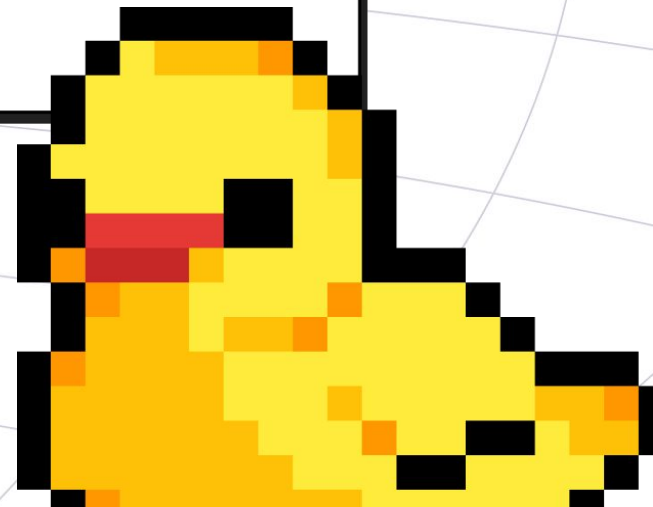
USETECH

Python + Keycloak: доверь аутентификацию профессионалам

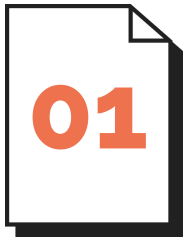
Казаков Мстислав
Системный архитектор



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'а, 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

JWS

- JSON Web Signature
- Преобразование хеша данных с помощью криптографической функции
- Криптография с открытым и закрытым ключом (ассиметричное 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.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUx
NjIzOTAyMn0.NHVaYe26MbtOYhSKkoKYdFVomg4
i8ZJd8_-
RU8VNbftc4TSMb4bXP3l3YlNWACwyXPGffz5aXH
c6lty1Y2t4SWRqGteragsVdZufDn5BlnJl9pdR_
kdVFUsra2rWKEofkZeIC4yWytE58sMIihvo9H1S
cmmVwBcQP6XETqYd0aSHp1gOa9RdUPDvoXQ5oqy
gTqVtxaDr6wUFKrKItgBMzWIdNZ6y7O9E0DhEPT
bE9rfBo6KTFsHAZnMg4k68CDp2woYIaXbmYTWcv
bzIuHO7_37GT79XdIwkm95QJ7hYC9RiwrV7mesb
Y4PAahERJawntho0my942XheVLmGwLMBkQ
```

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

Зрелые решения

On Site

- Keycloak
- Open IAM
- Gluu
- Dex + FreeIPA

Подробнее о Keycloak

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

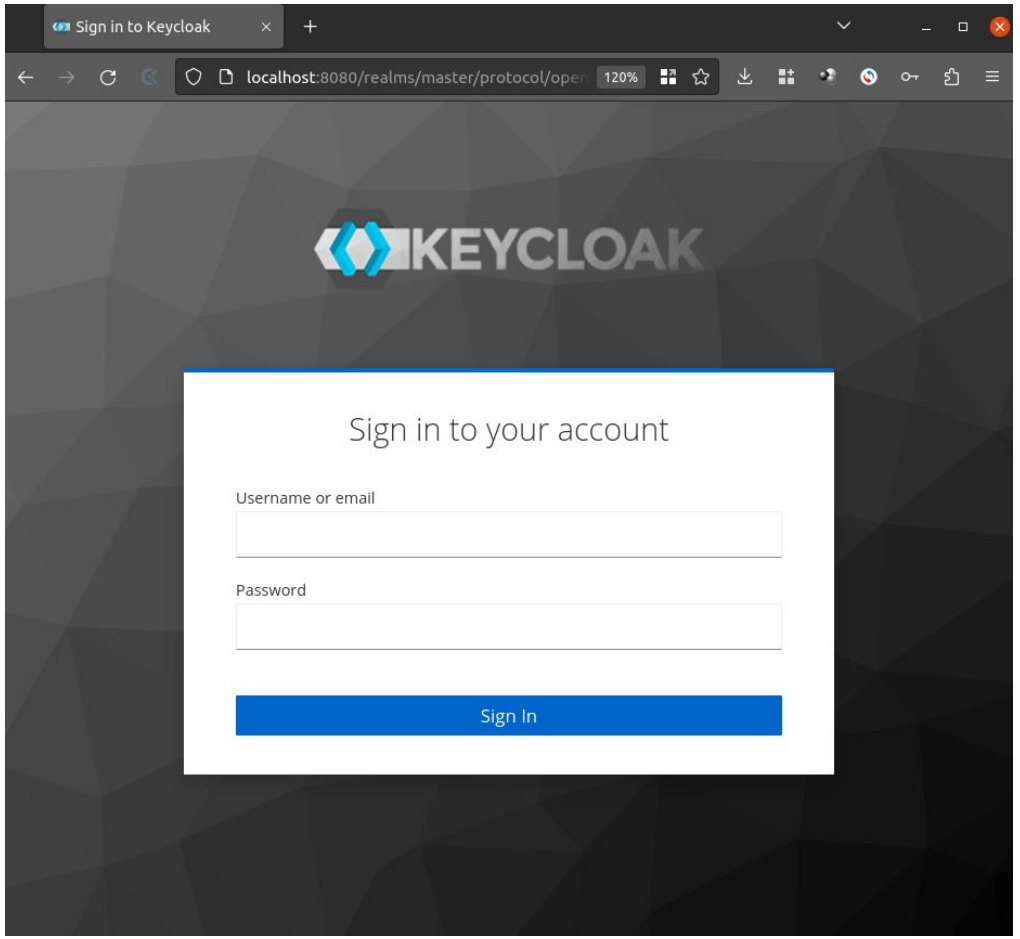
Что умеет:

- 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.

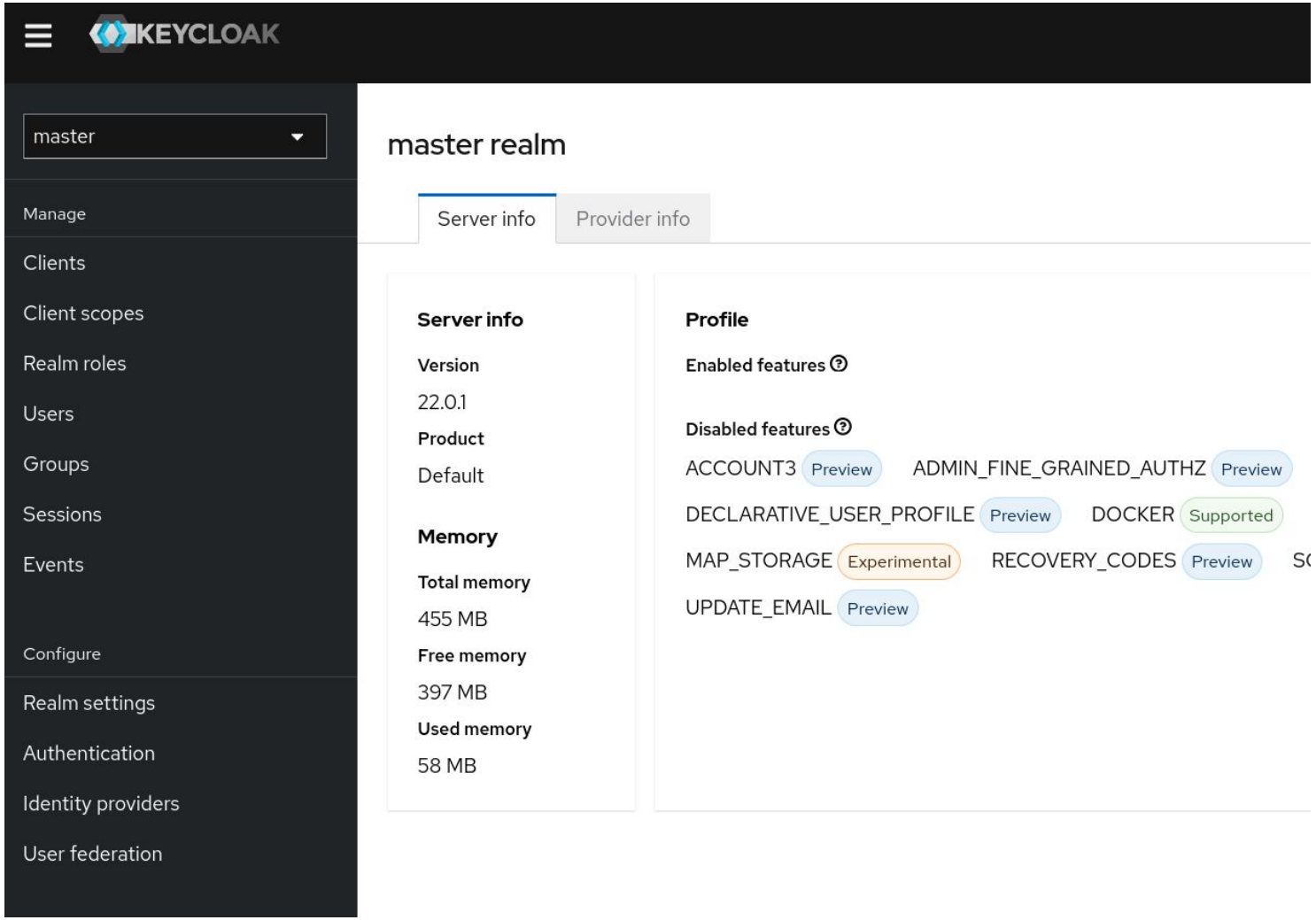
Keycloak: поднимаем локально

compose.yml

```yaml
services:
  keycloak:
    environment:
      KC_DB: dev-file
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
    image: quay.io/keycloak/keycloak:22.0.1
    command: start-dev
    ports:
      - 8080:8080
    restart: unless-stopped
```



Keycloak: что мы видим?



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 * : django

Name : Django

Keycloak: настройки Client

Используем Authorization Code Flow

Access settings

Root URL

Home URL

Valid redirect URIs: http://localhost:8000/*

Add valid redirect URIs

Valid post logout redirect URIs: http://localhost:8000/*

Add valid post logout redirect URIs

Web origins: *

Add web origins

Keycloak: настройки Client

Используем Authorization Code Flow

Capability config

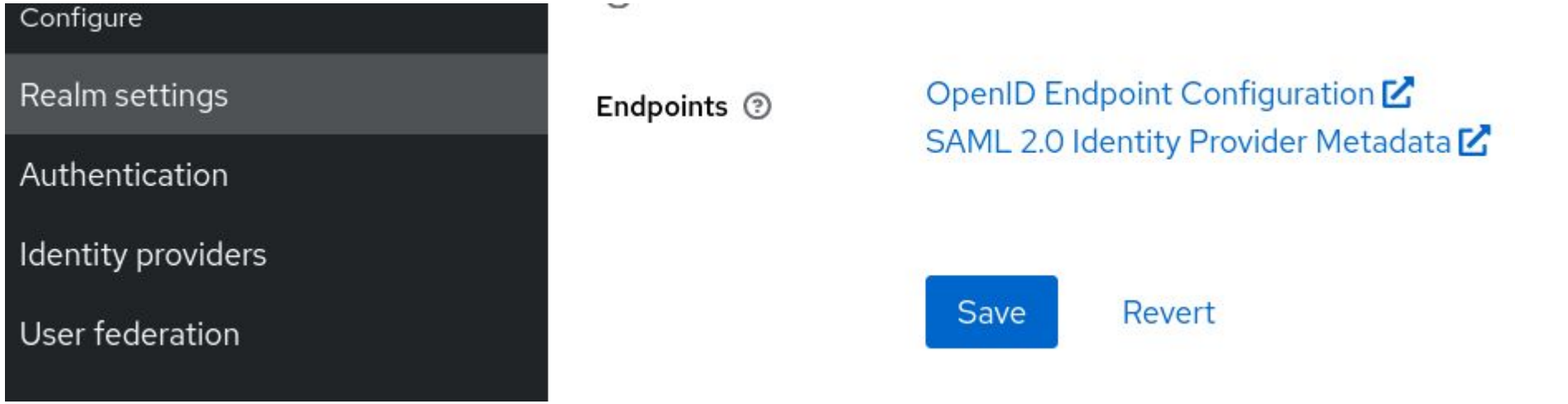
Client authentication Off

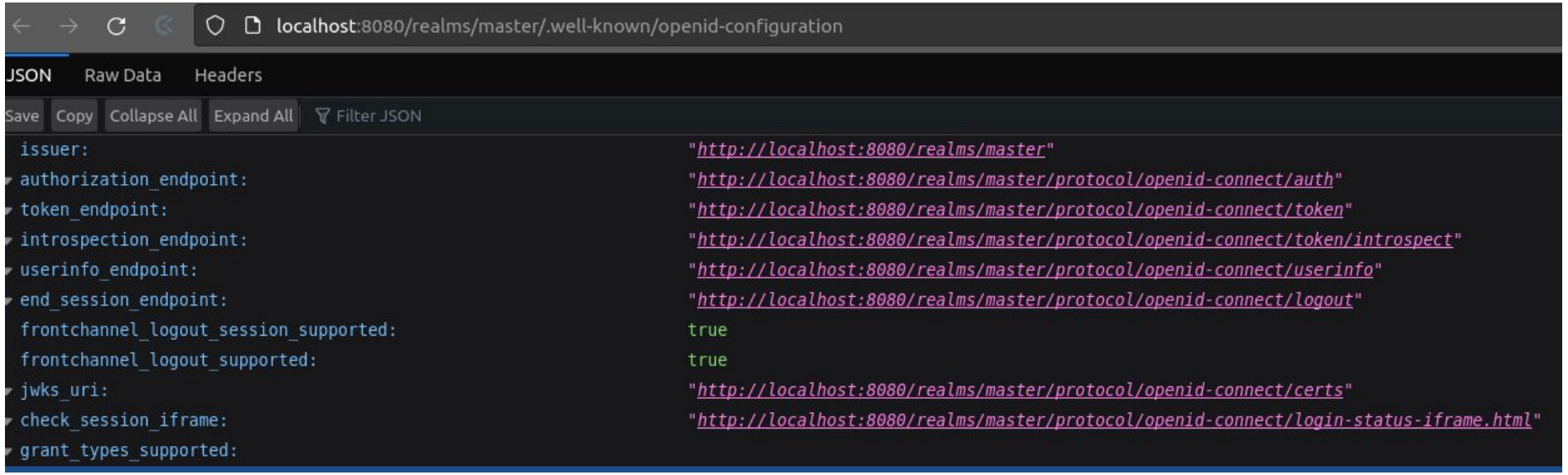
Authorization Off

Authentication flow

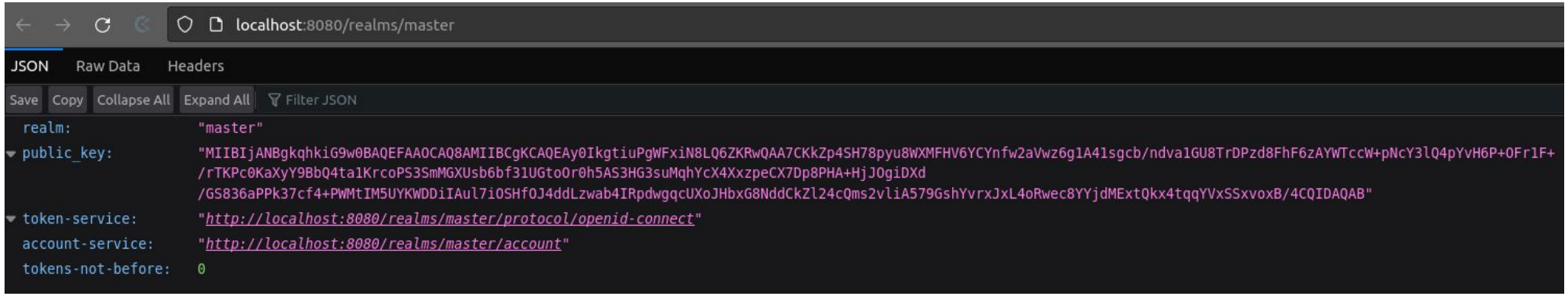
- [x] Standard flow
- [ ] Direct access grants
- [ ] Implicit flow
- [ ] Service accounts roles
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant

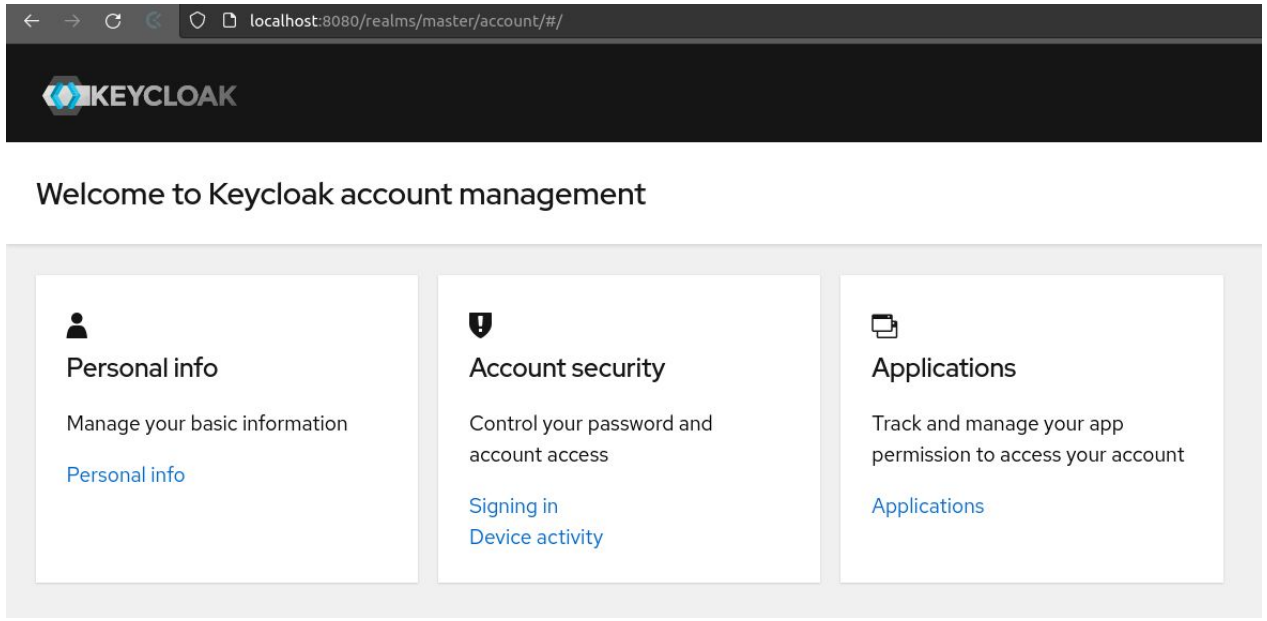
Keycloak: настройки OpenID Connect (OIDC)





Keycloak: настройки OpenID Connect (OIDC)





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

```python
# 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 = ""
```

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

Django: сессионная аутентификация

```python
class OIDCAuthenticationBackendUUID(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

```python
# 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

```python
# 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")
```

```python
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