Timebox3 – RAPI- auth

Oversigt

OpgaveNavn	Implementering af restAPI authorization		
Implementering af krav	Delvis implementation af WEB-13		
Udført af	Jan	**Dato**	07-10-2021
Timebox	3	**Område**	Website

Contents

INTRODUKTION **1**
ANALYSE **2**
DESIGN **2**
IMPLEMENTERING **2**
VERIFIKATION **3**
TESTRESULTAT **4**
KONKLUSION **5**
REFERENCER ..... FEJL! BOGMÆRKE ER IKKE DEFINERET.

Introduktion

For at kun autoriserede bruger af hjemmesiden kan bruger restAPI'en til at indsætte og hente oplysninger, skal der være et restAPI autoriserings system i hjemmesiden.

Analyse

Der er overordnet set 3 opgaver til denne deliverables hjemmeside funktionaliteter:

- Til autorisering af brug af restAPI'en skal der "logges ind" gennem hjemmesiden, hvilket vil returnere et token.

- Der skal bruges et token som erstatning for login kredentialer på de andre restAPI endpoints.

- Tokenet skal være muligt at slette efter brug.

For at implementere de 3 overstående funktionaliteter, kan der bruges python biblioteket "djoser", der er et plugin til python biblioteket "djangorestframework", som restAPI'en bliver baseret på.

Djoser giver nogle boiler endpoints til forholdsvis:

- Login
    - For at genere, gemme i databasen og udlevere et token.
- Logout
    - For at fjerne en token i databasen

Samtidigt giver djoser muligheden for at ændre restAPI certificering til djoser's token baserede certificering, gennem Django's settings.py fil, og gør derved at koden til restAPI funktionerne kan skrives uden ekstra kode til certificering, eller gøre brug af python dekoratorer.

Design

Der vil til certificeringen ikke være nogen frontend, udover hvad djoser stiller til rådighed, gennem deres boiler templates.

Backend mæssigt, vil følgende flow ske:

1. For at få et token udleveret, bliver bruger email og password, sendt til et djoser login restAPI endpoint, hvor hvis brugeren eksisterer, og korrekt password er givet, bliver der returneret et token, som kan bruges på alle andre restAPI endpoints.
2. Når der sendes en POST eller GET request til hjemmesidens restAPI endpoints for data håndtering fx for logs, vil requesten først gå igennem djoser's certificerings tjek, for derefter blive sendt videre til restAPI'en. Itilfælde af der ikke er et token i POST eller GET requesten, vil der blive givet en autoricerings fejl, og requesten vil derfor også fejle.
3. Når en bruger er færdig med at skulle bruge et token, kan brugerens pågældende token, blive slette ved sende den til djosers logout restAPI endpoint, og vil derefter blive slettet i databasen.

Implementering

For at kunne implementere brug af djoser skal følgende gøres, efter installering af django-restframework:

1. installere djoser:
    a. $ pip install djoser

2. Indsættelser i Djangos settings fil:
    a. Installed apps = [
    'rest_framework.authtoken',
    'djoser',
    …]
    b. REST_FRAMEWORK = {
     'DEFAULT_AUTHENTICATION_CLASSES': (
      'rest_framework.authentication.TokenAuthentication',
      'rest_framework.authentication.SessionAuthentication',
     ),
     'DEFAULT_PERMISSION_CLASSES': (
      'rest_framework.permissions.IsAuthenticated',
     ),
    }
3. indsættelser i website apps urls.py
    a. urlpatterns = [
     path('api/auth/', include('djoser.urls.authtoken'))
     …]

Efter de overstående punkter er sat ind i projektet, vil djosers tokens models kunne føres til databasen, med Django's migrate kommando.

Når der skal sendes en request til en af restAPI'ens endpoints, vil det kunne gøres med cURL, hvor cURL strukturen på windows er:

- login endpoint (rød tekst er steder der skal fyldes ud):
    - curl -X POST -d "{\"email\": \"email_adresse\",\"password\": \"psswrd\"}" -H "Content-Type:application/json" http://url/api/auth/token/login/

- POST data endpoint (rød tekst er steder der skal fyldes ud):
    - curl -X POST -d "{\"D1_navn\":\"D1_værdi\",\"D2_navn\":\"D2_værdi\"}" -H "Authorization: Token token" -H "Content-Type:application/json" http://url/rest_api/api_endpoint /

- logout endpoint (rød tekst er steder der skal fyldes ud):
    - curl -X POST http://url/api/auth/token/logout/ -H "Authorization: Token token"

Verifikation

For at teste funktionaliteterne bliver der testes 4 punkter:

- login endpoint
- POST endpoint
    - Med korrekt token

    - Med forkert token
- Logout endpoint
- test.py køres, for at tjekke intet andet i hjemmesiden er ødelagt.

Der er forventet at:

- office, field og admin test brugere er lavet.
- Der bliver brugt en windows maskine til at køre curl i CMD.
- Hjemmesiden er oppe og køre.

Tabel 1: Tests til verifikation af opgave

Test	Test Steps	Pre-requisites	Pass-betingelser	Resultat
Login	1. start cmd 2. send ”login” cURL streng fra Tabel 2	N/A	Et token bliver sendt tilbage	Bestået
Post KK	1. Start cmd 2. kopier token fra test ”login”. 3. kopier ”log og key” cURL streng fra Tabel 2, og skift token ud. 4. send streng	1. der er oprettet et facilitet med navnet ”test fac1” 2. der er lavet et jointable mellem ”test fac1” og ”field@field.com” brugeren.	Status success og facilitets nøglen fra test fac1 bliver retuneret	Bestået
Post FK	1. Start cmd 2. kopier ”log og key” cURL streng fra Tabel 2, og skift token ud, med noget random. 3. send streng	1. der er oprettet et facilitet med navnet ”test fac1” 2. der er lavet et jointable mellem ”test fac1” og ”field@field.com” brugeren.	Invalid token bliver retuneret	Bestået
Logout	1. start cmd 2. kopier token fra test ”login”. 3. kopier ”logout” cURL streng fra Tabel 2, og skift token ud. 4. send streng		Token bliver slettet fra databasen	Bestået
Test.py	1. start testen test.py i django projektet, med pytest		Alle test bliver bestået	Bestået

Tabel 2 cURL kommandoer

Login	curl -X POST -d "{\"email\": \"field@field.com\",\"password\": \"field\"}" -H "Content-Type:application/json" http://127.0.0.1:8000/api/auth/token/login/
Log og key	curl -X POST -d "{\"facility\":[{\"name\":\"test fac1\"}],\"log\":[{\"userName\":\"field\",\"companyName\":\"ewe\",\"dateTime\":\"2021-10-10T23:44:22\",\"userEmail\":\"field@field.com\",\"facilityName\":\"test fac1\",\"facilityLocation\":\"Herning\"}]}" -H "Authorization: Token 9245753f73108f6f53cec4082807bb7b35a61451" -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/log_api/
Facilitet joins	curl -X POST -d "{\"userEmail\":\"field@field.com\"}" -H "Authorization: Token 9245753f73108f6f53cec4082807bb7b35a61451" -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/key_api/
logout	curl -X POST http://127.0.0.1:8000/api/auth/token/logout/ -H "Authorization: Token 9245753f73108f6f53cec4082807bb7b35a61451"

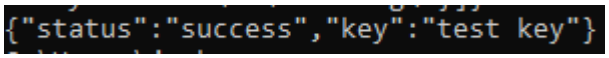
Testresultat

Resultatet fra test ”login” i Tabel 1 kan ses på Figur 1 login test.

{"auth_token":"9245753f73108f6f53cec4082807bb7b35a61451"}

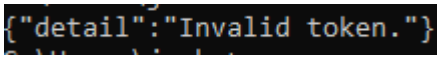
Figur 1 login test

Resultatet fra test ”POST kk” i Tabel 1 kan ses på Figur 2 POST kk test. Figur 1 login test



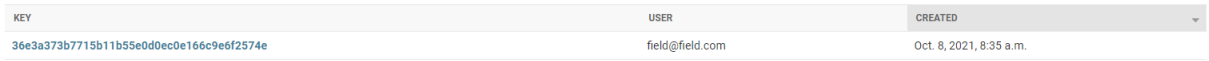
Figur 2 POST kk test

Resultatet fra test ”POST uk” i Tabel 1 kan ses på Figur 3 POST uk test.

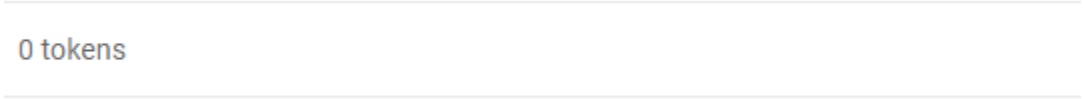


Figur 3 POST uk test

Resultatet fra test ”logout” i Tabel 1 kan ses på Figur 4 før logout test og Figur 5 efter logout test.

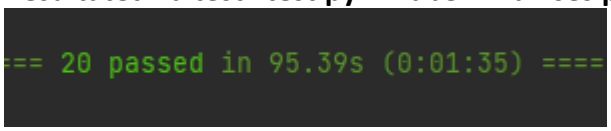


Figur 4 før logout test



Figur 5 efter logout test

Resultatet fra test ”test.py” i Tabel 1 kan ses på Figur 6 pytest.



Figur 6 pytest

Konklusion

Det hele virker som planlagt, og gør det derfor muligt at kunne hente og indsætte selektive former for informationer til og fra databasen.
Det hele kunne implementeres hurtigere end planlagt pga. biblioteket djoser, som laver boiler template endpoints til tokens.