# Programowanie Aplikacji Internetowych

Laboratorium nr 4

Django

Poniżej znajdują się zadania, które należy wykonać w ramach laboratoriów, a następnie sporządzić sprawozdanie w formie archiwum .zip. Plik archiwum powinien mieć nazwę zgodną ze wzorem: PAI_Lab<nr_laboratorium>_<pierwsza_litera_imienia>.<nazwisko_bez_polskich_znaków>.zip, np. PAI_Lab4_J.Kowalski.zip. W archiwum powinna znajdować się poniższa struktura katalogów i plik:

```
\todo
    | - manage.py
    | - db.sqlite3
    | - \templates
    |     | - base.html
    |     | - current_datetime.html
    |     | - hours_ahead.html
    |     \ - <pozostałe_pliki_szablonów>
    | - \todo
    |     | - __init__.py
    |     | -
    \ - \tasks
```

Zadania **muszą** być wykonywane w zadanej kolejności – od 1 do 8.

## Zadanie 1 Instalacja Django w systemie MS Windows

1) Pobrać pakiet instalacyjnny Python 2.7.x ze strony https://www.python.org/downloads/windows/. Należy wybrać Windows x86 / x86-64 MSI installer.
2) Następnie należy pobrać i zainstalować Django:
   a) poprzez program pip (https://pip.pypa.io/en/latest/installing.html):
   ```
   pip install Django==1.7.1
   ```
   b) poprzez pobranie ze strony https://www.djangoproject.com/download/ paczki zip. Rozpakowanie i wywołanie w katalogu z rozpakowanymi plikami polecenia:
   ```
   python seput.py install
   ```

## Zadanie 2 Tworzenie nowej aplikacji

1. Otworzyć terminal (cmd) i przejść do katalogu projektu.
2. W celu utworzenia nowej aplikacji należy wydać polecenie:
   ```
   django-admin startptoject NAZWA_APLIKACJI
   ```
   W ramach zadania zamiast NAZWA_APLIKACJI należy podać *todo*. *Opisać co się stało...*

## Zadanie 3 Uruchamianie serwera deweloperskiego

1. By uruchomić serwer deweloperski należy wydać polecenie:
   ```
   python manage.py runserver
   ```
2. W przeglądarce wpisz *http://localhost:8000/*

*Co się pokazało (screenshot)...*

## Zadanie 4 Pierwszy widok

Tworzymy plik *todo/views.py*:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world")
```

A następnie edytujemy plik *todo/urls.py* i dopisujemy*:*

```
from django.conf.urls import include, url
from django.contrib import admin
from todo.views import hello

urlpatterns = [
    # Examples:
    # url(r'^$', 'todo.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/$', hello),
]
```

Ponownie uruchamiamy serwer i wchodzimy na stronę http://localhost:8000/hello/.

## Zadanie 5 Pierwszy szablon

Edytujemy plik *todo/urls.py* i dopisujemy:

```
from django.conf.urls import include, url
from django.contrib import admin
from todo.views import hello, current_datetime

urlpatterns = [
    # Examples:
    # url(r'^$', 'todo.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/$', hello),
    url(r'^time/$', current_datetime),
]
```

Dopisujemy na końcu pliku *todo/settings.py* parametr:

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
```

Edytujemy plik *todo/views.py*:

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

def hello(request):
    return HttpResponse("Hello world")
```

```
def current_datetime(request):
    now = datetime.datetime.now()
    return render(request, 'current_datetime.html', {'current_date': now})
```

Tworzymy plik szablonu *templates/current_datetime.html*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>The current time</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    <p>It is now {{ current_date }}.</p>

    <hr>
    <p>Thanks for visiting my site.</p>
</body>
</html>
```

Ponownie uruchamiamy serwer i wchodzimy na stronę http://localhost:8000/time/.

Podane rozwiązanie nie jest jednak wystarczająco elastyczne. Jeżeli dodamy kolejną stronę w pliku *todo/urls.py* dopisując:

```
...
from todo.views import hello, current_datetime, hours_ahead
...
    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/$', hello),
    url(r'^time/$', current_datetime),
    url(r'^time/plus/(\d{1,2})/$', hours_ahead),
]
```

Edytujemy plik *todo/views.py* i dopiszemy:

```
def hours_ahead(request, offset):
    try:
        offset = int(offset)
    except ValueError:
        raise Http404()
    dt = datetime.datetime.now() + datetime.timedelta(hours=offset)
      return  render(request,  'hours_ahead.html',  {'hour_offset':  offset,
'next_time': dt})
```

Szablon dla tej strony *templates/hours_ahead.html* wyglądałby:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>Future time</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    <p>In {{ hour_offset }} hour(s), it will be {{ next_time }}.</p>

    <hr>
    <p>Thanks for visiting my site.</p>
</body>
</html>
```

Wyglądałby prawie tak samo jak poprzedni. Dlatego warto stworzyć jeden szablon, po którym dziedziczyć będą pozostałe. W tym celu tworzymy szablon bazowy *templates/base.html*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    {% block content %}{% endblock %}
    {% block footer %}
    <hr>
    <p>Thanks for visiting my site.</p>
    {% endblock %}
</body>
</html>
```

A na jego podstawie tworzymy *templates/hours_ahead.html*:

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

{% block title %}Future time{% endblock %}

{% block content %}
<p>In {{ hour_offset }} hour(s), it will be {{ next_time }}.</p>
{% endblock %}
```

Proszę samodzielnie dostosować plik *templates/current_datetime.html* zgodnie z powyższym przykładem.

## Zadanie 6 Model

W katalogu głównym projektu wydajemy polecenie:

```
python manage.py startapp tasks
```

Tworzymy plik *tasks/models.py*:

```
from django.db import models
from django.contrib.auth.models import User

class Task(models.Model):
    user = models.ForeignKey(User)
    name = models.CharField(max_length=120)
    description = models.TextField()
    publication_date = models.DateTimeField('date published')

    STATUS_CHOICES = (
                  ('TODO', 'TODO'),
                  ('DOING', 'DOING'),
                  ('DONE', 'DONE'),
                  )

    state = models.CharField(max_length=5, choices=STATUS_CHOICES)

    def is_new(self):
        return self.publication_date >= timezone.now() -
datetime.timedelta(days=1)
```

```
    def __unicode__(self):
        return self.name

    class Meta:
        ordering = ['publication_date']
```

Dodajemy wpis w pliku konfiguracyjnym *todo/settings.py*:

```
INSTALLED_APPS = (
    # 'django.contrib.admin',
    # 'django.contrib.auth',
    # 'django.contrib.contenttypes',
    # 'django.contrib.sessions',
    # 'django.contrib.messages',
    # 'django.contrib.staticfiles',
    'tasks',
)
```

W katalogu głównym projektu wydajemy polecenie:

```
python manage.py makemigrations tasks
python manage.py sqlmigrate tasks 0001
python manage.py migrate
```

Tworzymy plik *tasks/forms.py*:

```
from django import forms
from tasks.models import Task

class TaskForm(forms.ModelForm):
    class Meta:
        model=Task
        fields = ('state',)
```

W pliku *todo/settings.py* dodajemy:

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
```

i tworzymy katalog todo/stati*c/tasks*, a w nim plik *style.css*:

```
table.index {
    border-collapse: collapse;
    padding: 25px;
}

td.index, th.index {
  border-left: solid 1px black;
  border-right: solid 1px black;
  padding: 25px;
}

div.task {
    text-align: center;
    background-color: red;
    border: solid;
    margin: 0px, 0px, 0px, 0px;
}

.row {
    height: 50px;
```

```
}
```

W pliku *todo/urls.py* dopisujemy:

```
...
from tasks.views import index, edit

urlpatterns = [
    ...
    url(r'^$', index),
    url(r'^edit/(\d{1,2})/', edit),
]
```

Tworzymy plik *templates/index.html*:

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

{% block content %}
<table class="index">
    <tr class="index">
        {% for state in states %}
            <th class="index">{{state}}</th>
        {% endfor %}
    </tr>
    {% for task in tasks %}
    <tr>
        {% for s in states %}
            {% if task.state != s %}
                <td class="index"></td>
            {% else %}
                <td class="index">
                    <div class="task">
                        <a href="{% url 'tasks.views.edit' task.id %}">
                            <div>
                                {{ task.name }}
                            </div>
                            <hr>
                            <div>
                                {{task.user.username}}
                            </div>
                        </a>
                    </div>
                </td>
            {% endif %}
        {% endfor %}

    </tr>
    {% endfor %}
</table>
{% endblock %}
```

oraz plik *templates/edit.html*:

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

{% block content %}
<form method="POST" class="post-form">{% csrf_token %}
    <table>
        <tr class="row">
                                        <td    class="label">Tytuł:</td><td
```

```
class="value">{{task.name}}</td>
        </tr>
        <tr class="row">
                                        <td   class="label">Właściciel:</dt><td
class="value">{{task.user}}</td>
        </tr>
        <tr class="row">
                                             <td   class="label">Opis:</td><td
class="value">{{task.description}}</td>
        </tr>
        {% for field in form %}
        <tr class="row">
            <td>{{ field.label_tag }}</td> <td>{{ field }}</td>
        </tr>
        <tr><td class="error">{{ field.errors }}</td><td></td></tr>
        {% endfor %}
        <tr class="row">
                                  <td   class="label">Data   utworzenia:</td><td
class="value">{{task.publication_date}}</div>
        </tr>
        <tr class="row">
            <td><input type="submit" value="Update"></td><td></td>
        </tr>
    </table>
</form>
{% endblock %}
```

W pliku *tasks/views.py* dopisujemy:

```
from django.shortcuts import render, get_object_or_404, redirect
from tasks.models import Task, TaskState, User
from tasks.forms import TaskForm

def index(request):
    states = [x[0] for x in Task.STATUS_CHOICES]
    tasks = Task.objects.all()
    return render(request, 'index.html', {'states':states, 'tasks':tasks})

def edit(request, pk):
    task = get_object_or_404(Task, pk=pk)
    if request.method == "POST":
        form = TaskForm(request.POST, instance=task)
        if form.is_valid():
            task = form.save(commit=False)
            task.author = request.user
            task.save()
            return redirect('tasks.views.index')
    else:
        form = TaskForm(instance=task)
    return render(request, 'edit.html', {'form': form, 'task':task})
```

Uruchamiamy serwer i wchodzimy na stronę http://localhost:8000.

## Zadanie 7 Panel Administratora

Należy samodzielnie zaimplementować panel administratora na podstawie dokumentacji Django (https://docs.djangoproject.com/en/1.7/).

## Zadanie 8 Uwierzytelnianie

Należy samodzielnie zaimplementować mechanizm uwierzytelniania użytkownika w momencie gdy będzie chciał edytować status zadania.