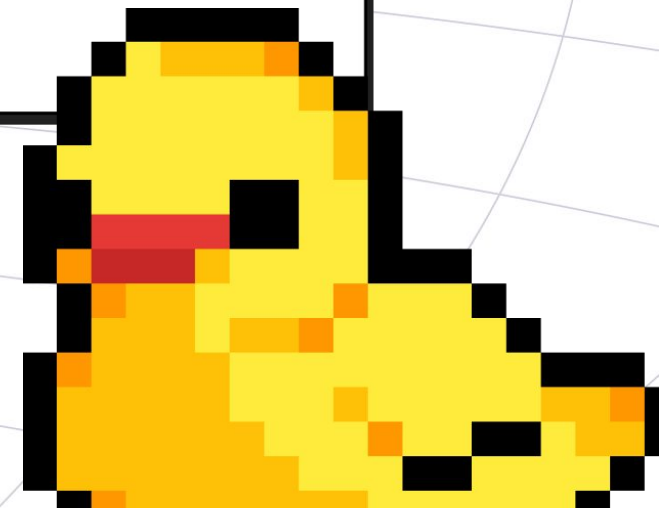




USE TECH

Python + Keycloak: доверь аутентификацию профессионалам

Казakov Мстислав
Системный архитектор





ПРО СЕБЯ



Стаж: +10 лет

Компания: ГК Юзтех

Должности: тим- тех- лид, руководитель Python-практики, системный архитектор

Сферы: финтех, гостех, медтех, сельское хозяйство, etc.



ПЛАН



Классика



**Проблемы
классики**



**Зрелые
решения**



Понятия



**Настраиваем
Keycloak**



**Интегрируем
Keycloak**



Аутентификация в Django



- DRF + Simple JWT: аутентификация
- Django All Auth: подключение внешних Identity Providers
- Django Auth LDAP: для интеграции со службой каталогов по LDAP протоколу
- ADFS, Kerberos: ...



Прочее в Django



- Кастомные DRF эндпоинты: восстановление пароля, подтверждение email'a, etc.
- Настройка политик пароля с помощью Django Password Validators
- Django Axes для ограничения попыток ввода неверного пароля



Вероятные проблемы



- Отсутствие «золотого» набора решений
- Дублирование функционала
- Повторяющаяся работа
- Информационная безопасность
 - OWASP TOP 10: A07:2021-Identification and Authentication Failures



И что же делать?



- Аутентификация, авторизация и связанные функции стоят отдельных подсистем
- Доверьте аутентификацию профессионалам



Стандарты



- ГОСТ Р 59383 — 2021 МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ
Основы управления доступом (ISO/IEC 29146:2016, NEQ)
- RFC 6749: The OAuth 2.0 Authorization Framework
- OpenID Connect Core 1.0
- RFC 7519 - JSON Web Token (JWT)
- RFC 7515 - JSON Web Signature (JWS)



JWT



- Структура данных с информацией (claims) о пользователе и электронной подписью для проверки целостности и авторства JWS (JSON Web Signature RFC 7515)
- Без дополнительных действий полезная нагрузка лишь ИДЕНТИФИЦИРУЕТ пользователя
- После проверки подписи JWT-токена пользователя можно считать аутентифицированным



- JSON Web Signature
- Преобразование хеша данных с помощью криптографической функции
- Криптография с открытым и закрытым ключом (асимметричное шифрование)
 - Например RS256
- Криптография с закрытым ключом (shared secret)
 - Например HMAC
- Не храним токен в БД Django или любом другом персистентном хранилище



JWT



Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiNj0iLCJ0eXAiOiJKV1QiLCJkaXI6ImNpdRkdVFUsra2rWKEofkZeIC4yWytE58sMIihvo9H1ScmmVwBcQP6XETqYd0aSHp1g0a9RdUPDvoXQ5oqygTqVtxaDr6wUFKrKItgBMzWIdNZ6y709E0DhEPTbE9rfBo6KTFsHAZnMg4k68CDp2woYIaXbmYTWcvbzIuH07_37GT79XdIwkm95QJ7hYC9RiwrV7mesbY4PAahERJawntho0my942XheVLmGwLMBkQ
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "RS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "John Doe",  "admin": true,  "iat": 1516239022}
```

VERIFY SIGNATURE



По понятиям



- Идентификация: кто ты?
- Аутентификация: а точно ли ты тот, за кого себя выдаёшь?
- Авторизация: а можешь ли ты совершать запрошенное действие?

АВТОРИЗАЦИЯ – ЭТО НЕ ВВОД ЛОГИНА И ПАРОЛЯ



On Site

- Keycloak
- Open IAM
- Gluu
- Dex + FreeIPA



Подробнее о Keycloak



Keycloak относится к классу систем Identity and Access Management (IAM)

Что умеет:

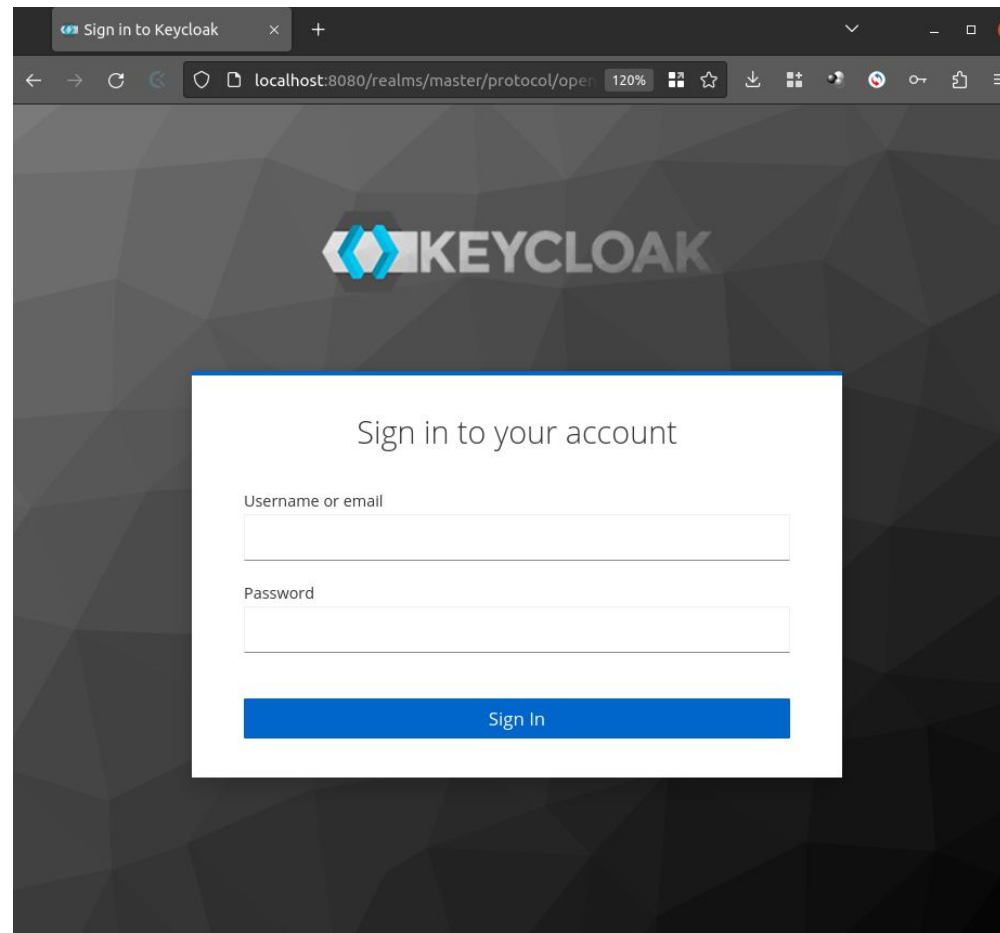
- Управление пользователями, ролями
- Регистрация, восстановление пароля, изменение личных данных
- Аутентификация в том числе MFA и выдача токенов доступа
- Синхронизация учётных записей с серверами каталогов
- Аутентификация “через социальные сети”
- Поддерживает протоколы OAuth2, OIDC, SAML (здравствуй, ADFS из коробки), etc.



Keycloak: поднимаем локально



```
compose.yml
1  services:
2    keycloak:
3      environment:
4        KC_DB: dev-file
5        KEYCLOAK_ADMIN: admin
6        KEYCLOAK_ADMIN_PASSWORD: admin
7      image: quay.io/keycloak/keycloak:22.0.1
8      command: start-dev
9      ports:
10     - 8080:8080
11     restart: unless-stopped
12
```





Keycloak: что мы видим?



The screenshot displays the Keycloak Admin Console interface. On the left is a dark sidebar with a hamburger menu icon and the 'KEYCLOAK' logo. Below the logo is a dropdown menu showing 'master'. The sidebar contains the following menu items: Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'master realm' and has two tabs: 'Server info' (active) and 'Provider info'. The 'Server info' tab is divided into three sections: 'Server info', 'Memory', and 'Profile'. The 'Server info' section shows 'Version: 22.0.1' and 'Product: Default'. The 'Memory' section shows 'Total memory: 455 MB', 'Free memory: 397 MB', and 'Used memory: 58 MB'. The 'Profile' section shows 'Enabled features' and 'Disabled features'. The disabled features are listed as follows: ACCOUNT3 (Preview), ADMIN_FINE_GRAINED_AUTHZ (Preview), DECLARATIVE_USER_PROFILE (Preview), DOCKER (Supported), MAP_STORAGE (Experimental), RECOVERY_CODES (Preview), and UPDATE_EMAIL (Preview).

master

master realm

Server info Provider info

Server info

Version
22.0.1

Product
Default

Memory

Total memory
455 MB

Free memory
397 MB

Used memory
58 MB

Profile

Enabled features ⓘ

Disabled features ⓘ

ACCOUNT3 Preview ADMIN_FINE_GRAINED_AUTHZ Preview

DECLARATIVE_USER_PROFILE Preview DOCKER Supported

MAP_STORAGE Experimental RECOVERY_CODES Preview S

UPDATE_EMAIL Preview



Keycloak: настройки Client



Используем Authorization Code Flow

[Clients](#) > Client details

django OpenID Connect

Clients are applications and services that can request authentication of a user.

Settings

Roles

Client scopes

Sessions

Advanced

General Settings

Client ID * ?

Name ?



Keycloak: настройки Client



Используем Authorization Code Flow

Access settings


Root URL 

Home URL 

Valid redirect URIs 



[+ Add valid redirect URIs](#)

Valid post logout
redirect URIs 



[+ Add valid post logout redirect URIs](#)

Web origins 



[+ Add web origins](#)



Keycloak: настройки Client



Используем Authorization Code Flow

Capability config

Client authentication Off [?](#)

Authorization Off [?](#)

Authentication flow


- Standard flow [?](#)
- Direct access grants [?](#)
- Implicit flow [?](#)
- Service accounts roles [?](#)
- OAuth 2.0 Device Authorization Grant [?](#)
- OIDC CIBA Grant [?](#)



Keycloak: настройки OpenID Connect (OIDC)



- Configure
- Realm settings
- Authentication
- Identity providers
- User federation

Endpoints 

[OpenID Endpoint Configuration](#) 

[SAML 2.0 Identity Provider Metadata](#) 

Save

Revert

```
localhost:8080/realms/master/.well-known/openid-configuration
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
issuer: "http://localhost:8080/realms/master"
authorization_endpoint: "http://localhost:8080/realms/master/protocol/openid-connect/auth"
token_endpoint: "http://localhost:8080/realms/master/protocol/openid-connect/token"
introspection_endpoint: "http://localhost:8080/realms/master/protocol/openid-connect/token/introspect"
userinfo_endpoint: "http://localhost:8080/realms/master/protocol/openid-connect/userinfo"
end_session_endpoint: "http://localhost:8080/realms/master/protocol/openid-connect/logout"
frontchannel_logout_session_supported: true
frontchannel_logout_supported: true
jwks_uri: "http://localhost:8080/realms/master/protocol/openid-connect/certs"
check_session_iframe: "http://localhost:8080/realms/master/protocol/openid-connect/login-status-iframe.html"
grant_types_supported:
```



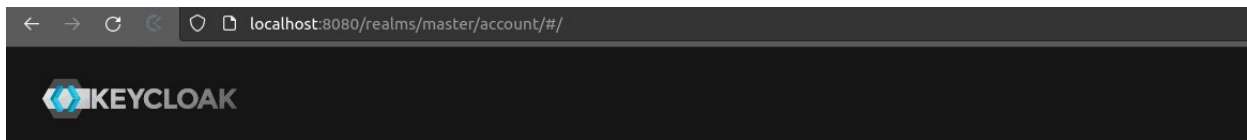
Keycloak: настройки OpenID Connect (OIDC)






```
localhost:8080/realms/master

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

realm: "master"
public_key: "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy0IkgtiuPgWfxiN8LQ6ZKRwQAA7CKKZp4SH78pyu8WXMfHV6YCYnfw2aVwz6g1A41sgcb/ndva1GU8TrDPzd8FhF6zAYWTccw+pNcY3LQ4pYvH6P+0Fr1F+/rTKPc0KaXyY9BbQ4ta1Krc0PS3SmMGXUsb6bf31UGto0r0h5AS3HG3suMqHYcX4XxzpeCX7Dp8PHA+HjJ0giDXd/GS836aPPk37cf4+PwMtIM5UYKWDDiIAuL7i0SHf0J4ddLzwab4IRpdwgqCUXoJHbxG8NddCkZL24cQms2vLiA579GshYvrxJxL4oRwec8YYjdMExt0kx4tqqYVxSSxvoxB/4CQIDAQAB"
token-service: "http://localhost:8080/realms/master/protocol/openid-connect"
account-service: "http://localhost:8080/realms/master/account"
tokens-not-before: 0
```



Welcome to Keycloak account management

 Personal info Manage your basic information Personal info	 Account security Control your password and account access Signing in Device activity	 Applications Track and manage your app permission to access your account Applications
--	--	--



Django



1. Сессионная аутентификация: обмен JWT на Django Session
2. Stateless токенная аутентификация: токен в каждом запросе, записи в БД не создаются
3. Statefull токенная аутентификация: токен в каждом запросе, записи в БД создаются



Django: сессионная аутентификация



```
# Настройка интеграции с Keycloak на основе сессий
if kc_type == "session":
    INSTALLED_APPS.append("mozilla_django_oidc")

    LOGIN_REDIRECT_URL = "/"
    LOGOUT_REDIRECT_URL = "/"

    AUTHENTICATION_BACKENDS = (
        "mozilla_django_oidc.auth.OIDCAuthenticationBackend",
    )
    MIDDLEWARE.append("mozilla_django_oidc.middleware.SessionRefresh")

KEYCLOAK_BASE_URL = "http://keycloak:8080/"
KEYCLOAK_REALM = "master"

OIDC_OP_JWKS_ENDPOINT = f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/certs"
OIDC_OP_AUTHORIZATION_ENDPOINT = f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/auth"
OIDC_OP_TOKEN_ENDPOINT = f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/token"
OIDC_OP_USER_ENDPOINT = f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/userinfo"

# Не храните эти значения в системе контроля версий!
OIDC_RP_CLIENT_ID = "django"
OIDC_RP_SIGN_ALGO = "RS256"
OIDC_RP_CLIENT_SECRET = ""
```

```
# Настройка интеграции с Keycloak
path('oidc/', include('mozilla_django_oidc.urls')),
```



Django: сессионная аутентификация



```
class OIDCAuthenticationBackend(OIDCAuthenticationBackend):
    def get_username(self, claims):
        """Использовать в качестве username UUID пользователя из Keycloak

        Если изменить email в keycloak, то при входе в Django будет создан новый
        пользователь. Однозначно мэтчим пользователя по UUID, чтобы избежать этого поведения
        """
        username_algo = self.get_settings("OIDC_USERNAME_ALGO", None)

        if username_algo:
            if isinstance(username_algo, str):
                username_algo = import_string(username_algo)
                return username_algo(claims.get("email"))

        # UUID instead of email
        return claims.get("sub")

    def update_user(self, user, claims):
        # TODO: см. в сторону keycloak spi event listener и http вебхуков
        changed = False
        claims_checklist = [("email", "email")]
        print(claims)
        for claim_field, user_field in claims_checklist:
            if not getattr(user, user_field) != claims.get(claim_field):
                continue

            setattr(user, user_field, claims.get(claim_field))
            changed = True

        if changed:
            user.save()
            user.refresh_from_db()

        return user
```




Django: stateless токенная аутентификация



```
# Настройка интеграции с Keycloak на основе токенов (stateless)
if kc_type == "stateless_token":
    SIMPLE_JWT = {
        "ALGORITHM": "RS256",
        "JWK_URL": f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/certs"
    }
    INSTALLED_APPS.append("rest_framework_simplejwt")
    REST_FRAMEWORK["DEFAULT_AUTHENTICATION_CLASSES"].append("rest_framework_simplejwt.authentication.JWTStatelessUserAuthentication")
```



Django: stateful токенная аутентификация



```
# Настройка интеграции с Keycloak на основе токенов (Stateful)
if kc_type == "stateful_token":
    SIMPLE_JWT = {
        "ALGORITHM": "RS256",
        "JWK_URL": f"{KEYCLOAK_BASE_URL}realms/{KEYCLOAK_REALM}/protocol/openid-connect/certs"
    }
    INSTALLED_APPS.append("rest_framework_simplejwt")
    REST_FRAMEWORK["DEFAULT_AUTHENTICATION_CLASSES"].append("rest_framework_simplejwt.authentication.JWTAuthentication")
```

```
class JWTAuthentication(_JWTAuthentication):

    def get_user(self, validated_token):
        try:
            user_id = validated_token[settings.USER_ID_CLAIM]
            user = self.user_model.objects.get_or_create(**{settings.USER_ID_FIELD: user_id})
            # <UPDATE ПОЛЬЗОВАТЕЛЯ ПИСАТЬ ТУТ>
        except self.user_model.DoesNotExist:
            raise AuthenticationFailed(_("User not found"), code="user_not_found")

        return user
```

○○○ **А что же с Flask, FastAPI, Tornado, Pyramid, BottlePy, etc.?** ☆

**Всё хорошо
Их хранит PyJWT**



Исходные коды примеров

