Timebox 3-RestAPI: Get key, post logs og get facilitets

Oversigt

OpgaveNavn	Implementering af funktionaliteterne i Rest API		
Implementering af krav	Delvis implementering af krav WEB-13		
Udført af	Marc	**Dato**	08-10-2021
Timebox	3	**Område**	Website – Rest API

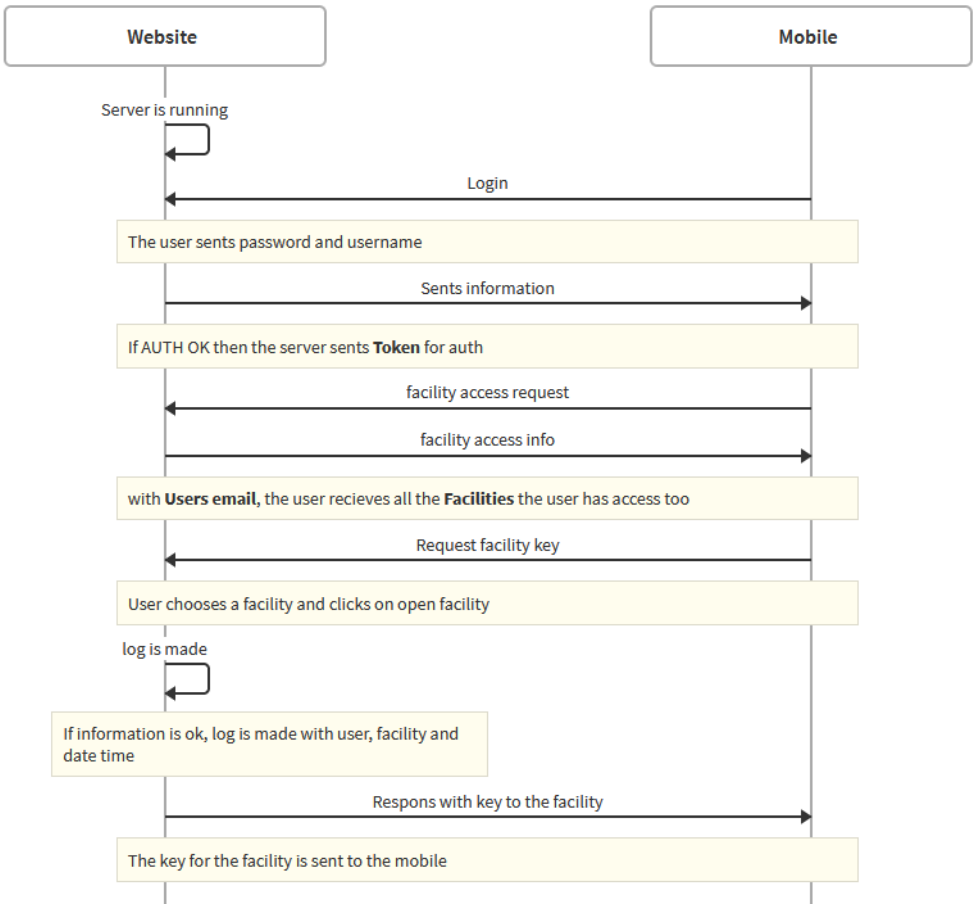
Contents

Introduktion

Det skal være muligt at tilgå, informationer fra databasen med en mobil. Derfor skal der sættes et REST API (*representational state transfer application programming interface*) op, for at håndtere endepunkterne. Dette dokument om handler opsætningen af REST API og dens funktionaliteter.

Analyse og Design

Som REST API anvend Django REST framework [1], som er nemt at interagere med Django, og derved spare udviklingstid, dertil har frameworket også nogle sikkerheds features som kan implementeres.
Først analyseres der, hvem sender hvad og hvilken rækkefølge, altså en protokol.
Protokollen for at hente en ”nøgle” tilhørende en facilitet, sende og skrive en log i databasen og hente information om facilitets adgang, er som på Figur 1.

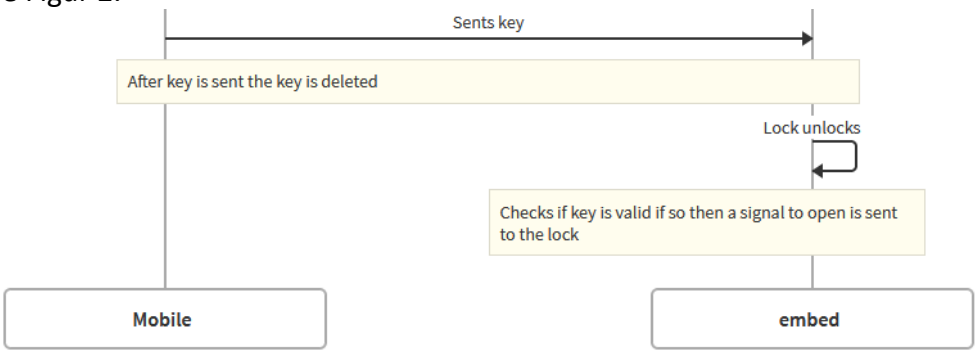


Figur 1 Sekvens for RestAPI del 1 – website og mobil

Dette dokument står for at udvikle håndteringen af ”facility access request”, ”request facility key”, ”log is made” og ”respons with key to facility” og de skal kunne følgende:

- ”facility access request”, skal via “user email” kunne sende alle brugerens facilitets adgange.
- ”log is made”, når en ”request facility key” logges tiden, “user” og “facility” før “nøglen” sendes
- ”request facility key”, skal via “facility name” og “user email” tjekke om bruger har adgang til den pågældende facilitet.
- ”respons with key to facility”, hvis alle tjeks er OK og der er lavet en log, så sendes “nøglen” til den ønskede facilitet.

Det er så mening at efter nøglen er blevet brugt til at åbne facilitetslåsen med, at nøgle slettes, se Figur 2.



Figur 2 Sekvens for RestAPI del 2 – mobil og embed

Der udvikles to klasser i appen ”rest_api”:

- LogsView som står for at håndtere at skrive en log i databasen og sende facilitetsnøglen til brugeren.
- FacilityView som står for at finde de faciliteter som brugeren har adgang til.

Implementering

I settings.py filen skal der tilføjes de apps, som skal med, se Figur 3.

```
38      'rest_api',
39      'rest_framework',
```

Figur 3 filen setting.py - installed apps

Derefter skal der under appen i urls.py tilføjes stigen til ”views”, se Figur 4.

```
6   urlpatterns = [
7       path('log_api/', LogsView.as_view()),
8       path('key_api/', FacilityView.as_view())
9   ]
```

Figur 4 Appens urls.py

Under Website i filen urls.py, skal der tilføjes stien til appen, som på Figur 5.

```
15      path('rest_api/', include('rest_api.urls')),
```

Figur 5 Website urls.py

I appen “rest_api” laves filen ”Serializer.py”, hvor alle ”Serializers” befinder sig.

På Figur 6, kan man se implementeringen at ”FacilityView”. Den indeholder en funktion post, der først bruger en ”Serializer” ”UserSerializer”, se Figur 7, som laver en JSON-streng om til Python dict, som der så kan anvendes. Brugerens Email hentes ud og bruges til at finde primary key’en, hvorefter alle brugens facilitetsadgange findes og sendes tilbage.

```
42  class FacilityView(APIView):
43      def post(self, request):
44          post_data = UsersSerializer(data=request.data)
45          if post_data.is_valid():
46              data = post_data.data['userEmail']
47              pk = Users.objects.get(email=data)
48              list_fac = JoinTable.objects.all().filter(user=pk).values_list("facility__name", "facility__location",
49                                                                             "user__email", "user__company",
50                                                                             "user__name")
51              return Response({"status": "success", "list":list_fac}, status=status.HTTP_200_OK)
52          else:
53              return Response({"status": "error", "data": post_data.errors}, status=status.HTTP_400_BAD_REQUEST)
```

Figur 6 Class facilityView

På Figur 7 ses ”UserSerializer” som bruger modellen ”Users”, hvor field ”userEmail” er det eneste element.

```
35  class UsersSerializer(serializers.HyperlinkedModelSerializer):
36      userEmail = serializers.EmailField()
37
38      class Meta:
39          model = Users
40          fields = ['userEmail']
```

Figur 7 UsersSerializer

Klassen LogsView, se Figur 8 og Figur 9, indeholder funktionen post, som bruger en ”Serializer” ”GetKeyPostLogSerializer”, se Figur 10, indholdet tjekkes om valid og information hentes ud.

```
class LogsView(APIView):
    def post(self, request):
        post_data = GetKeyPostLogSerializer(data=request.data)
        if post_data.is_valid():
            # Get message data
            log_userName = post_data.data['log'][0]['userName']
            log_companyName = post_data.data['log'][0]['companyName']
            log_dateTime = post_data.data['log'][0]['dateTime']
            log_userEmail = post_data.data['log'][0]['userEmail']
            log_facilityName = post_data.data['log'][0]['facilityName']
            log_facilityLocaltion = post_data.data['log'][0]['facilityName']
            data2 = post_data.data['facility'][0]['name']
```

Figur 8 Class LogsView del 1

Der laves et Logs objekt, hvor brugerne hentes ind i Logs sammen med den pågældende facilitet, alle bruger og facilitets informationer sættes i objektet med et tidsstempel, som derefter gemmes i databasen.

Nøglen til den pågældende facilitet, findes og sendes.

```
            # make log
            make_log = Logs()
            # get User object from unique Email
            make_log.user = Users.objects.get(email=log_userEmail)
            # get facility object from unique facility name
            make_log.facility = Facilities.objects.get(name=log_facilityName)
            make_log.companyName = log_companyName
            make_log.dateTime = log_dateTime
            make_log.facilityLocation = log_facilityLocaltion
            make_log.facilityName = log_facilityName
            make_log.userEmail = log_userEmail
            make_log.userName = log_userName
            make_log.save()
            # Get key
            facility = Facilities.objects.get(name=data2)
            key = facility.key
            return Response({"status": "success", "key": key}, status=status.HTTP_200_OK)
        else:
            return Response({"status": "error", "data": post_data.errors}, status=status.HTTP_400_BAD_REQUEST)
```

Figur 9 Class LogsView del 2

Klassen “GetKeyPostLogSerializer” er en ”mixed class” bestående af to klasser ”FacilitySerializer” og ”LogSerializer”, se Figur 10.

```python
class LogSerializer(serializers.HyperlinkedModelSerializer):
    dateTime = serializers.DateTimeField()
    companyName = serializers.CharField(max_length=120)
    userName = serializers.CharField(max_length=80)
    userEmail = serializers.EmailField()
    facilityName = serializers.CharField(max_length=80)
    facilityLocation = serializers.CharField(max_length=120)

    class Meta:
        model = Logs
        fields = ['companyName', 'userName', 'dateTime', 'userEmail',
                  'facilityName', 'facilityName', 'facilityLocation']


class FacilitySerializer(serializers.HyperlinkedModelSerializer):
    name = serializers.CharField(max_length=80)

    class Meta:
        model = Facilities
        fields = ['name']


class GetKeyPostLogSerializer(serializers.Serializer):
    facility = FacilitySerializer(many=True)
    log = LogSerializer(many=True)
```

Figur 10 GetKeyPostLogSerializer

Verifikation

For at kunne udføre de test som i Tabel 1, skal der være følende:

- Serveren kører
- Tre faciliteter er i databasen:
    1. test fac1
    2. test fac2
    3. test fac3
- To bruger:
    1. field@field.com med navn og kodeordet field
    2. office@office.com med navn og kodeordet office
- Følgende facilitets adgang:
    1. field har adgang til test fac1
    2. field har adgang til test fac2
    3. office har adgang til test fac3
- Loggen er tom
- At man har adgang til admin
- Man har logget ind via kommandoprompt og fået en token

Tabel 1: Tests til verifikation af opgave

Test	Test Steps	Pre-requisites	Pass-betingelser	Resultat
Get facilities	1. Send kommandoen: curl -X POST -d "{\"userEmail\":\"field@field.com\"}" -H "Authorization: Token indsæt-token-her" -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/key_api/ 2. Send kommandoen: curl -X POST -d "{\"userEmail\":\"office@office.com\"}" -H "Authorization: Token indsæt-token-her" -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/key_api/	1. Kommandoprompter åben	1. Man kan se de faciliter field og office har adgang til, se Figur 11 og Figur 12.	Bestået
Request Key and log	1. Send kommandoen: curl -X POST -d "{\"facility\":[{\"name\":\"test fac1\"}],\"log\":[{\"userName\":\"field\",\"companyName\":\"ewe\",\"dateTime\":\"2021-10-10T22:44:22\",\"userEmail\":\"field@field.com\",\"facilityName\":\"test fac1\",\"facilityLocation\":\"Herning\"}]}" -H "Authorization: Token indsæt-token-her " -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/log_api/	2. Kommandoprompter åben	1. Man får en nøgle tilbage og man kan på admin siden se logen, se Figur 13 og Figur 14.	Bestået

Testresultat

```
C:\Users\Bluns>curl -X POST -d "{\"userEmail\":\"field@field.com\"}" -H "Authorization: Token 0f45c8fad1fcec7d9022e15121f959d929ee97f6" -H "
Content-Type:application/json" http://127.0.0.1:8000/rest_api/key_api/
{"status":"success","list":[["test fac1","Herning","field@field.com","ewe","field"],["test fac2","Birk","field@field.com","ewe","field"]]}
```

Figur 11 "Get facilities" resultat for field brugeren

```
C:\Users\Bluns>curl -X POST -d "{\"userEmail\":\"office@office.com\"}" -H "Authorization: Token 0f45c8fad1fcec7d9022e15121f959d929ee97f6" -H
 "Content-Type:application/json" http://127.0.0.1:8000/rest_api/key_api/
{"status":"success","list":[["test fac3","Herning","office@office.com","ewe","office"]]}
```

Figur 12 "Get facilities" resultat for office brugeren

```
C:\Users\Bluns>curl -X POST -d "{\"facility\":[{\"name\":\"test fac1\"}],\"log\":[{\"userName\":\"field\",\"companyName\":\"ewe\",\"dateTime
\":\"2021-10-10T22:44:22\",\"userEmail\":\"field@field.com\",\"facilityName\":\"test fac1\",\"facilityLocation\":\"Herning\"}]}" -H "Authori
zation: Token 0f45c8fad1fcec7d9022e15121f959d929ee97f6" -H "Content-Type:application/json" http://127.0.0.1:8000/rest_api/log_api/
{"status":"success","key":"22222222"}
```

Figur 13 "Request Key and log" resultat for field brugeren med test fac 1

Figur 14 "Request Key and log" resultat for field brugeren med test fac 1 på admin siden

Konklusion

Det er nu muligt at hente data fra databasen ved hjælp af REST API.

Referencer

[1]	Django REST framework https://www.django-rest-framework.org/ (sidst besøgt 08-10-2021)