?????????? ??????????

Почему-то для fastapi предпочтительнее использовать механизм Depends. Однако мне проще декораторы.

```
from functools import wraps

from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.concurrency import run_in_threadpool
from uvicorn import run
from typing import Optional

from keycloak import KeycloakOpenID
from jose import jwt
from jose.exceptions import JWTError, ExpiredSignatureError

app = FastAPI(docs_url=None, redoc_url=None, openapi_url=None)

KEYCLOAK_URL = "http://192.168.1.195:9090/"
REALM_NAME = "pythonsimpletest"
CLIENT_ID = "clientforsimpletest"
CLIENT_SECRET = "Vix6txRyHwt81KyZIpl7O06CxpWMFtib"
REDIRECT_URI = "http://192.168.1.3:8100/auth/callback"

KEYCLOAK_PUBLIC_KEY = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtrnsNk3z21imvfNFDT8teaekNgFhnOSuKcZiPc1Iv+qknDg8X1
zF7KJLtZMrCZSLRJM6T754F8KtNUSj0Ioa3ehYSPEdF4SX7tBdacpeDz0GlKWYFa845/e6H2349I+7EuO1bkHfWz/v6n2m
m+jeZKU0ujqr2boBoPWwkMoKwXNMl/Sac0YMlv1fVHiISyREDG+FordAwYlbLVYCD36ckk0UnAKBc59Q36DMiKSx7JpdvR
0vHIeWb4mQFlRbLdmWkK4ebUe+B/k1/cdd3LHdQ6vc1i45bMcJlrFYocDOGK99Mf8pT8OMPQvTJ8cTe9xSkCAx0l7YiSv+
5hfK1C6DswIDAQAB
-----END PUBLIC KEY-----"""
ALGORITHMS = ["RS256"]

# Инициализация Keycloak клиента
keycloak_openid = KeycloakOpenID(
    server_url=KEYCLOAK_URL,
    client_id=CLIENT_ID,
```

```
    realm_name=REALM_NAME,
    client_secret_key=CLIENT_SECRET,
)

# ============================Декораторы авторизации=======================
def verify_token(token: str):
    return jwt.decode(
        token,
        KEYCLOAK_PUBLIC_KEY,
        algorithms=ALGORITHMS,
        audience="account",
        options={
            "verify_signature": True,
            "verify_aud": False,
            "verify_exp": True,
        }
    )

def get_login_url():
    """Генерация URL для входа через Keycloak"""
    auth_url = keycloak_openid.auth_url(
        redirect_uri=REDIRECT_URI,
        scope="openid email profile",
        state="random_state_string"  # В реальном приложении используйте генерацию случайной
строки
    )
    return auth_url

def session_required(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        request: Request = kwargs.get("request")
        if request is None:
            for arg in args:
                if isinstance(arg, Request):
                    request = arg
                    break
        if request is None:
            raise RuntimeError("Request object not found")
```

```
        token = request.cookies.get("access_token")

        if not token:
            return RedirectResponse(get_login_url())

        try:
            claims = verify_token(token)
            request.state.user = claims
        except ExpiredSignatureError:
            response = RedirectResponse(get_login_url())
            response.delete_cookie("access_token")
            response.delete_cookie("refresh_token")
            return response
        except JWTError:
            response = RedirectResponse(get_login_url())
            response.delete_cookie("access_token")
            response.delete_cookie("refresh_token")
            return response
        return await func(*args, **kwargs)
    return wrapper

def html_norole():
    html_content = f'''<!DOCTYPE html><html><body><a href="/"><button>На главную</button></a>
    <p><a href="/logout"><button>Выход</button></a></p>
    <p>У вас нет доступа к этой странице</p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

def role_secondrole_required(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        request: Request = kwargs.get("request")
        if request is None:
            for arg in args:
                if isinstance(arg, Request):
                    request = arg
                    break
        if request is None:
            raise RuntimeError("Request object not found")
        userinfo = request.state.user
```

```
        roles = userinfo.get("realm_access", {}).get("roles", [])
        if 'secondrole' not in roles:
            return html_norole()
        return await func(*args, **kwargs)
    return wrapper
# ================================================================================

@app.get("/", response_class=HTMLResponse)
@session_required
async def simpleauth(request: Request):
    userinfo = request.state.user #await get_current_user_from_cookie(request)
    roles = userinfo.get("realm_access", {}).get("roles", [])

    html_content = f'''<!DOCTYPE html><html><body><a href="/logout"><button>Выход</button></a>
    <p>Доступные ключи: {userinfo}</p>
    <p>Доступные роли: {roles}</p>
    <p><a href="/secret"><button>Доступ к секрету</button></a></p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/secret", response_class=HTMLResponse)
@session_required
@role_secondrole_required
async def roleneeded(request: Request):
    html_content = f'''<!DOCTYPE html><html><body><a href="/logout"><button>Выход</button></a>
    <p><a href="/"><button>На домашнюю страницу</button></a></p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/auth/callback")
async def auth_callback(code: str):
    """Callback URL для обработки ответа от Keycloak после логина"""
    def append_cookie(response, key, value):
        response.set_cookie(
            key=key,
            value=value,
            httponly=True,  # Защита от XSS
            secure=False,   # True для HTTPS
            samesite="lax"
        )
```

```
        return response
    try:
        # Обмен кода на токены
        token_response = keycloak_openid.token(
            grant_type="authorization_code",
            code=code,
            redirect_uri=REDIRECT_URI
        )
        # Создаем редирект с токеном в заголовке (через установку cookie)
        response = RedirectResponse(url="/")
        access_token = token_response.get('access_token')
        # Устанавливаем токен в cookie (альтернатива Authorization header для браузера)
        response = append_cookie(response, "access_token", access_token)
        refresh_token = token_response.get('refresh_token')
        response = append_cookie(response, "refresh_token", refresh_token)
        return response

    except Exception as e:
        return HTMLResponse(content=f"<h1>Ошибка авторизации: {str(e)}</h1>", status_code=400)

@app.get("/logout")
async def logout(request: Request):
    """Выход из системы"""
    refresh_token = request.cookies.get("refresh_token")
    if refresh_token:
        try:
            keycloak_openid.logout(refresh_token)
        except Exception as e:
            print(f"Keycloak logout error: {e}")
    response = RedirectResponse(url="/")
    response.delete_cookie("access_token")
    response.delete_cookie("refresh_token")
    return response

if __name__ == '__main__':
    run(app="02_withroles:app", host='0.0.0.0', port=8100, workers=4, log_level='warning')
```

Revision #1
Created 20 May 2026 15:53:54 by Admin
Updated 20 May 2026 16:23:27 by Admin