Aleksi Tuominen

TLO-32400 Ohjelmallinen sisällönhallinta

2. vaihe: Esimerkkisovelluksen toteutus

Toisen vaiheen palautuksessa laaditaan yksinkertainen esimerkkisovellus, joka mahdollistaa tietojen lukemisen, lisäämisen, muokkaamisen ja poistamisen (CRUD). Esimerkkisovelluksen sisällöllä ei sinänsä ole väliä, tarkoitus on vain harjoitella CRUD-toimintojen luomista.

Luodaan aluksi Djangolla uusi sovellus:

```
C:\Users\DELL\Desktop\Djangotest\mysite>manage.py startapp esimerkki
```

Aktivoidaan myös virtuaaliympäristö sovellusta varten

```
C:\Users\DELL\Desktop\Djangotest\mysite>cd venv/Scripts

C:\Users\DELL\Desktop\Djangotest\mysite\venv\Scripts>activate

(venv) C:\Users\DELL\Desktop\Djangotest\mysite\venv\Scripts>cd ../..
```

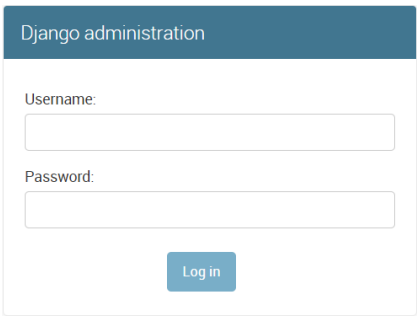
Asennetaan Django virtuaaliympäristössä ja tehdään tietokantapäivitykset varmuuden vuoksi.

```
(venv) C:\Users\DELL\Desktop\Djangotest\mysite>python manage.py makemigrations
No changes detected
```

Luodaan pääkäyttäjä

```
(venv) C:\Users\DELL\Desktop\Djangotest\mysite>python manage.py createsuperuser
Username (leave blank to use 'dell'): aleksi
Email address:
Password:
Password (again):
```

Ajetaan palvelin komennolla python manage.py runserver ja navigoidaan selaimella localhost-porttiin 8000. Alihakemisto /admin/ ohjaa sisäänkirjautumissivulle, johon syötetään edellä luodut tunnukset.



Tässä kohtaa tunnuksilla voi lähinnä muokata käyttäjien tietoja. Varmistetaan kuitenkin, että ne toimivat.

Aloitetaan varsinaisten CRUD-toimintojen toteutus. Luodaan yksinkertainen Product-luokka suoraan YouTube-oppaan mukaan (https://www.youtube.com/watch?v=Kf9KB_TZY5U)

Aluksi luodaan models.py-tiedostoon luokka, sen attribuutit ja metodit.

```python
from django.db import models

class Product(models.Model):

    description = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=9, decimal_places=2)
    quantity = models.IntegerField()

    def __str__(self):
        return self.description
```

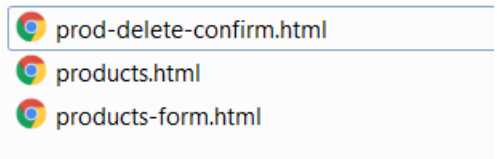
Luodaan 4 toiminnolle URL-tunnisteet urls.py-tiedostoon

```python
from django.urls import include, path
from .views import list_products, create_product, update_product, delete_product

urlpatterns = [
    path('', list_products, name='list_products'),
    path('new', create_product, name='create_products'),
    path('update/<int:id>/', update_product, name='update_product'),
    path('delete/<int:id>/', delete_product, name='delete_product'),
]
```

Varsinainen toiminnallisuus rakennetaan views.py-tiedostoon kaikille neljälle operaatiolle. Esimerkkinä päivitysoperaatio (U)

```python
def update_product(request, id):

    product = Product.objects.get(id=id)
    form = ProductForm(request.POST or None, instance=product)

    if form.is_valid():
        form.save()
        return redirect('list_products')

    return render(request, 'products-form.html', {'form': form, 'product': product})
```

Luodaan tarvittavat html-lomakkeet tietojen näyttämistä varten templates/ - kansioon

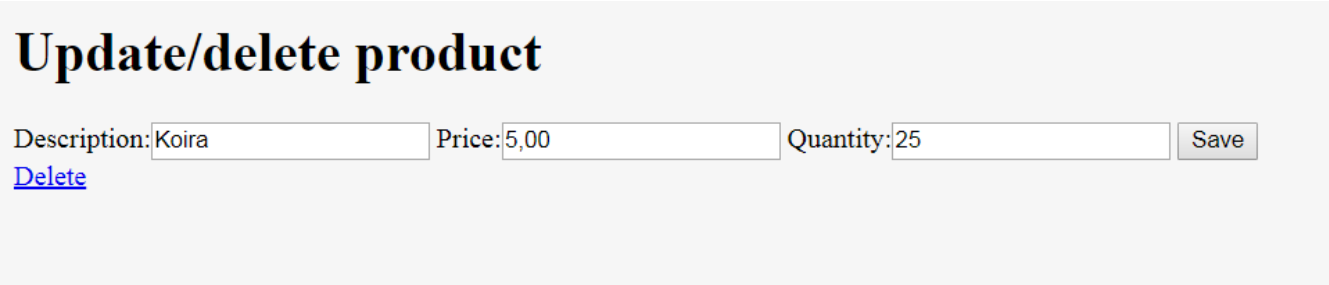


Kun tarvittavat toiminnot on toteutettu, voidaan ajaa palvelin runserver-komennolla

- Koira
- Kissa
- Kettu

New product

Luotu sivu näyttää yksinkertaisuudessaan tältä. Tuotetta pääsee muokkaamaan tai poistamaan klikkaamalla sitä:



Yksinkertaiset CRUD-toiminnot on siis toteutettu ja pääkäyttäjän kirjautuminen on mahdollista. Seuraavaksi toteutetaan muiden käyttäjien rekisteröinti ja kirjautuminen.

Djangoon on sisäänrakennettu sisäänkirjautumistoiminto. Sille tulee vain luoda template. Aloitetaan lisäämällä templates-kansio urls.py-tiedostoon:

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

Lisätään myös accounts-alihakemisto urls-tiedostoon:

```
path('accounts/', include('django.contrib.auth.urls')),
```

Luodaan kansio templates/registration ja sinne tiedosto login.html.

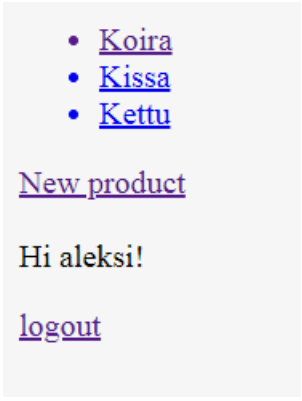
```
<body class "body" style=background-color:#f6f6f6>

    <h2>Login</h2>

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

Ajetaan palvelin uudestaan. Nyt navigoitaessa osoitteeseen localhost:8000/accounts/login saadaan seuraavannäköinen sivu:

Kirjautuminen aiemmin luoduilla pääkäyttäjän tunnuksilla toimii. Lisätään seuraavaksi uloskirjautumismahdollisuus pääsivulle (aiemmin toteutettu products.html)

```
{% 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 %}
```

Nyt sivusto tunnistaa kirjautuneen käyttäjän ja tarjoaa uloskirjautumispainikkeen



Seuraavaksi toteutetaan rekisteröinti. Luodaan tätä varten uusi sovellus

```
python manage.py startapp accounts
```

Lisätään sovellus projektin urls- ja settings-tiedostoihin kuten edellä.

Seuraavaksi lisätään accounts-sovelluksen omaan urls-tiedostoon rekisteröintipolku:

```python
from django.urls import path

from . import views


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

Luodaan views-tiedostoon luokka rekisteröintiä varten:

```python
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'
```

Luodaan tiedosto signup.html templates-kansioon samaan tapaan kuin aiemmin luotiin login.html

Sign Up

Sign up

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

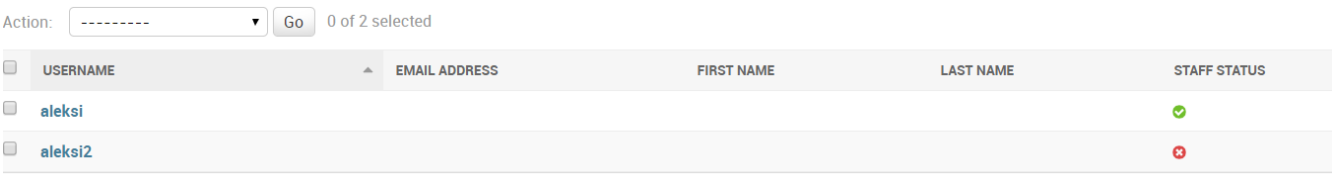
Password:

- 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.

Sign up

Luodaan vielä kokeeksi uusi käyttäjätunnus. Uusi käyttäjä näkyy admin-ikkunassa eli käyttäjä on luotu onnistuneesti.



Sovelluskehitys Djangolla valmiita oppaita seuraamalla on todella helppoa, eikä tähän mennessä ole tullut vastaan kummempia ongelmia. Vikatilanteet johtuvat pääosin kirjoitusvirheistä ja virheviestien avulla ne on helppo paikantaa. Käytin tässä kohtaa apuna seuraavia oppaita:

https://www.youtube.com/watch?v=Kf9KB_TZY5U

https://wsvincent.com/django-user-authentication-tutorial-login-and-logout/

https://wsvincent.com/django-user-authentication-tutorial-signup/

Erityisen helpoksi tällä teknologiavalinnalla osoittautui tietokantojen käyttö. SQLite mahdollistaa tietokantojen käytön ilman tietokantojen manuaalista luontia tai tietokantakyselyjen rakentamista. Toisaalta tämä ei ole oppimisen kannalta kovin hyvä asia. Valitsisin seuraavaan projektiin jonkin muun tietokantaratkaisun.

Haasteellisimpana asiana tässä kohtaa on ehkä HTML-dokumentteihin upotettu jinja, jonka syntaksi ei ole kovin tuttua ja ymmärtäminen on välillä hidasta.