

Janmiikka Osara, janmiikka.osara@tuni.fi

HARJOITUSTYÖ, VAIHE 1

TLO-32400 Ohjelmallinen sisällönhallinta 2020

Tausta ja ympäristön valinta

Kyseisen harjoitustyön ensimmäisessä dokumentissa käsitellään harjoitustyön toteutusympäristöä, eli vastataan kysymyksiin, millainen projekti on kyseessä ja millä työkaluilla ja teknologioilla se suoritetaan. Harjoitustyön lopputuloksena on tarkoitus syntyä koodiportfolio, joka kokoaa yhteen harjoitustyöhön toteutetut toiminnot ja niiden tiiviin dokumentoinnin. Projektin tavoitteena on toteuttaa web-alustainen, avointa dataa hyödyntävä palvelu.

Itse olen vasta aloitteleva koodaaja, jolla on kokemusta vain Ohjelmointi 1 – Johdanto, kurssin opeista. Alkeellisen taustan vuoksi käytän harjoitustyössä Pythonin ohjelmointikielen ja Django-kehiksen yhdistelmää, mikä on suositeltu menetelmä aloittavalle web-ohjelmoijalle. Tulen käyttämään muun muassa seuraavia teknologioita:

- Python 3.8.0 (ohjelmointikieli)
- Django 2... (kehitysympäristö)
- Visual Studio Code (koodieditori)
- Virtualenv (virtuaaliympäristö)
- SQLite (tietokanta)
- Windows 10 (käyttöjärjestelmä)
- Git 2.25. (versionhallinta)

Rakennan kyseisen kehitysympäristön omalle kannettavalleni tietokoneelle (Lenovo Yoga 530) sopivaksi.

Ympäristön asentaminen ja vaiheen 1 toteutus

Suoritin Django-ympäristön asennuksen käytännössä kurssin luentokalvojen ja Kujalan koodiklinikan ohjeistuksen pohjalta. Python 3.8.0. ohjelmointikieli oli valmiiksi asennettuna koneelle, mutta varmistin kuitenkin vielä, oliko Add Python Path voimassa. Visual Studio Code oli myös asennettuna koneelle valmiiksi aiemmin kurssilla käydyin HTML+CSS koodiklinikan harjoituksen kautta. Olin aiemmin käyttänyt Notepad++:saa, mutta käännyin Kujalan suosittelemaan näppärämpään Visual Studio Code vaihtoehtoon.

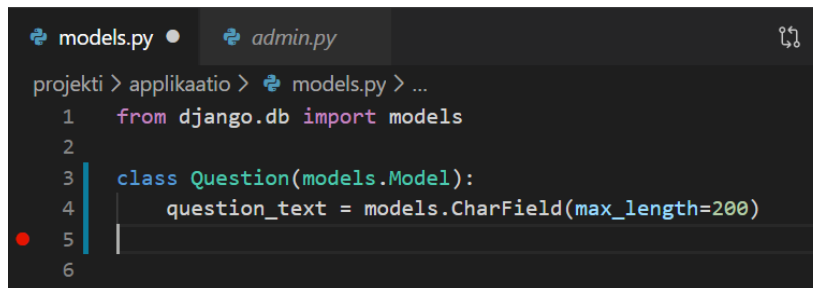
Harjoitustyöympäristön toteuttaminen alkoi varsinaisesti jo Windows laitteilla valmiiksi asennetusta Windows PowerShell:istä. Aluksi varmistimme, että PowerShell oli avattu järjestelmävalvojana, minkä jälkeen sallimme Scriptien suoritukset seuraavalla komennolla **Set ExecutionPolicy RemoteSigned**, mihin annettiin vastaus komennolla **Y** (yes). Tämän jälkeen tarkistettiin Pythonin version olemassaolo **Python -V** komennolla sekä pip-järjestelmän toimivuus komennolla **pip freeze**, mikä tarkistaa Pythonin sisäänrakennetun pip-järjestelmän toimivuuden. Virtualenv ympäristö varsinaisesti asennettiin komennolla **pip install virtualenv**, mikä latasi PowerShellin näkymään tiedot asennuksen etenemisestä. Kun virtuaaliympäristö asennettiin, valittiin kansio, mihin harjoitustyö tulisi tallentaa. Kansioiden hakeminen ja tulostaminen onnistui komennolla **ls**, joka tulostikin turhankin suuren listan koneen kansioista. Komento **cd..**, vie taas taaksepäin kansioista kansioon. Tällöin löytyi haluamani tallennettava kansio Käyttäjät, mihin päästiin komennolla **cd Käyttäjät**, sekä niin edelleen **cd Janmiikka → cd OHSIHA → cd harjoitustyö**, jolloin saavuttiin haluttuun kansioon. Tallennuspaikkani oli siten c:\Käyttäjät\Janmiikka\OHSIHA\harjoitustyö. Kansion luominen harjoitustyötä varten onnistui **mkdir harjoitustyö** komennolla. Virtuaaliympäristö oli nyt asennettuna PowerShelliin, mutta loimme ja avasimme sen virallisesti komennoilla **virtualenv env** ja **cd env**. minkä jälkeen virtuaaliympäristön avaaminen onnistui komennolla **.\Scripts\activate**, jonka edessä tulisi nyt olla (env) liite, jolloin ollaan virtuaaliympäristön sisällä.

Virtuaaliympäristön luomisen jälkeen keskityttiin Django asentamiseen. Django asennettiin komennolla **pip install django**, minkä jälkeen asennettiin pylint vastaavasti komennolla **pip install pylint**, joka oli tarkoitettu vain VS Coden käyttäjille, mukaan luettuna itselleni. Palasimme edelliseen kansioon aiemmin tutustutulla komennolla **cd..**, ja luotiin Django-projekti komennolla **django-admin startproject projekti**. Näiden vaiheiden jälkeen avasimme Visual Studio Coden, missä tarkistettiin aluksi Python lisäosan toimivuus. Tämän jälkeen suljimme VS Coden ja uudelleenkäynnistettiin projekti **code.** -komennolla Windows PowerShellissä. VS Codessa varmistettiin myös oikean tulkin valinta, mikä suoritettiin View-valikosta Command Palette Python: Select Interpreter valinnalla.

Kun kaikki edellä mainitut toiminnot ja komennot olivat suoritettuna, palasimme takaisin Power Shelliin ja tarkistimme, että kaikki toimivat niin kuin piti. Siirryimme siis projektikansioon tutulla **cd** komennolla, jonka jälkeen käynnistettiin Django serveri **py manage.py runserver**-komennolla.

Virtuaaliympäristön pystytyksen jälkeen lähdettiin itsessään tutustumaan Django-kehikseen. Aktivoituneeseen virtuaaliympäristöön asennettiin Django projektin pohjaksi, jotka suoritettiin komennolla **`pip install Django`**, minkä jälkeen toteutettiin projektin perustaminen komennolla **`django-admin.exe startproject projekti`**. Kun projektiympäristö kansioineen oli luotu, voi virallisesti alkaa tutkia projektia.

Tietokannaksi valitsin SQLiten, jolle löytyi settings.py kansioista jo valmiiksi DQlille tehdyt tekstit (database). Tietokannan asentamisessa ja käyttöönotossa seurasin seuraavan lähdemateriaaliosastossa esiintyneen verkkosivun ohjeistusta. Loin yhden testifunktion Guestion() models.py kansioon, jota ajoin WindowsPowerShellin kautta komennolla **`py.mange migrate --> py manage.py makemigrations applikaatio --> py manage.py sqlmigrate 0001`**. Näin ollen sain tallennettua ajetut muutokset PowerShellin kautta Django ympäristöön. Ohessa VisualStudioCodesta koodin pätkä komennoista.



```

models.py • admin.py
projekti > applikaatio > models.py > ...
1  from django.db import models
2
3  class Question(models.Model):
4      question_text = models.CharField(max_length=200)
5
6

```

Tein myös samaa verkkosivua hyödyntäen Django käyttäjän Shellissä seuraavssa kuvassa esitetyillä komennoilla.

Helpottaakseen omaa työskentelyä, asensin Git:n koneelleni, jotta versionhallinta onnistuisi selkeämmin ja mukavammin. Asetin kansion OHSIHA (minkä alla on harjoitustyö) Git kansion alle, jotta editorin kanssa versionhallinta olisi kätevämpää. Tein myös uuden käyttäjän GitHubiin, mihin liitin oman reporitorin kyseistä harjoitustyötä varten.

Helpot ja vaikeat asiat

Helpot asiat:

- Django-ympäristön pystyttäminen koodiiklinikan ohjeistuksen avun saattelemana

- Teknologien valinta (aloittavalle ohjelmoijalle suositellut teknologiat)

Vaikeat asiat:

- Windows PowerShellin ja VisualStudioCoden käyttäminen ja ymmärtäminen
- Erilaisten rajapintojen liittäminen toisiinsa ja kokonais kuvan ymmärtäminen

Lähdemateriaalit

Git-asennus ja käyttöönotto, linkin avulla: <https://medium.com/@djstein/modern-django-part-0-introduction-and-initial-setup-657df48f08f8>

Tietokannan SQLite asentaminen, sekä Django käyttäjätunnuksen lisääminen: <https://docs.djangoproject.com/en/3.0/intro/tutorial02/>

Lisäksi otin muutamia tärppejä aiemmista harjoitustyötoteutuksista Slack-alustalta.

Janmiikka Osara, janmiikka.osara@tuni.fi

HARJOITUSTYÖ, VAIHE 2

TLO-32400 Ohjelmallinen sisällönhallinta 2020

Vaiheen 2 toimenpiteet

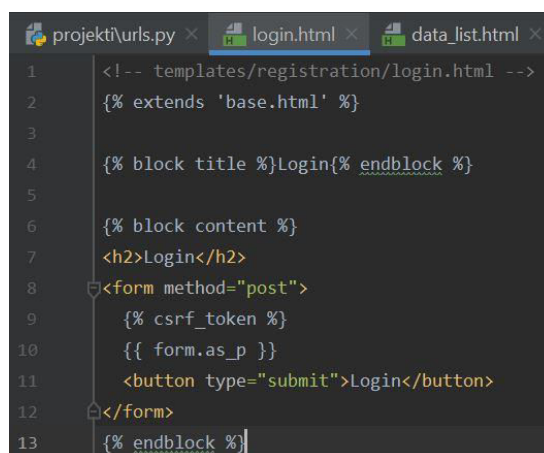
Harjoitustyön toisessa vaiheessa oli tarkoitus toteuttaa yksinkertainen sisään- ja uloskirjautuminen sekä oman käyttäjätunnuksen rekisteröityminen esimerkkipalvelun sisään. Tämän lisäksi sivulle toteutettiin alkeellinen CRUD (lisää-näytä-päivitä-poista) -palvelu.

Sisään- ja uloskirjautuminen

Harjoitustyön ensimmäisen vaiheen päätarkoituksena oli saada harjoitustyö teknologioineen ja ympäristöineen pystyyn, kun taas toisessa vaiheessa pääsi enemmän toteuttamaan itse. Sivun rakentaminen lähti käyttäjän sisään- ja uloskirjautumisesta, johon otin pitkälti mallia LearnDjango.com verkkosivustolta (linkki lähteissä) löytyneestä ohjeistuksesta. Django:n projektitaso jo valmiiksi luotua auth-appia hyödynnettiin, jonka urls.py -tiedostoon tehtiin seuraavat muokkaukset:

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('accounts/', include('django.contrib.auth.urls')),  
    path('', include('applikaatio.urls'))  
]
```

Uutta sisään- ja uloskirjautumista varten luotiin applikaation alle uusi kansio *templates*, ja edelleen uusi kansio *registration*. Registration kansioon luotiin uusi sisäänkirjautumista varten yksinkertainen login.html -tiedosto. Login -tiedosto näyttää seuraavalta:



```
1 <!-- templates/registration/login.html -->  
2 {% extends 'base.html' %}  
3  
4 {% block title %}Login{% endblock %}  
5  
6 {% block content %}  
7 <h2>Login</h2>  
8 <form method="post">  
9     {% csrf_token %}  
10     {{ form.as_p }}  
11     <button type="submit">Login</button>  
12 </form>  
13 {% endblock %}
```

Käyttäjän ohjaamisen takaamiseksi sisään- ja uloskirjautumisessa, lisäsin seuraavat komennot:

`LOGIN_REDIRECT_URL = '/'` ja `LOGOUT_REDIRECT_URL = 'home'` projektitason `settings.py` -tiedostoon, jotta reitti säilyisi. Kotisivua varten *templatesin* alle tuli vielä luoda *base.html* sekä *home.html* -tiedostot, jotta voimme välittää käyttäjän kommunikoinnin sisään- ja uloskirjautumisen tukena. Ohessa edellisiin viitatus html –tiedostot:

The screenshot shows two Django template files side-by-side in a code editor. The left pane shows `home.html` which extends `base.html` and contains logic to display a login link for non-authenticated users and a logout link for authenticated users. The right pane shows `base.html` which defines the overall HTML structure, including the `<doctype html>`, `<html>`, `<head>` with charset and title, and a `<main>` body containing the block content.

Kotisivun näyttämiseksi tuli vielä lisätä komento `path("", TemplateView.as_view(template_name='home.html'))`, jotta reitti saatiin toteutettua. Nyt PowerShellistä Django-projektia ajamalla saadaan sivustoomme localhost:8000 seuraavanlainen näkymä:

Login

Username:

Password:

Hi testi!

[logout](#)

You are not logged in

[login](#)

Näin ollen voimme todeta sisään- ja uloskirjautumisen olevan toiminnassa.

Rekisteröiminen

Rekisteröintiä varten noudatin samaisen verkkosivun ohjeistusta, linkki löytyy lähdemateriaaleissa. Rekisteröimistä varten PowerShellistä luotiin uusi app *accounts*, jonka `urls.py` - ja `views.py` -tiedostoihin tehtiin seuraavat muutokset:

The screenshot shows a snippet of a Django `urls.py` file. It imports `path` from `django.urls` and `views` from the current module. It then defines a `urlpatterns` list containing a path for `signup/` that points to the `SignUp.as_view()` method with the name `'signup'`.

Rekisteröitymissivua varten luotiin uusi `signup.html` -tiedosto `templates` -kansion alle, johon ajetaan näkymä Djangoa varten PowerShellin kautta syöttämällä `localhost:8000/accounts/signup` verkkoselaimen osoiterivistöön. Saadaan seuraavanlaiset näkymät:

Sign up

Username: Required. 150 characters or fewer. Letters, digits and `@/./+/-/_` only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Onnistuneen rekisteröimisen ja sisäänkirjautumisen jälkeen käyttäjä saa vastaavanlaisen tervehdyksen kuin edellisessä kohdassa, sekä näkymästä voi myös kirjata ulos itsensä.

CRUD -toiminnot

Lisää-näytä-päivitä-poista -toimintoa, eli CRUD-toimintoa varten seurasin Rayed's Real Life -sivuston esittämää ohjeistusta, jonka linkki esitetään lähdemateriaaleissa. CRUD -toimintoja varten luotiin uusi app data, jonka tarkoitus on toimia kuten CRUD -periaatteiden mukaan. Käyttäjä voi siis itse lisätä, muokata tai poistaa tietoa palveluun. CRUD -palvelua varten datan alle luotiin migrations -kansio, johon tehtiin seuraavat muutokset `urls.py` -, `views.py` -, `models.py` – ja `admin.py` -tiedostoihin:

```

views.py x  urls.py x  models.py x  admin.py x
1  from django.contrib import admin
2  from data.models import Data
3
4  admin.site.register(Data)
5

```

```

views.py x urls.py x
1 from django.urls import path
2
3 from . import views
4
5 urlpatterns = [
6     path('', views.DataList.as_view(), name='data_list'),
7     path('view/<int:pk>', views.DataView.as_view(), name='data_view'),
8     path('new', views.DataCreate.as_view(), name='data_new'),
9     path('view/<int:pk>', views.DataView.as_view(), name='data_view'),
10    path('edit/<int:pk>', views.DataUpdate.as_view(), name='data_edit'),
11    path('delete/<int:pk>', views.DataDelete.as_view(), name='data_delete'),
12 ]

```

```

views.py x urls.py x models.py x
1 from django.db import models
2 from django.urls import reverse
3
4 class Data(models.Model):
5     name = models.CharField(max_length=200)
6     pages = models.IntegerField()
7
8     def __str__(self):
9         return self.name
10
11     def get_absolute_url(self):
12         return reverse('data_edit', kwargs={'pk': self.pk})
13

```

```

views.py x urls.py x
1 from django.http import HttpResponse
2 from django.views.generic import ListView, DetailView
3 from django.views.generic.edit import CreateView, UpdateView, DeleteView
4 from django.urls import reverse_lazy
5 from data.models import Data
6
7 class DataList(ListView):
8     model = Data
9
10 class DataView(DetailView):
11     model = Data
12
13 class DataCreate(CreateView):
14     model = Data
15     fields = ['name', 'pages']
16     success_url = reverse_lazy('data_list')
17
18 class DataUpdate(UpdateView):
19     model = Data
20     fields = ['name', 'pages']
21     success_url = reverse_lazy('data_list')
22
23 class DataDelete(DeleteView):
24     model = Data
25     success_url = reverse_lazy('data_list')

```

Tämän jälkeen migrations -kansion alle luotiin edelleen uusi data -kansio, johon lisäsimme jokaiselle CRUD -vaiheelle erilaiset näkymät html -sivustojen valossa, joiden koodinpätkiä esitän seuraavissa kuvissa:

```

data_confirm_delete.html x data_detail.html x
<h1>Data Details</h1>
<h2>Name: {{object.name}}</h2>
Pages: {{ object.pages }}

<h1>Data Delete</h1>
<form method="post">{% csrf_token %}
    Are you sure you want to delete "{{ object }}" ?
    <input type="submit" value="Submit" />
</form>

1 <h1>Data</h1>
2
3 <table border="1">
4 <thead>
5 <tr>
6 <th>Name</th>
7 <th>Pages</th>
8 <th>View</th>
9 <th>Edit</th>
10 <th>Delete</th>
11 </tr>
12 <head>
13 <tbody>
14 {% for data in object_list %}
15 <tr>
16 <td>{{ data.name }}</td>
17 <td>{{ data.pages }}</td>
18 <td><a href="{% url 'data_view' data.id %}">view</a></td>
19 <td><a href="{% url 'data_edit' data.id %}">edit</a></td>
20 <td><a href="{% url 'data_delete' data.id %}">delete</a></td>
21 </tr>
22 {% endfor %}
23 </tbody>
24 </table>
25
26 <a href="{% url 'data_new' %}">New</a>

<h1>Data Edit</h1>
<form method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit" />
</form>

```

Ajamalla Django -projekti PowerShellin kautta saatiin seuraavanlainen näkymä verkkopalvelimelle:

Data

Name	Pages	View	Edit	Delete
New				

Palvelimelle pystyi täten lisäämään, poistamaan sekä muokkaamaan tietoa, joten CRUD -palvelun voidaan sanoa onnistuneen.

Helpot ja vaikeat asiat

Helpot asiat:

- Sisään- ja uloskirjautumisen, rekisteröitymisen ja CRUD -toiminnon nopea ja helppo toteuttaminen lähdemateriaalin avustamana
- Kattava avun saaminen joko Slack -alustalta tai verkosta

Vaikeat asiat:

- Tiettyjen koodinpätkien muokkaaminen ja virheiden löytäminen omalta koodieditorilta
- Kokonaiskuvan puuttuva hahmotus, kuinka rajapinnat todella ovat kytköksissä

Lähdemateriaalit

LearnDjango – Tutoriaali rekisteröitymisestä: <https://learndjango.com/tutorials/django-signup-tutorial>

LearnDjango – Tutoriaali sisään- ja uloskirjautumisesta: <https://learndjango.com/tutorials/django-login-and-logout-tutorial>

Rayed – Tutoriaali CRUD -toiminnon toteuttamisesta: <https://learndjango.com/tutorials/django-login-and-logout-tutorial>

Lisäksi otin muutamia tärppejä aiemmista harjoitustyötoteutuksista Slack-alustalta.

Janmiikka Osara, janmiikka.osara@tuni.fi

HARJOITUSTYÖ, VAIHE 3

TLO-32400 Ohjelmallinen sisällönhallinta 2020

Vaiheen 3 toimenpiteet

Vielä harjoitustyön edellisessä vaiheessa olin hieman epävarma oman ideani suhteen, mutta päädyin lopulta oman mielenkiinnon kohteeni – jalkapallon saattelmana lähteä rakentamaan kevyttä sovellusta, joka näyttäisi tietyn päivän jalkapallo-otteluita eri puolilla maailmaa, jalkapallodataa hyödyntämällä. Kuitenkin alkaessa työstämään vaihetta, korona perui kaikki jalkapallo-ottelut, pilaten samalla myös omat suunnitelmani. Näin ollen päädyin toteuttamaan yksinkertaisen Django-Python sääsovelluksen, joka näyttää käyttäjän tiedusteleman paikkakunnan säätilan verkkosivun näkymään. Hyvä puoli tässä on myöskin se, että korona ei tulisi ainakaan kyseistä ideaa pilaamaan. API-rajapintojen hyödyntäminen on muutenkin täysin uusi tuttavuus omalla kohdalla, joten kyseiseen vaihtoehtoon monipuolinen tukimateriaalia esimerkiksi Youtubesta ja Stack Overflow:sta. OpenWeatherMap -sivusto tarjosi ilmaiseksi Current weather data API:n, josta löytyi hyvin monipuolisesti erilaista tietoa monen eri paikkakunnan säätilasta.

API-rajapinnan käyttö / Sääsovelluksen toteutus

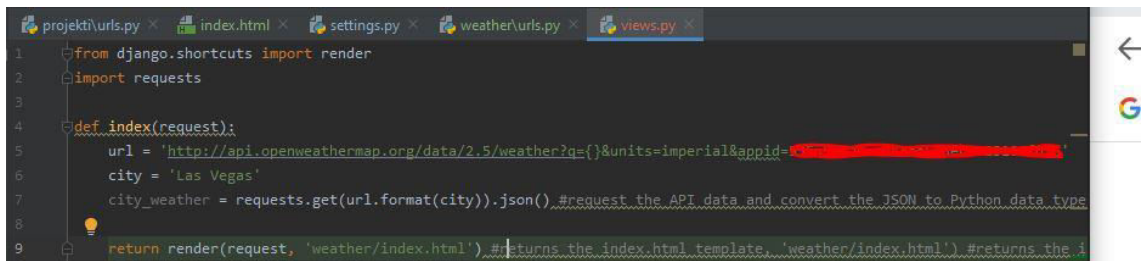
Ensimmäisenä toimenpiteenä asensin PowerShellin kautta *requests*-pyynnöt komennolla *pipenv install requests*, jotta voimme käsitellä API-dataa Pythonin kautta myöhemmin editorissa. Sääsovelluksen toteuttaminen lähti taas kuten aikaisemminkin, uuden sääsovelluksen *weather* luomisesta projekti- kansion alle. Näin ollen lisäsin projektitason *settings.py*- ja *urls.py* tiedostoihin tarvittavat muutokset (joita on aikaisemminkin vaiheissa tehty), jotta Django tietää, että haluamme käyttää kyseistä sääsovellusta omaan projektiimme. Sääsovellusta varten oli luotava tunnukset OpenWeatherMap:iin ja hankittava sieltä oma APIKEY datan keräämistä varten. Testasin oman APIKEY:n toimivuuden selaimella laittamalla osoiterivistölle OpenWeatherin API-linkin appid:n tilalle oman APIKEY:ni ja paikkakunnaksi kotoseutuni Hämeenkyrön. Selain antoi seuraavat tiedot Hämeenkyrön säätilasta:

```
{"coord":{"lon":23.2,"lat":61.64},"weather":
[{"id":804,"main":"Clouds","description":"overcast
clouds","icon":"04d"}],"base":"stations","main":
{"temp":44.83,"feels_like":29.01,"temp_min":44.6,"temp_max":45,"pressure":1023,"humidit
y":39,"visibility":10000,"wind":{"speed":19.46,"deg":230},"clouds":
{"all":97},"dt":1585063198,"sys":
{"type":1,"id":1360,"country":"FI","sunrise":1585023278,"sunset":1585068720},"timezone"
:7200,"id":659184,"name":"Hämeenkyrö","cod":200}
```

Huomataan, että näinkin pienestä paikkakunnasta löytyy hyvin kattava datapaketti. Aion itse hyödyntää sovellukseeni ainakin "weather", "description" ja "main" rakenteista saatavaa tietoa.

Koodimuunnokset

Sääsovelluksen toimeenpaneminen aiheutti eniten lisäyksiä ja muokkauksia sovellustason (weather) *views.py*, *models.py* ja *forms.py* sekä uuden html- sivun *index.html* -tiedostoihin. Seurasin pitkälti Scotch- verkkosivun noudattamaa oheistusta sovelluksen toteuttamiseen, mutta otin myös pieniä tärppejä Slack- alustalta löytyneiltä aiemmilta toteutuksilta. Sääsovelluksessa *views.py* pyytää API:sta tietyn linkin kautta hakemaan kyseiset säätiedot, mihin pyritään sisällyttämään kaikki käyttäjän syöttämät paikkakunnat, joista hän haluaa tarkastella säätilaa. Ohjelma mahdollistaa siis uusien paikkakuntien lisäämisen suoraan sivustolla olevan napin kautta. Views.py- tiedoston koodi näytti tässä vaiheessa tältä:



```
1 from django.shortcuts import render
2 import requests
3
4 def index(request):
5     url = 'http://api.openweathermap.org/data/2.5/weather?q={}&units=imperial&appid=XXXXXXXXXXXXXXXXXXXX'
6     city = 'Las Vegas'
7     city_weather = requests.get(url.format(city)).json() #request the API data and convert the JSON to Python data type
8
9     return render(request, 'weather/index.html') #returns the index.html template, 'weather/index.html' #returns the i
```

Oma APIKEY on yksilöivä ja salassa pidettävä, joten suttasin sen kyseisestä url -linkistä piiloon.

Weather -kansion alle luotiin uusi templates -kansio, johon lisättiin siis kyseinen index.html -tiedosto, joka mahdollisti sääsovelluksen verkkosivun näkymän muokkaamisen siten, että sivusto hakee käyttäjän syöttämän paikkakuntien säätilat. Ohessa koodinpätkä html -tiedoston näkymästä:


```

forms.py x urls.py x index.html x
1  <!-- the_weather/weather/templates/weather/index.html -->
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <title>Sääpalvelu</title>
9      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.6.2/css/bulma.css" />
10 </head>
11 <body>
12     <section class="hero is-primary">
13         <div class="hero-body">
14             <div class="container">
15                 <h1 class="title">
16                     Jannun Sääpalvelu
17                 </h1>
18                 <h2 class="sub-title">
19                     Katso miltä näyttää sää eri puolella maailmaa!
20                 </h2>
21             </div>
22         </div>
23     </section>
24     <section class="section">
25         <div class="container">
26             <div class="columns">
27                 <div class="column is-offset-4 is-4">
28                     <form method="POST">
29                         {% csrf_token %}
30                         <div class="field has-addons">
31                             <div class="control is-expanded">
32                                 {{ form.name }}
33                             </div>
34                             <div class="control">
35                                 <button class="button is-info">
36                                     Lisää paikkakunta
37                                 </button>
38                                 <input class="button" type="reset" value="Tyhjennä">
39                             </div>
40                         </div>
41                     </form>
42                 </div>
43             </div>
44         </div>
45     </section>
46     <section class="section">
47         <div class="container">
48             <div class="columns">
49                 <div class="column is-offset-4 is-4">
50                     {% for weather in weather_data %}
51                     <div class="box">
52                         <article class="media">
53                             <div class="media-left">
54                                 <figure class="image is-50x50">
55                                     
56                                 </figure>
57                             </div>
58                             <div class="media-content">
59                                 <div class="content">
60                                     <p>
61                                         <span class="title">{{ weather.city }}</span>
62                                         <br>
63                                         <span class="subtitle">{{ weather.temperature }}° F</span>
64                                         <br>
65                                         {{ weather.description }}
66                                     </p>
67                                 </div>
68                             </div>
69                         </article>
70                     </div>
71                     {% endfor %}
72                 </div>
73             </div>
74         </div>
75     </section>
76     <footer class="footer">
77 </footer>
78 </body>
79 </html>

```

Views.py -tiedostoon tehtiin muokkaukset muun muassa request -funktion toteutuksesta, API -urlin hakemisesta, sanakirjan luomisesta säätilaan tallennettavista tiedoista (lämpötila, kuvaus, kuvake) ja tietojen lähettämisestä html -näkömään. Ohessa kuva lopullisesta views.py -tiedostosta:

```

1 from django.shortcuts import render
2 import requests
3 from .models import City
4 from .forms import CityForm
5
6 def index(request):
7     url = 'http://api.openweathermap.org/data/2.5/weather?q={}&units=imperial&appid=176a29245cd1ba498c483a37e43349f1'
8
9     if request.method == 'POST':
10         form = CityForm(request.POST)
11         form.save()
12
13     form = CityForm()
14     cities = City.objects.all()
15     weather_data = []
16
17     for city in cities:
18         r = requests.get(url.format(city)).json()
19
20         weather = {
21             'city': city,
22             'temperature': r['main']['temp'],
23             'description': r['weather'][0]['description'],
24             'icon': r['weather'][0]['icon']
25         }
26
27         weather_data.append(weather)
28
29     context = {'weather_data': weather_data, 'form': form}
30     return render(request, 'weather/index.html', context)

```

Viittaamme vielä juuri luotuun views -näkömään urls.py -tiedoston avulla. Models.py -tiedostoon luotiin tietokanta, johon tallennetaan ne paikkakunnat, joille halutaan lisätä tietoja säätilasta. Esitetään oheisia toimenpiteitä seuraavaksi:

```

1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index),
6 ]

```

```

1 from django.db import models
2
3 class City(models.Model):
4     name = models.CharField(max_length=25)
5
6     def __str__(self):
7         return self.name
8
9     class Meta:
10         verbose_name_plural = 'cities'

```

Admin.py -tiedostoon rekisteröidään vielä kyseiset toiminnot järjestelmänvalvojana hallintapaneeliin admin.py tiedostoa muokkaamalla, sekä forms.py -tiedosto mahdollistaa paikkakuntien lisäämisen sivuston kautta. Nämä toimenpiteet näyttäivät tältä:

```

admin.py | forms.py | urls.py | index.html | views.py | models.py
1 from django.contrib import admin
2 from .models import City
3
4 admin.site.register(City)

forms.py
1 from django.forms import ModelForm, TextInput
2 from .models import City
3
4 class CityForm(ModelForm):
5     class Meta:
6         model = City
7         fields = ['name']
8         widgets = {'name': TextInput(attrs={'class': 'input', 'placeholder': 'City Name'})}
9

```

Näin ollen kaikki tarvittavat koodit on lisätty sääsovellusta varten. Seuraavassa osiossa esitän vielä sivuston näkymää.

Sivusto

Oheiset toimenpiteet tehtynä, näyttää verkkosivusto siten seuraavalta:

Jannun Sääpalvelu
Katso miltä näyttää sää eri puolella maailmaa!

City Name

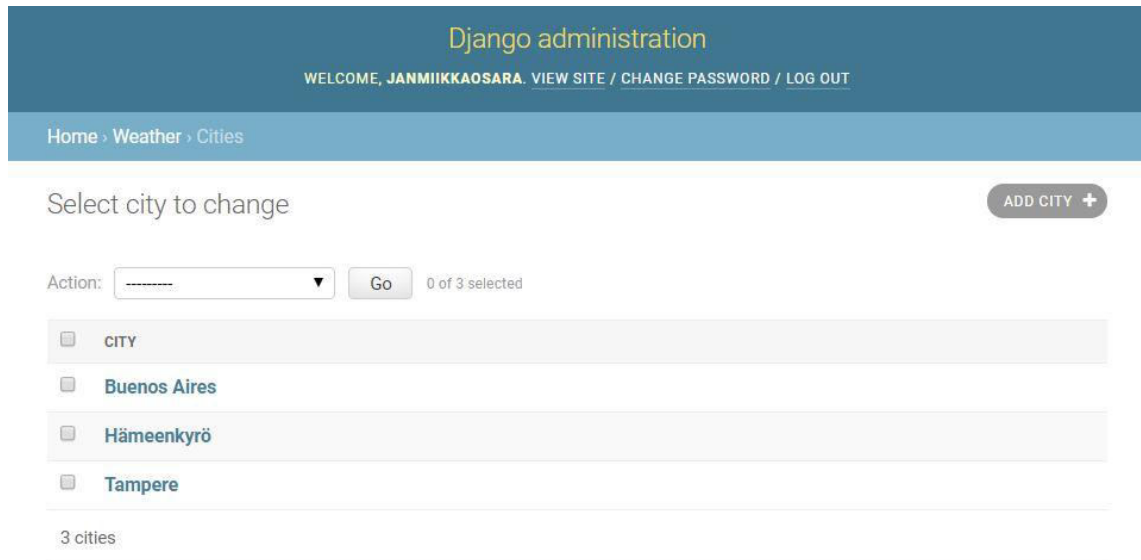
Näkymään voidaan siis syöttää halutut paikkakunnat, esimerkiksi Tampere, Hämeenkyrö ja Buenos Aires. Näin ollen API:n kautta kerätään kyseisten paikkakuntien säätiiedot, minkä jälkeen saadaan sivusto näyttämään tältä:

Jannun Sääpalvelu
Katso miltä näyttää sää eri puolella maailmaa!

City Name

- Tampere**
39.31° F
overcast clouds
- Hämeenkyrö**
38.32° F
broken clouds
- Buenos Aires**
83.05° F
few clouds

Nyt kyseiset paikkakunnat ovat tallentuneet verkkosivun admin -näkymään, joista voidaan myöhemmin käydä poistamassa juuri lisätty paikkakunta. Admin/city -sivusto näyttää siis tältä:



The screenshot shows the Django administration interface for the 'city' model. At the top, there is a blue header with the text 'Django administration' and a welcome message for 'JANMIIKKAOSARA' with links for 'VIEW SITE', 'CHANGE PASSWORD', and 'LOG OUT'. Below the header is a breadcrumb trail: 'Home > Weather > Cities'. The main content area has a title 'Select city to change' and an 'ADD CITY +' button. Below this is an 'Action:' dropdown menu with a 'Go' button and a status '0 of 3 selected'. A table lists three cities: 'CITY', 'Buenos Aires', 'Hämeenkyrö', and 'Tampere'. At the bottom, it says '3 cities'.

<input type="checkbox"/>	CITY
<input type="checkbox"/>	Buenos Aires
<input type="checkbox"/>	Hämeenkyrö
<input type="checkbox"/>	Tampere

3 cities

Täten voidaan siis sanoa sääsovelluksen toimivan oikeaoppisesti. Harjoitustyön neljännessä vaiheessa on tarkoitus visualisoida näkymästä käyttäjäystävällisempi ja monin puolin mielekkäämpi ja monipuolisempi.

Helpot ja vaikeat asiat

Helpot asiat:

- Sääsovelluksen suoraviivainen ja helppohko toteuttaminen hyvän tukimateriaalin saattamana
- Kattava avun saaminen muun muassa Slack -alustalta, Youtubesta ja muista verkkosivuista

Vaikeat asiat:

- Tiettyjen koodinpätkien muokkaaminen ja virheiden löytäminen omalta koodieditorilta
- Scotchin tutoriaalissa oli yksi virhe, joten sen etsimiseen kului melkoinen tovi.
- Sovelluksen toiminnan ollessa aiempia vaiheita monipuolisempi, oli myös tiettyjen asioiden soveltaminen vaikeampaa

Lähdemateriaalit

Scotch – Tutoriaali API-rajapinnan toteutuksesta: <https://scotch.io/tutorials/building-a-weather-app-in-django>

Pretty Printed, Youtube – Tutoriaali API-rajapinnan toteutuksesta: <https://www.youtube.com/watch?v=v7xjdXWZafY&t=1438s>

Lisäksi otin muutamia tärpejä aiemmista harjoitustyötoteutuksista Slack-alustalta.

Janmiikka Osara, janmiikka.osara@tuni.fi

HARJOITUSTYÖ, VAIHE 4

TLO-32400 Ohjelmallinen sisällönhallinta 2020

Vaiheen 4 toimenpiteet

Harjoitustyön neljännen ja viimeisen vaiheen tarkoitus on visualisoida datarajapinnasta tuotua dataa käyttäjäystävällisempään muotoon. Lisäksi harjoitustyön web-sovellus on saatettava lopulliseen, sulavaan ja selkeään muotoon.

Lopuksi tarkoitus on tuoda koodit ja dokumentaatio GitHubiin tarkasteltavaksi.

Visualisointi ja toiminnallisuuden lisääminen

Viimeisen vaiheen jälkeen sääsovellukseni jäi vielä melko alkeelliseen muotoon, joten neljännessä vaiheessa olisi tarkoitus viimeistellä sääsovellus selkeään muotoon. Ensinnäkään käyttäjä ei pystynyt poistamaan näkymästä juuri lisättyä paikkakuntaa, joten tarkoitus oli lisätä tietty funktio, joka mahdollistaa tietyn paikkakunnan poistamisen näkymältä, klikkaamalla paikkakunnan oikeassa yläkulmassa olevaa ruksi-painiketta. Lisäksi tein funktion, eli painikkeen sivustolle, josta käyttäjä voi poistaa kaikki näkymälle lisätyt paikkakunnat. Seuraavat vaiheet edellyttivät muutoksia sekä views.py-, urls.py- ja weather.html- tiedostoihin. Ohessa esitän koodimuunnoksia:

```
def delete_city(request, city_name):
    city.objects.get(name=city_name).delete()
    return redirect('home')

def deleteAll(request):
    City.objects.all().delete()

    return redirect('home')
```

```
urlpatterns = [
    path('', views.index, name='home'),
    path('delete/<city_name>/', views.delete_city, name='delete_city'),
    path('deleteall/', views.deleteAll, name='delete-all')
```

Lisäykset sekä views.py- ja urls.py- tiedostoihin.

```
<div class="control" align="center">
    <button class="btn btn-danger remove-all">
    <a href="{% url 'delete-all' %}">Poista kaikki paikkakunnat</a></button></div>

<div class="media-right">
    <a href="{% url 'delete_city' weather.city %}">
        <button class="delete"></button>
    </a>
</div>
```

weather.html- tiedoston lisäykset

Tämän jälkeen tein sovellukseen viestinnällisen ominaisuuden, jonka avulla sivusto ilmoittaa käyttäjälle paikkakunnan syöttämisen jälkeen, mikäli tapahtuu jokin seuraavista tapauksista

- 1) Paikkakunta lisätty onnistuneesti

- 2) Käyttäjä ei voi lisätä jo olemassa olevaa paikkakuntaa
- 3) Käyttäjä ei voi lisätä paikkakuntaa, jota ei ole olemassa

Ohessa esitän seuraavia näkymän viestejä oheisista tapahtumista järjestyksessä, sekä koodimuunnoksia sekä views.py- että waether.html- tiedostoon.

```
err_msg = ''
message = ''
message_class = ''

if request.method == 'POST':
    form = CityForm(request.POST)

    if form.is_valid():
        new_city = form.cleaned_data['name']
        existing_city_count = city.objects.filter(name=new_city).count()

        if existing_city_count == 0:
            r = requests.get(url.format(new_city)).json()

            if r['cod'] == 200:
                form.save()
            else:
                err_msg = 'Paikkakuntaa ei ole olemassa!'
        else:
            err_msg = 'Paikkakunta on jo lisätty!'

    if err_msg:
        message = err_msg
        message_class = 'is-danger'
    else:
        message = 'Paikkakunta lisätty onnistuneesti!'
        message_class = 'is-success'
```

```
{% if message %}
    <div class="notification {{ message_class }}">{{ message }}</div>
{% endif %}
```

Lisäksi tällä hetkellä paikkakunnasta sai vain lämpötilan tiedot, sekin Farenheit-muodossa, joten päätin lisätä näkymään maan tunniste, lämpötilan celsius-muodossa, "tuntuu kuin"- lämpötilan sekä tuulen nopeuden kyseiseltä paikkakunnalta. Kaikki tiedot löytyivät samasta API-linkistä, eli OpenWeatherMapin tarjoamasta avoimesta datapaketista. Farenheitin, sekä mailia per tunnin muutos täytyi kuitenkin muuntaa format-muotoilulla itse Pythonissa. Edellä mainitut toimenpiteet vaativat muutoksia sekä views.py- että waether.html- tiedostoihin.

```
weather = {
    'city': city,
    #vaihdetaan lämpötila celsius-asteiksi kahden desimaalin tarkkuudella"
    'temperature': "{:.1f}".format(((r['main']['temp'] - 32) * 0.55)),
    'speed': "{:.1f}".format((r['wind']['speed'] * 0.44704)),
    'feels_like': "{:.1f}".format(((r['main']['feels_like'] - 32) * 0.55)),
    'description': r['weather'][0]['description'],
    'icon': r['weather'][0]['icon'],
    'country': r['sys']['country'],
```



Hämeenkyrö, FI

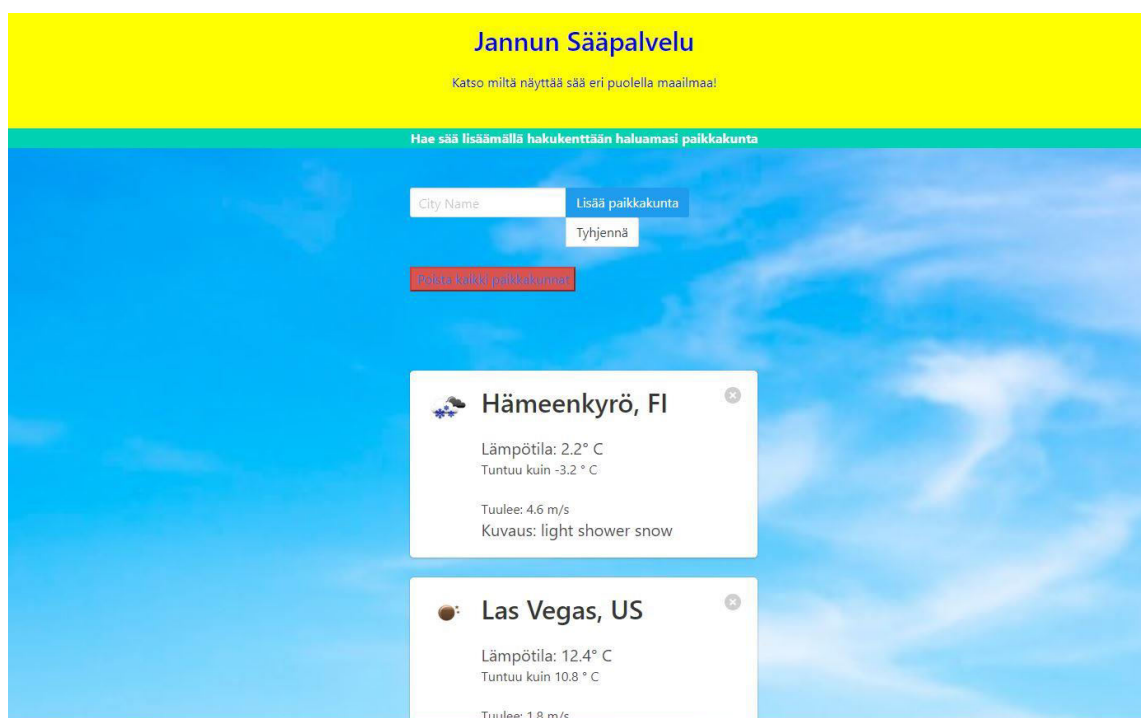
Lämpötila: 2.6° C
Tuntuu kuin -5.8° C

Tuulee: 8.7 m/s
Kuvaus: broken clouds

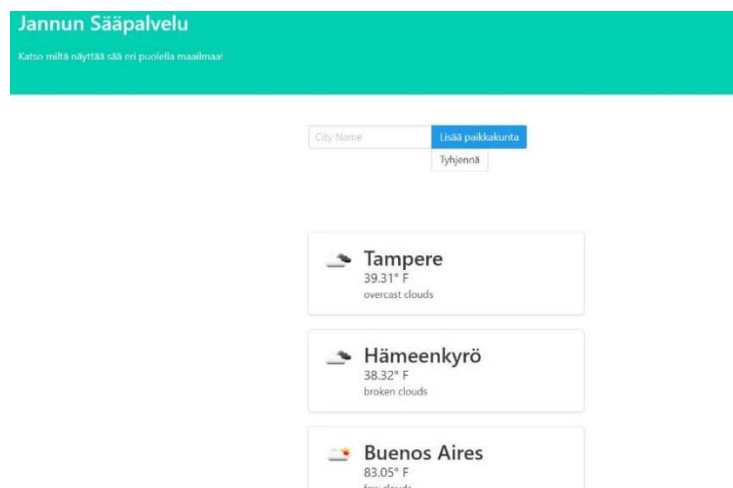
Ohessa siis funktio farenheitista celsiukseksi, mailia per tunti metriä per sekuniksi, sekä tietyn paikkakunnan säätilan näkymä sivustolla toimenpiteiden jälkeen.

Lisäksi muokkailin sivuston näkymää selkeämpään muotoon muuntelemalla pitkälti HTML/CSS tason koodia. Lisäsin muun muassa taivas-aiheisen taustakuvan näkymään sekä muokkasin tekstien ja bannereiden värejä ja sijaintia.

Kaikkine muutoksineen sivusto näyttää käyttäjän avatessa sivun, johon on lisätty valmiiksi Hämeenkyrön sekä Las Vegasin säätila, tältä.



Aiemmassa vaiheessa sivuston näkymä oli tämän kaltainen:



GitHubin integrointi

Olin jo harjoitustyön ensimmäisessä vaiheessa kontriboinut oman projektini GitHubiin omaksi reporitorikseen. Tarkoitus oli enää päivittää oma projektini lopulliseen muotoon PowerShellistä komannolle *git push*. Nyt oma koodini ja dokumentaatio löytyy GitHubista, linkistä: <https://github.com/janmiikkaosara/OHSIHA>.

Helpot ja vaikeat asiat

Helpot asiat:

- HTML/CSS muotoilu oli suhteellisen näppärää, kun käytössä oli suhteellisen kattava lähdemateriaalisto
- Visualisointi oli helppoa, kun dataan oli jo ehtinyt tutustua aiemmassa vaiheessa

Vaikeat asiat:

- Tiettyjen koodinpätkien muokkaaminen ja virheiden löytäminen omalta koodieditorilta
- En oikein keksinyt, mitenkä kehittää yksinkertaista sääsovellusta omien taitojeni mukaan

Lähdemateriaalit

Viestinnälliset ominaisuudet paikkakunnan lisäämisestä sekä tietyn paikkakunnan poistaminen näkymältä: Pretty Printed, Youtube – Tutorials API-rajapinnan toteutuksesta osa 2: <https://www.youtube.com/watch?v=oPuYTGyW4dU>

HTML/CSS muotoilut sääsovelluksen näkymään.
<https://www.w3schools.com/html/default.asp>