HARJOITUSTYÖN 4. VAIHE: VISUALISOINTI

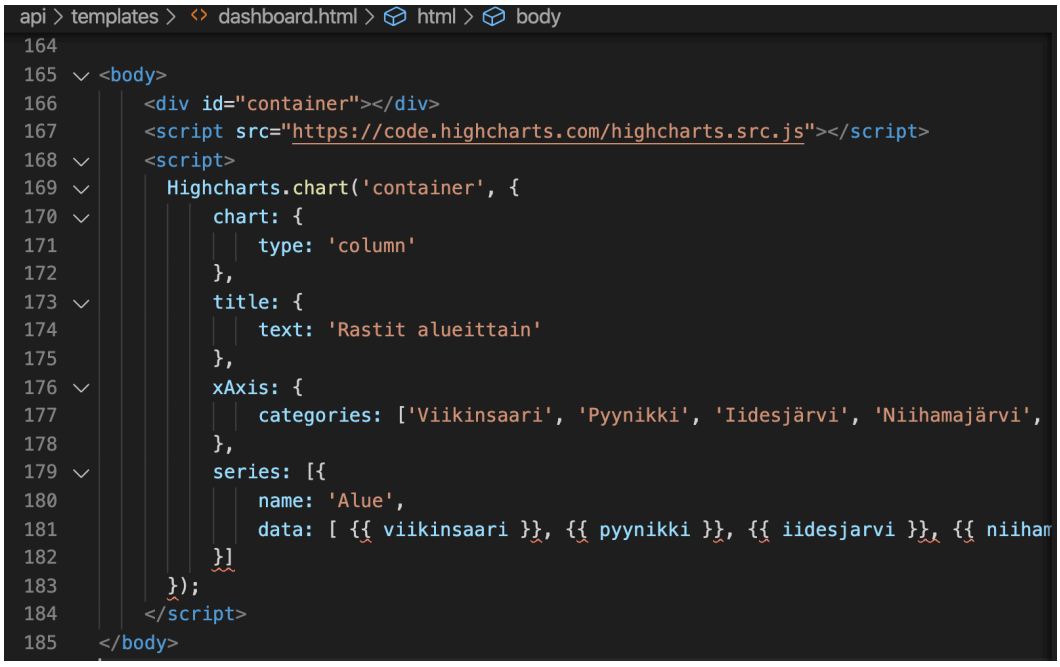
YLEISTÄ

Harjoitustyön neljännessä vaiheessa mallinnetaan ja visualisoidaan dataa kojelaudalle *Highchartsin* avulla. Edelleen käytössä oleva data on harjoitustyön kolmannessa vaiheessa käyttöön otettua dataa *data.tampere.fi* -sivuilta. Tässä vaiheessa on siis esitetty dataa Tampereen luontopolkurasteista pylväsdiagrammin ja puoli ympyrän avulla. Pylväsdiagrammissa on esitetty rastien määrät alueittain ja puoli ympyrässä puolestaan alueiden sisältämien rastien prosentuaalinen osuus suhteessa kaikkien rastien määrään.

KOJELAUDAN LUOMINEN

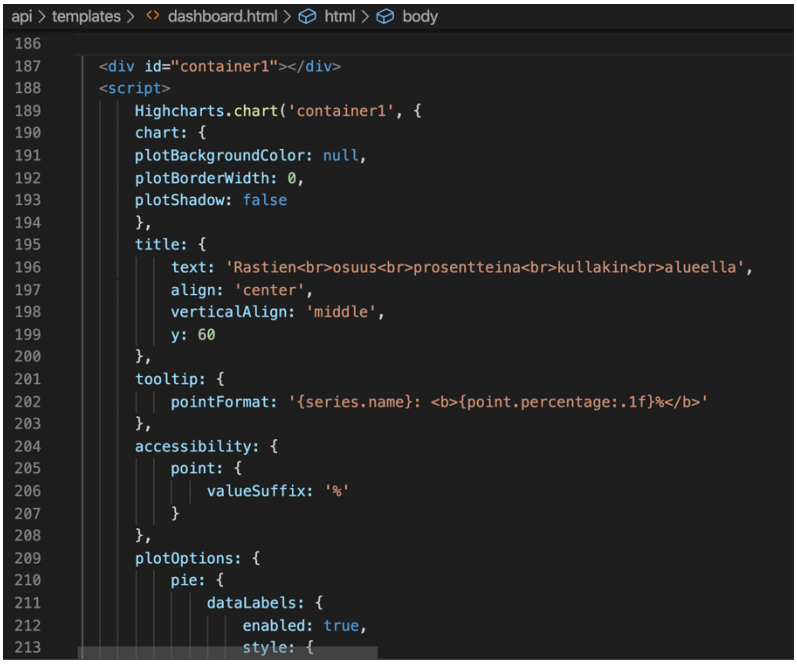
Ensin lähdin luomaan dashboardia eli kojelautaa alustaksi kuvaajille. Löysin mieleiseni ohjeen kojelaudan luomiselle *github.com* -nimiselät sivustolta. Sivustolta löytyy tarkat ja selkeät ohjeet kojelaudan luomiseen.

Aluksi loin *dashboard.html* -nimisen tiedoston *github*in ohjeiden mukaan. Tämän jälkeen lähdin toteuttamaan kuvaajia *simpleisbetterthancomplex.com* -nimisen sivun ohjeiden mukaisesti kojelautaan. Käytin sivuston Titanic -esimerkin koodipohjaa ja sovelsin esimerkin pohjalle omaa dataani luontopolkurasteista. Tarkoituksena oli aluksi luoda pylväsdiagrammi. Pylväsdiagrammin visualisointi on nähtävissä kuvassa 1.

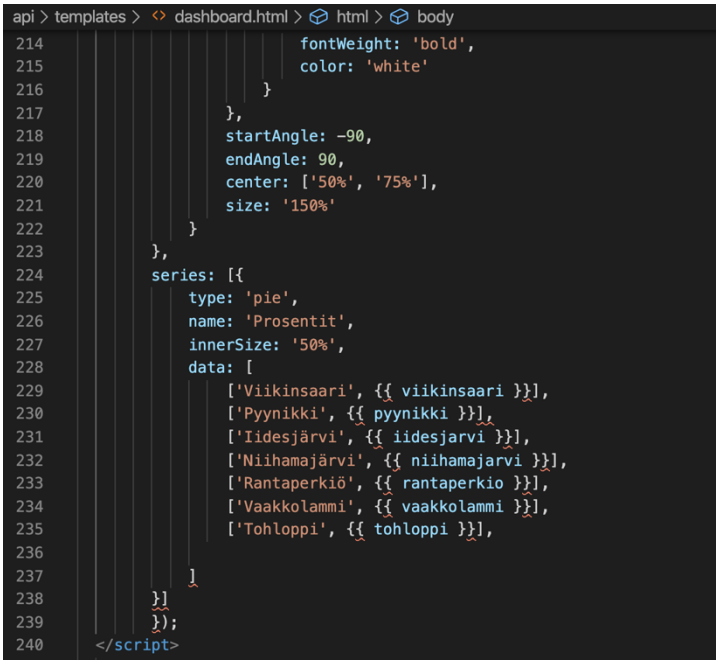


Kuva 1. *dashboard.html* -tiedoston osa, jossa pylväsdiagrammin visualisointi.

Tämän jälkeen loin *jsfiddle.net* -nimisen sivuston ohjeiden avulla toisen, puoliympyräkuvaajan pylväsdiagrammin alapuolelle. Kuvissa 2 ja 3 on esitetty puoliympyrän visualisointi.



Kuva 2. *dashboard.html* -tiedoston osa, jossa puoliympyrän visualisointi osa 1/2.



Kuva 3. *dashboard.html* -tiedoston osa, jossa puoliympyrän visualisointi osa 2/2.

Jotta visualisoinnit saatiin konkreettisesti näkyviin kojelaudalle, täytyi *views.py* ja *urls.py* -tiedostoihin rakentaa ehdot datan mallintamiselle. Kuvissa 4, 5, 6, 7 ja 8 on esitetty nämä tiedostot ja niihin rakennetut ehdot.

```
api > views.py > dashboard
from django.shortcuts import render
import requests



url = 'https://data.tampere.fi/data/api/action/datastore_search?resource_id=16b6e9d3

def show_data(request):
        response = requests.get(url)
        geodata = response.json()
        geodata = geodata['result']
        geodata = geodata['records']
        cleanData = []

        viikinsaari = 0
        pyynikki = 0
        iidesjarvi = 0
        niihamajarvi = 0
        rantaperkio = 0
        vaakkolammi = 0
        tohloppi = 0

```

Kuva 4. *views.py* -tiedoston osa 1/4.

```
api > views.py > dashboard

        for field in geodata:
                cleanData.append(field)
                if field['TUNNUS'] == 1:
                        viikinsaari += 1
                if field['TUNNUS'] == 2:
                        pyynikki += 1
                if field['TUNNUS'] == 3:
                        iidesjarvi += 1
                if field['TUNNUS'] == 4:
                        niihamajarvi += 1
                if field['TUNNUS'] == 5:
                        rantaperkio += 1
                if field['TUNNUS'] == 6:
                        vaakkolammi += 1
                if field['TUNNUS'] == 7:
                        tohloppi += 1


        return render(request, 'api.html', {
                        'geodata': cleanData, 'viikinsaari': viikinsaari, 'pyynikki'
                        'rantaperkio': rantaperkio, 'vaakkolammi': vaakkolammi, 'toh
        })

```

Kuva 5. *views.py* -tiedoston osa 2/4.

api > views.py > dashboard

```
46  def dashboard(request):
47          response = requests.get(url)
48          geodata = response.json()
49          geodata = geodata['result']
50          geodata = geodata['records']
51          cleanData = []
52
53          viikinsaari = 0
54          pyynikki = 0
55          iidesjarvi = 0
56          niihamajarvi = 0
57          rantaperkio = 0
58          vaakkolammi = 0
59          tohloppi = 0
60
```

Kuva 6. *views.py* -tiedoston osa 3/4.

api > views.py > dashboard

```
60
61          for field in geodata:
62                  cleanData.append(field)
63                  if field['TUNNUS'] == 1:
64                          viikinsaari += 1
65                  if field['TUNNUS'] == 2:
66                          pyynikki += 1
67                  if field['TUNNUS'] == 3:
68                          iidesjarvi += 1
69                  if field['TUNNUS'] == 4:
70                          niihamajarvi += 1
71                  if field['TUNNUS'] == 5:
72                          rantaperkio += 1
73                  if field['TUNNUS'] == 6:
74                          vaakkolammi += 1
75                  if field['TUNNUS'] == 7:
76                          tohloppi += 1
77
78
79          return render(request, 'dashboard.html', {
80                          'geodata': cleanData, 'viikinsaari': viikinsaari, 'pyynikki'
81                          'rantaperkio': rantaperkio, 'vaakkolammi': vaakkolammi, 'toh
82          })
83
84
```

Kuva 7. *views.py* -tiedoston osa 4/4.

```
api > urls.py > ...
1   from django.urls import path
2   from . import views
3
4   urlpatterns = [
5       path('', views.show_data, name='show_data'),
6       path('dashboard', views.dashboard, name='dashboard')
7   ]
```

Kuva 8. *urls.py* -tiedosto.

Lopuksi muokkasin kojelaudan pohjaa ja lisäsin kolme *button* -toimintoa kojelaudan yläosaan. *Kojelauta* -tekstiä napauttamalla pääsee takaisin kotisivulle, *Muistiinpanot* -tekstiä napauttamalla pääsee muistiinpanojen luontisivulle ja *Luontorastit* -tekstiä napauttamalla pääsee tarkastelemaan listattua dataa luontopolkurasteista.

Lopputuloksena syntyi kuvan 9 mukainen siisti kojelauta.

Kuva 9. Kojelaudan selainnäkymä.

Kun hiiren vie vaikkapa yhden tolpan tai yhden väritetyn ympyrän osan päälle, näkee tarkemmat tiedot kyseisestä kuvaajan osasta.

HELPPOA

- Dashboardin luominen oli suhteellisen helppoa, sillä käytin apuna käytännössä valmista pohjaa josta vain poistin turhat *button* -toiminnot ja lisäsin vain sellaiset itse, jotka katsoin tarpeelliseksi. Lisäksi tein pohjaan pieniä komponenttien sijaintimuutoksia, jotta sain kojelaudasta haluamani näköisen.
- Ensimmäinen datan visualisointi tuotti paljon hankaluuksia ja vei aikaa, mutta toisen visualisoinnin rakentaminen sujui helpommin kun ymmärsi, miten datan saa näkyviin.

VAIKEAA/HAASTAVAA

- Kaikista haastavinta oli saada oma data visualisoinnin malleihin eli *views.py* -tiedoston koodin rakentaminen niin, että data saataisiin esitettyä toivotulla tavalla. Koska omat koodaustaitoni rajoittuvat ohjelmointi ykkösen kurssiin ja siitä kun on vielä aikaakin tovi, *views.py* -tiedoston koodi on hyvin kankeaa ja uskon, että toteutukselle olisi ollut helpompikin tapa.
- Jälleen tiedon ja avun löytäminen kojelaudan toteutukselle oli haastavaa. Koodiklinikoista ei juuri ollut apua, joten täytyi turvautua googlailuun ja vanhojen toteutusten selailuun, jotta sai edes ideasta kiinni.

LÄHTEET

https://github.com/Stack-Legacy/Dashboard-using-Highcharts-and-Django/blob/master/templates/index.html

https://simpleisbetterthancomplex.com/tutorial/2018/04/03/how-to-integrate-highcharts-js-with-django.html

https://jsfiddle.net/gh/get/library/pure/highcharts/highcharts/tree/master/samples/highcharts/demo/pie-semi-circle/