Python for Beginners

Web Programming with Django

Advanced Concepts

Advanced Topics

- File Upload
- Authentication (Login / Logout / Sign up)
- Pagination

File Upload (directories)

- Inside the project directory in the folder “DjangoBasics” create a folder “static” and inside another folder “media”
  → this is the place where uploaded files are stored
- Add the following lines to the settings.py:

```
STATIC_URL = '/static/'
MEDIA_URL = '/media/'

PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
STATIC_ROOT = os.path.join(PROJECT_ROOT, 'static')
MEDIA_ROOT = os.path.join(STATIC_ROOT, 'media')
```

File Upload (urls.py)

- Add the yellow lines to the urls.py to enable the routing of the static media URL

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # the admin urls
    path('admin/', admin.site.urls),
    path('people/', include('demoapp.urls')),  # the people urls are defined in the url.py of the app

    # the entry point (=empty url path) should be redirected to te desired landing page
    path('', lambda request: redirect('person_index_route', permanent=False)),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

File Upload (model)

- Add the field “imagePath” to the Person model:

```
imagePath = models.TextField(blank=True)
```

- Make a new migration and execute it to update the database:

```
python manage.py makemigrations
Migrations for 'demoapp':
  demoapp\migrations\0002_person_imagepath.py
    - Add field imagePath to person

python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, demoapp, sessions
Running migrations:
  Applying demoapp.0002_person_imagepath... OK
```

File Upload (template files)

- Add a file input to the template files “edit.html” and “create.html”:

```
<input type="file" name="image" />
```

- Add the encoding type attribute to the form tag:

```
enctype="multipart/form-data"
```

- The resulting file input looks like this so the user can brows and select a file to upload:

Image: Browse... No file selected.

- You can show the uploaded file:

```
Image: <img src="{{ person.imagePath }}" height="100">
```

File Upload (views.py)

- Create a function “save_image” in views.py:

```
# save_file saves the uploaded file 'image' to the person
def save_image(image_file, person):
    fs = FileSystemStorage()
    if person.imagePath:
        filename = person.imagePath.rpartition('/')[2]
        fs.delete(filename)

    filename = str(person.id) + '_' + image_file.name
    if fs.exists(filename):
        fs.delete(filename)
    fs.save(filename, image_file)
    person.imagePath = fs.url(filename)
```

- And call the function in store and update:

```
if 'image' in request.FILES:
    save_image(request.FILES['image'], p)
```

- Before you delete a Person delete an image file too:

```
# try to get the Person object with the person_id
person = Person.objects.get(id=person_id)
# delete the image file of the person
if person.imagePath:
    fs = FileSystemStorage()
    fs.delete(person.imagePath)
# delete the person
person.delete()
```

Authentication (Apps)

- Open a terminal and create a further app called “accounts”:

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

- Check/add two apps to the installed apps in settings.py:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'demoapp',
    'accounts.apps.AccountsConfig',
]
```

Add the two lines to the end of settings.py:

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL ='/accounts/login/'
```

Authentication (urls.py)

- Add the urls of the accounts-app and auth-app:

```
urlpatterns = [
    # the admin urls
    path('admin/', admin.site.urls),
    path('people/', include('demoapp.urls')),  # the people urls are defined in the url.py of the app
    path('accounts/', include('accounts.urls')),
    path('accounts/', include('django.contrib.auth.urls')),

    # the entry point (=empty url path) should be redirected to te desired landing page
    path('', lambda request: redirect('person_index_route', permanent=False)),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

- The following new url paths can be used:

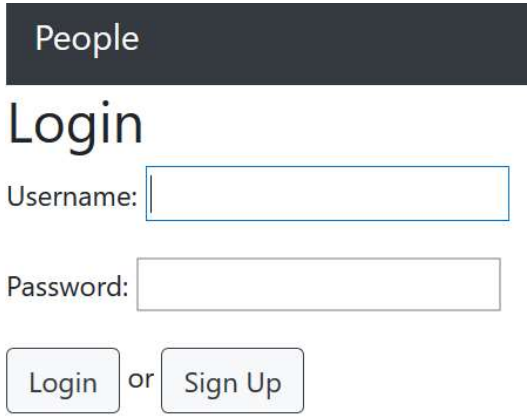
url path	view	name
/accounts/login/	django.contrib.auth.views.LoginView	login
/accounts/logout/	django.contrib.auth.views.LogoutView	logout
/accounts/signup/	accounts.views.SignUp	signup

Authentication (directories and login template)

- In the directory of the accounts app create a new “template” folder
- Inside the “template” folder create a “registration” folder
- In the registration folder create the login.html template file:

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

{% block content %}
  <h2>Login</h2>
    <form method="post">
      {% csrf_token %}
      {{ form.as_p }} {# this loads the complete standard login form #}
      <button type="submit" class="btn bg-light btn-outline-dark">Login</button>
        or
      <a href="{% url 'signup' %}" class="btn bg-light btn-outline-dark">Sign Up</a>
    </form>
{% endblock %}
```



- You can create a custom login.html instead.
  Just use the form inputs with name “username” and “password”

Authentication (base template)

- Modify the base.html template in the “demoapp” template directory to show the user info and the Logout button:


```
<!-- Links -->
{% if user.is_authenticated %}
<ul class="navbar-nav mr-auto">
    <li class="nav-item">
        <a href="{%  url 'person_index_route' %}„ class="btn btn-outline-light">All People</a>
    </li>
    <li class="nav-item">
        <a href="{%  url 'person_create_route' %}„ class="btn btn-outline-light">Create Person</a>
    </li>
</ul>
<ul class="navbar-nav float-right">
    <li class="nav-item">
        <span class="navbar-brand">Hi {{ user.username }}!</span>
        <a href="{% url 'logout' %}" class="btn btn-outline-light">logout</a>
    </li>
</ul>
{%  endif %}
```

Authentication (protect the views)

- In the views.py of the demoapp import the login_required decorator:

```
from django.contrib.auth.decorators import login_required
```

- Protect each view function from anonymous access by decorating it with @login_required

```
# the person_create view returns an empty form to enter the new person data
@login_required
def person_create(request):
    departments = Department.objects.all()
    sex_choices = Person.SEX_CHOICES
    return render(request, 'person/create.html', {'departments': departments, 'sex_choices': sex_choices})
```

Authenticaton (Sign up url and view)

Create a SignUp view + template, if you want new users to register:

- Edit the urls.py of the accounts-app:

```
from django.urls import path
from . import views


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

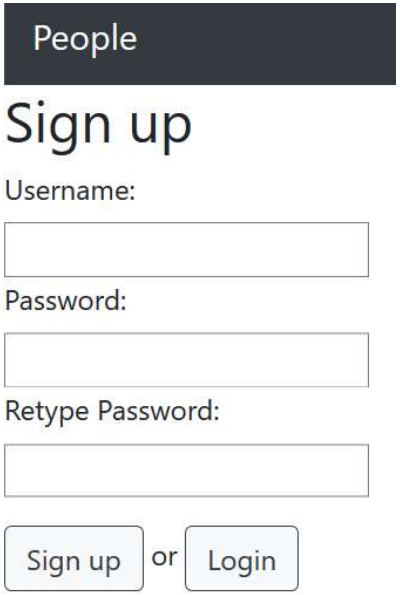
- Edit the views.py of the accounts-app:

```
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 = 'registration/signup.html'
```

Authentication (Sign up template)

- Create and edit the signup.html template file in the accounts-app:




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

{% block content %}
  <h2>Sign up</h2>
  <form method="post">
    {% csrf_token %}
      <p>
        <label>Username:</label><br><input type="text" name="username"/>
        <br>
          <label>Password:</label><br><input type="password" name="password1"/>
        <br>
          <label>Retype Password:</label><br><input type="password" name="password2"/>
      </p>
    <button type="submit" class="btn bg-light btn-outline-dark">Sign up</button>
    or
    <a href="{% url 'login' %}" class="btn bg-light btn-outline-dark">Login</a>

      <br><br>
      {%  for field in form %} {# show all errors after submit #}
          {% if field.errors %}
              <p style="color: red">{{ field.name }}: <br>
              {% for error in field.errors %}
                  - {{ error }} <br>
              {% endfor %}
              </p>
              {%  endif %}
      {% endfor %}
  </form>
{% endblock %}
```

Pagination (view)

If a result set of a database query is big you should paginate the list you show the user into pages so he can page forward and backward

- Modify the person_index function for pagination:

```
def person_index(request):
    #people_list = Person.objects.all()  # get all Person opjects from the database
    people_list = Person.objects.all().order_by('lastname')  # use a ordered queryset for pagination
    page = request.GET.get('page', 1)
    paginator = Paginator(people_list, 10)  # Show 10 people per page

    try:
        people = paginator.page(page)
    except PageNotAnInteger:
        people = paginator.page(1)
    except EmptyPage:
        people = paginator.page(paginator.num_pages)

    return render(request, 'person/index.html', {'people': people})
```

Pagination (template)

- Add the pagination controls to the index.html template:

```
{% if people.has_other_pages %}
  <ul class="pagination">
    {% if people.has_previous %}
      <li><a class="page-link" href="?page={{ people.previous_page_number }}">&laquo;</a></li>
    {% else %}
      <li class="page-item disabled"><span class="page-link">&laquo;</span></li>
    {% endif %}
    {% for i in people.paginator.page_range %}
      {% if people.number == i %}
        <li class="page-item active"><span class="page-link">{{ i }}</span></li>
      {% else %}
        <li class="page-item"><a class="page-link" href="?page={{ i }}">{{ i }}</a></li>
      {% endif %}
    {% endfor %}
    {% if people.has_next %}
      <li><a class="page-link" href="?page={{ people.next_page_number }}">&raquo;</a></li>
    {% else %}
      <li class="disabled"><span class="page-link">&raquo;</span></li>
    {% endif %}
  </ul>
{% endif %}
```