Harjoitustyön ensimmäinen vaihe

Valitut teknologiat

Kehitysympäristö ja ohjelmointikieli

Harjoitustyöni kehistysympäristöksi valitsin luennollakin suositellun Djangon (versio 2.0.1), sillä aiempaa kokemusta web-kehityksestä ei ollut. Djangon sanelemana ohjelmointikieleksi valikoitui siis Python (versio 3.6.4). Aiempaa kokemusta Pythonin käytöstä on yhden ohjelmointikurssin verran, joten kieli ei ole täysin uusi. Tekstieditorina kuitenkin tuolloin PyCharm.

Tietokanta ja palvelin

Tietokantana käytän Djangon mukana oletuksena tullutta SQLite:a ja sen todentamiseen DB Browser for SQLite -työkalua. Palvelimena toimii, myöskin pakettiin kuuluva Djangon oma, kehitystyöhön tarkoitettu kevyt web-palvelin.

Teksieditori ja versionhallinta

Web-ohjelmoinnin apuna toimii tekstieditori Atom. Aikaisempaa tekstieditorikokemusta on vain PyCharmin osalta, mutta koen Atomin käytön selkeämmäksi. Harjoitustyöhön liittyvät koodit sekä dokumentaation aion tallentaa Githubiin.

Asennus ja muut harjoitustyövaiheet

Python 3.6.4. ja pip

Python-paketin asentaminen onnistui kätevästi osoitteesta https://www.python.org/downloads/mac-osx/. Asennusvaiheessa kannatti varmistaa "Customize installation"-kohdasta, että asennetuksi tulee myös Pythonille tarkoitettu paketinhallintajärjestelmä pip. Ladatun python-version ja siihen kuuluvien pakettien tiedot saa näkyviin seuraavilla komennoilla:

```
$ python3
```

```
$ pip3 list
```

virtualenv

Pythonin jälkeen asensin virtualenv-paketin. Latauksen sai suoritettua kirjoittamalla komentotulkkiin seuraava komento:

```
$ pip3 install virtualenv
```

Django

Edellisten latausten jälkeen asensin Djangon edellisen asennuksen tavoin pip:ä käyttäen. Asennus onnistui kirjoittamalla komentotulkkiin seuraava komento:

```
$ pip3 install django
```

Varmistu siitä, että Djangon lataus on suoritettu, onnistuu komennolla:

```
$ python3
>>> import django
>>> print(django.get_version())
```

Ensimmäinen Django-projekti

Kun Django on ladattu, voidaan luoda ensimmäinen projekti. Se onnistuu komennolla:

```
$ django-admin startproject projektin_nimi
```

Komento luo projektille uuden kansion. Projektin saa käyntiin lokaalisti osoitteessa http://127.0.0.1:8000/ seuraavilla komennoilla:

```
$ cd projektin_nimi
$ python3 manage.py runserver
```

Applikaation lisääminen

Django-applikaation, eli koko web-toteutukseen yhden toiminnallisuuden lisääminen onnistuu projektihakemiston sisällä komennolla:

```
$ python3 manage.py startapp applikaation_nimi
```

Luentopäiväkirjan dokumentaatiossa esitettiin, että django-applikaatio tulisi linkittää projektiin lisäämällä luotu applikaatio settings.py -tiedostossa kohtaan ALLOWED_HOSTS. Sovellus toimi kuitenkin ilman tämän kohdan suorittamista.

Kokeilusovellus

Seuraavaksi loin "kokeilusovelluksena" dynaamisen html-sivun. Tämä onnistui ensin lisäämällä templatet settings.py -tiedostossa seuraavanlaisesti:

```
'DIRS': [os.path.join(BASE_DIR, "templates"),],
```

Tällöin Django hakee `projektin_nimi/templates/` -hakemistosta kaikki templatet. Seuraavaksi luodaan projektin alle `templates` -hakemisto, johon lisätään .html-tiedosto. Jotta `views.py` tietää, mihin sen tuoma sisältö kiinnitetään ja `urls.py` -tiedostossa luotua halutut polut, voidaan sisältö luoda esim. seuraavanlaisesti:

Sisältöä sivu.html-tiedostosta:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Testiapp</title>
  </head>
  <body>
    <h1>{{sisalto}}</h1>
  </body>
</html>
```

Sisältöä views.py-tiedostosta:

```
from django.shortcuts import render
from django.http import HttpResponse

def index1(request):
  eka_dict = {'sisalto': "Tämä on tekstiä views.py:stä!"}
  return render(request, 'app1/sivu.html', context=eka_dict)

def index2(request):
```

```
    eka_dict = {'sisalto': "Tämä on myös tekstiä views.py:stä!"}
    return render(request, 'app1/sivu.html', context=eka_dict)
```

Sisältöä urls.py-tiedostosta:

```
from django.urls import include, path
from django.contrib import admin
from app1 import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index1),
    path('testi/', views.index2),
]
```

Editorin laajennusosat

Atom-tekstieditorin Preferences-ikkunasta löytyy *Packages*-valinta, joka esittää kaikki ladatut paketit. Mm. *markdown-preview* -paketti on kätevä aloittelijalle toivotun ulkoasun varmistamiseen, sillä käyttäjänäkymän saa esille .md-tiedoston rinnalle sitä muokattaessa. *Install*-valinta tarjoaa mahdollisuuden hakea myös muita ladattavia paketteja. Itse asensin *platformio-ide-terminal* -paketin, joka mahdollistaa komentotulkin käyttämisen tekstieditorista käsin. Hyvä puoli on myös se, että paketti tarjoaa mahdollisuuden useamman kuin yhden komentoikkunan samanaikaiseen käyttöön.

Myös *markdown-pdf* -paketti on kätevä työkalu .md-tiedoston exporttaamiseksi pdf-tiedostoksi suoraan editorista käsin. Yksi mahdollisuus olisi käyttää web-tekstieditori Dillinger:iä, jossa voi samalla tavalla tulostaa markdow-tiedoston pdf-tiedostoksi.

Sovelluksen yhdistäminen tietokantaan

Tähän mennessä olen toteuttanut yksinkertaisen kirjautumis- ja rekisteröitymispalvelun sovellukseen. Hyvät ohjeet palvelun toteuttamiseen löytyvät osoitteesta https://wsvincent.com/django-user-authentication-tutorial-login-and-logout/. Ensimmäiseksi Djangon omaa auntetikaatio-sovellusta käyttäen loin sisäänkirjautumistoiminnon muokkaamalla projektin `urls.py` -tiedostoa seuraavanlaiseksi:

```
from django.contrib import admin
from django.urls import path, include

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

Seuraavaksi loin sivustolle näkymän `login.html` -tiedostoon:

```
<h2>Login</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
```

Jotta Djanog osaisi hakea `projektin_nimi/templates/` -hakemistosta kaikki templatet, `settings.py` -tiedostoon tuli tehdä seuraava muutos kohtaan TEMPLATES:

```
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        ...
    },
]
```

Seuraavaksi kirjautumistoimintoa testaakseni loin admin-käyttäjän seuraavalla komennolla:

```
$ python3 manage.py createsuperuser
```

Rekisteröitymistä varten loin sovelluksen komennolla:

```
$ django-admin startapp sovelluksen_nimi
```

Projektin `settings.py` -tiedostoon tuli myös lisätä luotu sovellus:

```
INSTALLED_APPS = [
    ...
    'accounts',
]
```

Myös rekisteröitymistä varten tulee luoda näkymä, jonka loin `projektin_nimi/templates/` -hakemistoon. Ote `signup.html` -tiedoston sisällöstä:

```
{% extends 'base.html' %}

{% block title %}Sign Up{% endblock %}

{% block content %}
  <h2>Sign up</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Sign up</button>
  </form>
{% endblock %}
```

Muutoksia tuli tehdä myös sovelluksen `views.py` - ja `urls.py` -tiedostoihin sekä projektin `urls.py` -tiedostoon:

```
#sovelluksen_nimi/urls.py

from django.urls import path

from . import views


urlpatterns = [
    path('signup/', views.SignUp.as_view(), name='signup'),
]
```

```
#sovelluksen_nimi/views.py

from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy
from django.views import generic


class SignUp(generic.CreateView):
    form_class = UserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'signup.html'
```

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic.base import TemplateView

urlpatterns = [
```

```
    path('', TemplateView.as_view(template_name='home.html'), name='home'),
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Rekisteröintipalvelun luonnissa käytetään apuna Djangon mukana tulleita valmiita näkymiä, kuten `UserCreationForm`ia. Yksinkertaisen kotisivunäkymän luomiseksi tein `templates`-kansioon myös `base.html` - sekä `home.html` -tiedostot:

```
#templates/base.html

<html>
<head>
  <meta charset="utf-8">
  <title>{% block title %}Django Auth Tutorial{% endblock %}</title>
</head>
<body>
  <main>
    {% block content %}
    {% endblock %}
  </main>
</body>
</html>
```

```
#templates/home.html

{% extends 'base.html' %}

{% block title %}Home{% endblock %}

{% block content %}
{% if user.is_authenticated %}
  Hi {{ user.username }}!
  <p><a href="{% url 'logout' %}">logout</a></p>
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'login' %}">login</a>
{% endif %}
{% endblock %}
```

Ajatuksia harjoitustyöstä

Ilmenneet hankaluudet

- Djangon latauksen yhteydessä komentotulkki valitti käyttäjän sudo-oikeuksien puuttumisesta, kun yritin tehdä muutoksia normaalikäyttäjällä ylläpitäjäkäyttäjän sijaan. Ongelma kuitenkin ratkesi järjestelmäasetuksista ylläpito-oikeuksia muuttamalla ja suorittamalla asennusprosessin uudestaan ohjeiden mukaan. Muutoin pakettien asennuksessa ei ole ilmennyt juuri yhtään vaikeuksia.
- Koska aiempaa ohjelmointikokemusta ei juurikaan ole, perusasioiden, kuten komentotulkkityöskentelyn, opetteleminen vaatii paljon omaa panosta. Onneksi opettelun tueksi on kuitenkin olemassa paljon erilaisia tutorialeja.
- Hankaluuksia on ollut myös ideoida harjoitustyön sisältöä niin, että se tarjoaisi vaadittavien toiminnallisuuksien lisäksi myös jonkin oikeasti hyödyllisen käyttäjän elämää helpottavan palvelun.

Harjoitustyöidea

Harjoitustyönä ajattelin toteuttaa eräänlaisen "*Kulttuuritsekkauslista*"-palvelun. Ideana on hakea avoimista rajapinnoista dataa tapahtumista ympäri Suomea, ja toteuttaa käyttäjälle mahdollisuus listata saadun tapahtumainformaation perusteella kiinnostavia vaihtoehtoja muokattavaan check-listiin. Hyvänä lisäarvona palvelu voisi tarjota käyttäjälle myös sääinformaatiota valittujen tapahtumien ajankohdalle.
Eventmore on hyvä esimerkki tällaisesta tapahtumarajapintoja hyödyntävästä palvelusta. Aion todennäköisesti käyttää mm. Helsinki Region Infoshare:n sekä Lounastiedon tarjoamaa avointa dataa palveluni kehittämisessä.

Hyödyllisiä lähteitä

- **Markdown Cheatsheet**:
  https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
- **Djangon dokumentaatio**:
  https://docs.djangoproject.com/en/2.0/
- **Django Girls Tutorial**:
  https://tutorial.djangogirls.org/en/
- **William S. Vincent - Django Login/Logout Tutorial**:
  https://wsvincent.com/django-user-authentication-tutorial-login-and-logout/