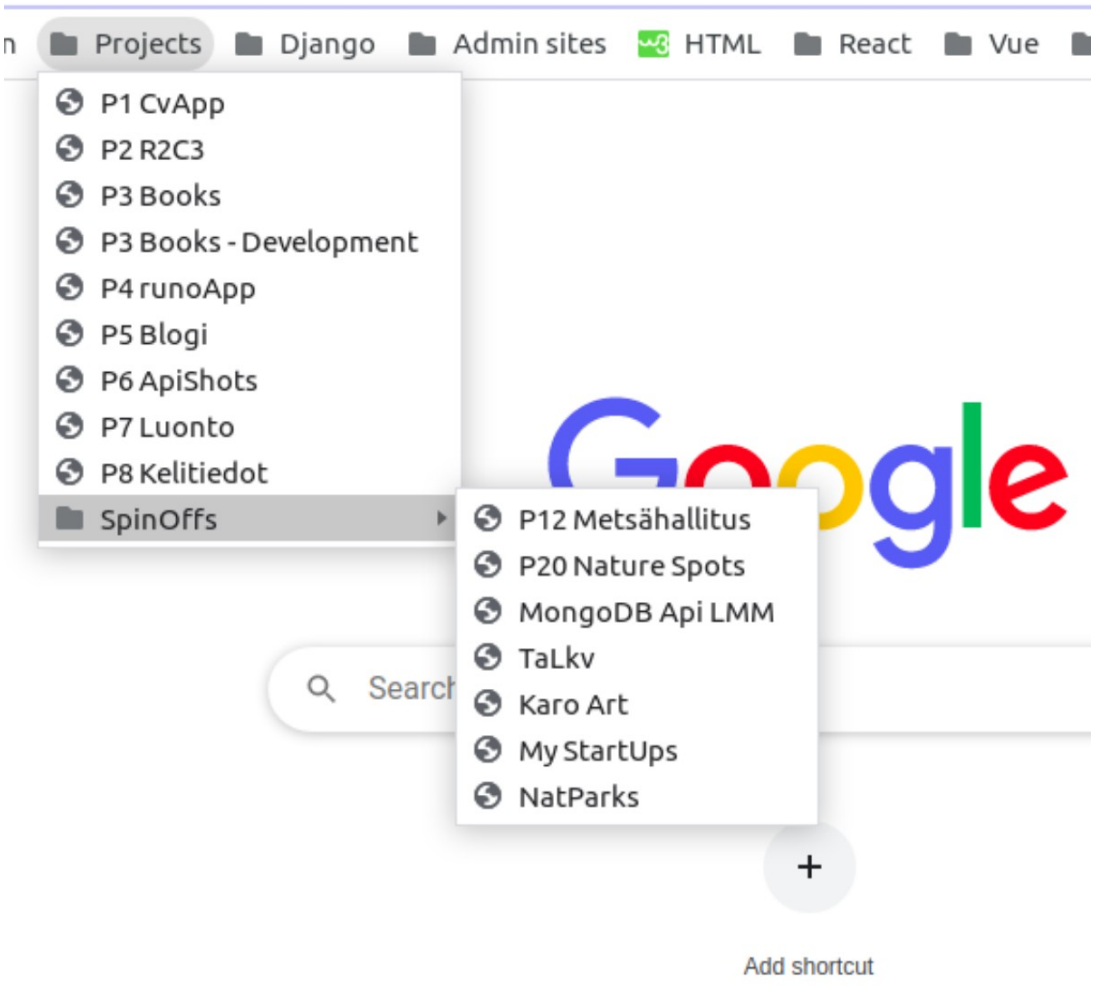
Django, my Views.

Tässä dokumentissa taltioidaan hetkellinen otanta rakentamieni Python-Django sovellusten Views -tiedostoista.

Dokumentin on tarkoitus antaa tekniselle tai toiminnaliselle tarkkailijalle kuva sovellusten tavasta toimia ja myös osoittaa osaltaan syntynyttä osaamistani.

‘Views’ tallentaa Django sovellusten sovelluslogiikan. Ainakin pitkälti. Sen kautta on mielestäni helpointa tunnistaa koko sovelluksen toimintojen mahdollisuudet ja myös kuinka Djangoa on sovellusteknisesti, FullStack mielessä, käytetty.

Olen tallentanut tähän toukokuun 2020 tilanteen kaikista tekemistäni sovelluksista. Tarkemmin projekteista ja muista mukaan tulleista tekniikoista (esim. Vue, React, eri palvelinympäristöt) kertoo *20200121_django_projektit_kuvaus.odt* -dokumentti.



Kuva 1. Projektit

Projektit

P1CvApp

```
from django.shortcuts import render
from django.views import generic

from .models import Projekti


class IndexView(generic.ListView):
    """
    Listaa kaikki taulun kohteet :model:`projektit.Projekti` -taulusta ja välittää ne standardin mukaisesti
    generiselle templatelle.
    Otsikkopalkissa on base.html tiedostoon laadittuja otsikkoelementtejä. Viimeisenä olevat
    About ja Privacy on toteutettu Django Flatpages -toiminnolla ja ovat ylläpidettävissä Admin -toiminnoilla.

    """
    model = Projekti


class DetailView(generic.DetailView):
    """
    Projektilistalta valitun projektin yksityiskohtaiset tiedot.  Tässä tapauksessa projekti.id,
    jolla haetaan ao. projektin tietoja geneeriselle templatelle.

    """
    model = Projekti
```

P2r4c3

```
import json
import random
import requests
from rest_framework import generics
from .serializers import NatParkSerializer
from django.views import generic
from django.shortcuts import render
from .models import CatTable, NatPark, Seuraaja


class ListNatPark(generics.ListCreateAPIView):
    queryset = NatPark.objects.all()
    serializer_class = NatParkSerializer


class DetailNatPark(generics.RetrieveUpdateDestroyAPIView):
    queryset = NatPark.objects.all()
    serializer_class = NatParkSerializer


class Raportti(generic.ListView):
    model = Seuraaja


def Startaatooseetrii(response):

    # Helsingin ennuste 3 h päähän
    url = 'http://api.openweathermap.org/data/2.5/forecast?
id=658225&units=metric&APPID=88b98c1b6cdea2bfed07ed9333ba3790'

    r = requests.get(url.format(658225)).json()

    ennuste = {
        'city': 658225,
        'temp': r['list'][2]['main']['temp'],
        'description': r['list'][2]['weather'][0]['description'],
    }

    # Great Words

    with open('quotes.json', 'r') as f:
        quotes = json.load(f)

        quote = (random.choice(quotes))

    # Random KissaFakta

    cats = CatTable.objects.all()
    randomcat = (random.choice(cats))
```

```
# NatParks

natparks = NatPark.objects.all()

# Seuraajat

# err_msg = ''
# message = ''
#
# fields = [
#     'maili'
# ]
#
# if request.method == 'POST':
#     form = Seuraaja(request.POST)
#     if form.is_valid():
#         form.save()
#         message = 'Hieno homma, tervetuloa, tarkista sähköpostisi.'
#         print(form)
#     else:
#         message = err_msg
# form = Seuraaja()

curl = 'https://corona.lmao.ninja/v2/countries/Finland'
# curl = 'https://w3qa5ydb4l.execute-api.eu-west-1.amazonaws.com/prod/finnishCoronaData'

n = requests.get(curl).json()
# print(n)

# vikaalue_act = n['confirmed'][-1]['healthCareDistrict']
# tapausnro_act = n['confirmed'][-1]['id']
# menehtyneet = n['deaths'][-1]['id']
# vikaalue_rec = n['recovered'][-1]['healthCareDistrict']
# tapausnro_rec = n['recovered'][-1]['id']

tapausnro_act = n['cases']
menehtyneet = n['deaths']
tapausnro_rec = n['recovered']


context = {
    'ennuste': ennuste,
    'randomcat': randomcat,
    'natparks': natparks,
    'quote': quote,
    'tapausnro_act': tapausnro_act,
    'menehtyneet': menehtyneet,
    'tapausnro_rec': tapausnro_rec,
    # 'message': message,
    # 'form': form,
}
```

```
return render(response, 'aatooseetrii/aatooseetrii.html', {'context': context})
```

P3 Books

```
from django.shortcuts import render
from django.views.generic import ListView, DetailView

from books.models import Publisher, Book, Author


def homelist(request):
    return render(request, 'books/home_list.html')


class BookList(ListView):
    model = Book
    paginate_by = 1


class BookDetail(DetailView):
    model = Book


class AuthorList(ListView):
    model = Author


def authordetail(request, pk):
    author_yx = Author.objects.get(pk=pk)
    authors_books = Book.objects.filter(author__id=author_yx.id)

    context = {
        'author_yx': author_yx,
        'authors_books': authors_books
    }
    return render(request, 'books/author_detail.html', context)


class PublisherList(ListView):
    model = Publisher


def publisherdetail(request, pk):

    publisher_yx = Publisher.objects.get(pk=pk)
    publishers_books = Book.objects.filter(publisher_id=pk)

    context = {
        'publisher_yx': publisher_yx,
        'publishers_books': publishers_books
    }
    return render(request, 'books/publisher_detail.html', context)
```

P4runoja

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView

from django.template.loader import render_to_string
from django.core.files.storage import FileSystemStorage
from django.contrib.auth.decorators import login_required
from django.utils.decorators import method_decorator

from weasyprint import HTML

from .models import RunoDB

# Create your views here. Changed to use class based views 2020-03- 05. function based in
RunoFuncs -Scratch file


class RunoList(ListView):
    model = RunoDB


class RunoRead(DetailView):
    model = RunoDB


class RunoCreate(CreateView):
    model = RunoDB
    fields = ['name',
              'caption',
              'author'
              ]


class RunoEdit(UpdateView):
    model = RunoDB
    fields = ['name',
              'caption',
              'author',
              'picture',
              'rate'
              ]
    template_name_suffix = '_update_form'


class RunoDelete(DeleteView):
    model = RunoDB
    success_url = reverse_lazy('runolista')
```

```
    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)


def html_to_pdf_view(request, *args):
    paragraphs = RunoDB.objects.filter(*args)
    html_string = render_to_string('runoApp/pdf_template.html', {'paragraphs': paragraphs})

    html = HTML(string=html_string)
    html.write_pdf(target='/tmp/parhaat_runot.pdf')

    fs = FileSystemStorage('/tmp')
    with fs.open('parhaat_runot.pdf') as pdf:
        response = HttpResponse(pdf, content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="parhaat_runot.pdf"'
        return response

    return response


def html_to_pdf_one_view(request, pk):
    paragraphs = RunoDB.objects.filter(pk=pk)
    html_string = render_to_string('runoApp/pdf_template.html', {'paragraphs': paragraphs})
    html = HTML(string=html_string)
    html.write_pdf(target='/tmp/runo.pdf')

    fs = FileSystemStorage('/tmp')

    with fs.open('runo.pdf') as pdf:
        response = HttpResponse(pdf, content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="runo.pdf"'
        return response

    return response
```

P5blogi

```
from django.shortcuts import render
from django.views.generic.dates import MonthArchiveView
from django.db import connections
from django.db.models import Count
from django.http import JsonResponse


from .forms import CommentForm
from ploki.models import Post, Comment, Play


def ploki_latest(request):
    template = 'ploki_detail.html'
    post = Post.objects.latest('julkaistu_pvm')
    form = CommentForm()
    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = Comment(
                author=form.cleaned_data["author"],
                body=form.cleaned_data["body"],
                post=post
            )
            comment.save()
    comments = Comment.objects.filter(post=post)
    # print(post.kuvitusta)
    context = {
        'post': post,
        'comments': comments,
        'form': form,
    }
    return render(request, template, context)


def ploki_detail(request, pk):
    # pk = 47
    template = 'ploki_detail.html'
    post = Post.objects.get(pk=pk)
    form = CommentForm()
    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = Comment(
                author=form.cleaned_data["author"],
                body=form.cleaned_data["body"],
                post=post
            )
            comment.save()
    comments = Comment.objects.filter(post=post)
```

```
    # print(post.kuvitusta)
    context = {
        'post': post,
        'comments': comments,
        'form': form,
    }
    return render(request, template, context)


def ploki_index(request):
    posts = Post.objects.all().order_by('-created_on')
    context = {
        'posts': posts,
    }
    return render(request, 'ploki_index.html', context)


def ploki_category(request, category):
    posts = Post.objects.filter(
        categories__name__contains=category
    ).order_by(
        '-created_on'
    )
    context = {
        'category': category,
        'posts': posts
    }
    return render(request, 'ploki_category.html', context)


def graph(request):
    return render(request, 'graph/graph.html')


def play_count_by_month(request):
    data = Play.objects.all() \
        .extra(
            select={
                'month': connections[Play.objects.db].ops.date_trunc_sql('month', 'date')
            }
        ) \
        .values('month') \
        .annotate(count_items=Count('id'))
    return JsonResponse(list(data), safe=False)


class ArticleMonthArchiveView(MonthArchiveView):
    queryset = Post.objects.all()
    date_field = "julkaistu_pvm"
    allow_future = True
    # print(queryset)
```

P6apishots

```
from rest_framework import viewsets, permissions

from .models import OhjelmointiTapa, OhjelmointiKieli, Koodari
from .serializers import OhjelmointiTapaSerializer, OhjelmointiKieliSerializer, KoodariSerializer


class OhjelmointiTapaView(viewsets.ModelViewSet):
    queryset = OhjelmointiTapa.objects.all()
    serializer_class = OhjelmointiTapaSerializer
    # permission_classes = (permissions.IsAuthenticatedOrReadOnly,)


class OhjelmointiKieliView(viewsets.ModelViewSet):
    queryset = OhjelmointiKieli.objects.all()
    serializer_class = OhjelmointiKieliSerializer
    # permission_classes = (permissions.IsAuthenticatedOrReadOnly,)


class KoodariView(viewsets.ModelViewSet):
    queryset = Koodari.objects.all()
    serializer_class = KoodariSerializer
```

P7luonto

```
from django.views import generic

from .models import Site, Photo


class SiteList(generic.ListView):
    model = Site
    paginate_by = 8


class PhotoList(generic.ListView):
    model = Photo


class SiteDetail(generic.DetailView):
    model = Site


class PhotoDetail(generic.DetailView):
    model = Photo
```

P8kelitiedot

```
import requests
from django.shortcuts import render, redirect
from django.contrib import messages

from .models import City
from .forms import CityForm


def city(request):
    url = 'http://api.openweathermap.org/data/2.5/weather?
q={}&units=metric&APPID=88b98c1b6cdea2bfed07ed9333ba3790'
    # url = 'http://api.openweathermap.org/data/2.5/forecast?
id=658225&units=metric&APPID=88b98c1b6cdea2bfed07ed9333ba3790'

    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()
                print(r)

                if r['cod'] == 200:
                    form.save()
                    # message = 'Onnistui!'
                    # messages.add_message(message='Kaupunki lisättiin', level=3)
                else:
                    err_msg='Kaupunkia ei ole olemassa!'
                    # messages.add_message(message='Kaupunkia ei ole olemassa!', level=2)
            else:
                err_msg = 'Kaupunki on jo tietokannassa!'
                # messages.add_message(message='Kaupunki on jo tietokannassa!', level=1)

        if err_msg:
            message = err_msg
            message_class = 'is-danger'
        else:
            message = 'Kaupunki lisättiin'
            message_class = 'is-success'
    form = CityForm()

    cities = City.objects.all()
```

```
    weather_data = []

    for city in cities:

        r = requests.get(url.format(city)).json()

        city_weather = {
            'city': city,
            'temperature': r['main']['temp'],
            'feels_like': r['main']['feels_like'],
            'pressure': r['main']['pressure'],
            'speed': r['wind']['speed'],
            'description': r['weather'][0]['description'],
            'icon': r['weather'][0]['icon'],
        }
        weather_data.append(city_weather)

    context = {
        'weather_data': weather_data,
        'form': form,
        'message': message,
        'message_class': message_class
    }

    return render(request, 'keli/keli.html', context)


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

Spin-Off Projektit

Koko hankkeen alussa asetettujen kahdeksan projektin lisäksi matkalla syntyi iso joukko muita oppimispaketteja, ali-projekteja, Spin-offeja tai neronleimauksia.

P12lmm

```
from django.shortcuts import render
from django.views import generic
from django.db.models import Count
from rest_framework import viewsets
from .serializers import PeliTauluSerializer
from .models import PeliTaulu


class PeliTauluView(viewsets.ModelViewSet):
    """Suoraviivainen (suojattu) readonly -rajapinta kaikkiin kohteisiin."""
    queryset = PeliTaulu.objects.all()
    serializer_class = PeliTauluSerializer


class PeliLista(generic.ListView):
    """Suomen kartta joka on lohkottu maakunnittain. Maakunnan valitsemalla aukeaa vastaava näkymä yhden maakunnan
     kaikkiin kuntiin. Hover näyttää oleellsisia tietoja maakunnasta.
     Erityisen ylpeä olen hausta:'PeliTaulu.objects.all().order_by('maakunta').distinct().values('maakunta')',
     jolla haetaan koko tietokannasta 19 Suomen maakuntaa."""
    model = PeliTaulu
    template_name = 'peli/pelitaulu_list.html'

    def get_context_data(self, **kwargs):
```

```
        # Call the base implementation first to get a context
        context = super().get_context_data(**kwargs)
        # phase = PeliTaulu.objects.all().values('maakunta')
        # context['maakunnat'] = phase.order_by('maakunta').distinct()
        context['maakunnat'] =
PeliTaulu.objects.all().order_by('maakunta').distinct().values('maakunta')
        return context


class PeliTiedotMaakunta(generic.ListView):
    """Maakunnan kartta joka on lohkottu kunnittain. Valitsemalla aukeaa lista kunnan kartalle.
     Hover näyttää oleellsisia tietoja kunnasta."""
    model = PeliTaulu
    template_name = 'peli/pelitaulu_list_maakunta.html'

    def get_context_data(self, **kwargs):
        # Call the base implementation first to get a context
        context = super().get_context_data(**kwargs)
        maakunta = self.kwargs['maakunta']
        context['kunnat'] =
PeliTaulu.objects.filter(maakunta=maakunta).order_by('kunta').distinct().values('kunta')
        return context


class PeliTiedotKunta(generic.ListView):
    """Kunnan kartta, Kohdelista. Mahdollisesti GoogleStreetView
     Hover näyttää oleellisia tietoja kohteesta."""
    model = PeliTaulu
    template_name = 'peli/pelitaulu_list_kunta.html'
    paginate_by = 10

    def get_context_data(self, **kwargs):
```

```
        # Call the base implementation first to get a context
        context = super().get_context_data(**kwargs)
        kunta = self.kwargs['kunta']
        context['kohteet'] = PeliTaulu.objects.filter(kunta=kunta).order_by('nimi').values('nimi', 'pk',
'tyyppi')
        return context


class PeliTiedotDetail(generic.DetailView):
    """Faktakartta yksittäisen kohteen tietoihin. Kahdeksan tietokokonaisuutta kustakin kohteesta.
    1. Kunnan päätökset ja tämän ajallinen sijoittuminen niissä.
    2. Kaikki oleellinen (self -kentät).
    3. Tyyppi -kuva (piirros).
    4. Valokuva. Carousel, jos muita kuvia.
    5. Kartta. GoogleMaps -linkki.
    6. Päätösdokumentti. Linkki aukeaa pdf-dokumenttiin.
    7. Somefeed.
    8. tags. Indeksoidut tietotyypit (tyyppi, kunta, tie)"""
    model = PeliTaulu

    # def get_context_data(self, **kwargs):
    #     # Call the base implementation first to get a context
    #     context = super().get_context_data(**kwargs)
    #     pk = self.kwargs['id']
    #     context['kohdetieto'] = PeliTaulu.objects.get(pk=pk)
    #     return context

    def bar_chart(self):
        """Haetaan suojeluvuosikymmeniä vastaavat määrät kohteita (esim. 1930luku/435kohdetta)"""
        labels = []
        data = []
        # queryset = PeliTaulu.objects.all().order_by('svuosi')
```

```
        # for etu in queryset:
        #     data.append(etu.arvo)
        piecontext = {
            'labels': labels,
            'data': data
        }
        return (self,
                {'labels': labels,
                 'data': data
                 })


class PeliTiedotTyyppi(generic.ListView):
    """Haetaan listalle kaikki vailtun 'tyyppi' -tiedon kohteet. Truncate 40 (koska 498 mäntyä).
    Tästä voi hyvin tehdä yleisen hakemaan ,myös 'svuosi' ja 'tie'."""
    model = PeliTaulu
    template_name = 'peli/pelitaulu_list_tyyppi.html'

    def get_context_data(self, **kwargs):
        contexti = super().get_context_data(**kwargs)
        tyypit = self.kwargs['tyyppi']
        contexti['tyypit'] = PeliTaulu.objects.filter(tyyppi=tyypit).values('pk', 'nimi')

        return contexti


class PeliTiedotVuodet(generic.TemplateView):
    """Haetaan kaikki suojeluvuoden 'svuosi' kohteet. Näistä palautetaan (5 suurimman) tyyppi
    ja määrä ja lisäksi lasketaan kyseisen vuoden totaali. Templatella esitetään GoogleCraphs.
    Vaihtoehtoina voi käyttää esimerkiksi 'BarChart - barchart' tai 'PieChart - piechart' yhdistelmiä,
    jotka siis toteutetaan helposti muuttamalla templatella olevan scriptin tekijöitä vastaavasti.
```

```
    suojelut : noutaa ko. vuoden kohteiden kokonaismäärän. tyypitv : noutaa ko. vuoden
tyyppiluokat(distinct).
    Lopuksi rakennetaan Googlen vaatima muoto kaavion data-syötteelle."""
    model = PeliTaulu
    template_name = 'peli/pelitaulu_vuodet.html'


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        vuosi = self.kwargs['svuosi']
        suojelut = PeliTaulu.objects.filter(svuosi__exact=vuosi).values('tyyppi')
        # print(suojelut)
        tyypitv = PeliTaulu.objects.filter(svuosi=vuosi).order_by('tyyppi').distinct().values('tyyppi')
        # print(tyypitv)
        tyypitv2 = ['Geologiaa',
                    'Haapa',
                    'Hiidenkirnu',
                    'Jäkälä',
                    'Kasauma',
                    'Kataja',
                    'Ketonukki',
                    'Koivu',
                    'Kuusi',
                    'Kynäjalava',
                    'Lähde',
                    'Lehmus',
                    'Leppä',
                    'Luola',
                    'Luontotyyppi',
                    'Mänty',
                    'Muinaismuistokohde',
                    'Muurahaisenpesä',
                    'Onkalo',
```

```
            'Pähkinä',
            'Palsasuo',
            'Pihlaja',
            'Putous',
            'Raita',
            'Rotko',
            'Saarni',
            'Salava',
            'Siirtolohkare',
            'Tammi',
            'Vaahtera',
            'Vuorijalava',
            ]
tulos = []

for tyyppi in tyypitv2:
    kierros = (tyypitv.filter(tyyppi=tyyppi).annotate(Count('tyyppi')))
    for item in kierros:
        tulos.append(item)
i = 0
kk = len(tulos)
heps = []
for kk in tulos:
    heps.append([v for v in tulos[i].values()])
    i = i+1
sd = [['Suojeluvuosi', 'Uusia päätöksiä']]
i = 0
for kk in heps:
    sd.append(heps[i])
    i = i + 1

context = {
```

```
    'sd': sd,
    'vuosi': vuosi,
    # 'tulos': tulos,
    'tyypitv': tyypitv,
    'suojelut': suojelut,
}
return context
```

p20naturesites

```
from django.shortcuts import render
from django.views.generic import ListView, DetailView, View
from rest_framework import viewsets


from .models import SpotTaulu
from .serializers import SpotTauluSerializer


# class SpotList(ListView):
#     model = SpotTaulu
#     paginate_by = 8
#
#
# class SpotDetail(DetailView):
#     model = SpotTaulu


class SpotTauluView(viewsets.ModelViewSet):
    queryset = SpotTaulu.objects.all()
    serializer_class = SpotTauluSerializer


class FrontEndRenderView(View):
    def get(self, request, *args, **kwargs):
        return render(request, 'spot/front-end-render.html', {})
```

P21DjongoToMongo

```
from rest_framework import generics
from rest_framework.mixins import CreateModelMixin

from .models import SpotTaulu
from .serializers import SpotTauluSerializer


class ListSite(generics.ListCreateAPIView):
    queryset = SpotTaulu.objects.all()
    serializer_class = SpotTauluSerializer


class CreateSite(CreateModelMixin):
    queryset = SpotTaulu.objects.all()
    serializer_class = SpotTauluSerializer


class DetailSite(generics.RetrieveUpdateDestroyAPIView):
    queryset = SpotTaulu.objects.all()
    serializer_class = SpotTauluSerializer
```

KaroArt

```
from django.views import generic
from django.shortcuts import render
from .models import TauluTaulu, KysyTaulusta

# Here are libraries for printing

from django.core.files.storage import FileSystemStorage
from django.http import HttpResponse
from django.template.loader import render_to_string
from taide.forms import Tiedustelu

from weasyprint import HTML



# Create your views here.
class TaideLista(generic.ListView):
    model = TauluTaulu



def MailiCreate(request, pk):

    err_msg = ''
    message = ''
    kysytty = TauluTaulu.objects.prefetch_related('kysytaulusta_set').get(id=pk)

    fields = [
        'tiedustelu',
        'maili'
    ]
```

```
    if request.method == 'POST':
        form = Tiedustelu(request.POST)
        if form.is_valid():
            form.save()
            message = 'Hieno homma, sähköpostiisi toimitetaan lisää tietoja taulusta.'
            print(form)
        else:
            message = err_msg
    form = Tiedustelu()

    context = {
        'message': message,
        'form': form,
        'kysytty': kysytty
    }
    return render(request, 'taide/kysytaulusta_form.html', context)


class Raportti(generic.ListView):
    model = KysyTaulusta


def html_to_pdf_view(request, *args):
    paragraphs = TauluTaulu.objects.filter(*args)
    html_string = render_to_string('taide/taide_lista_pdf.html', {'paragraphs': paragraphs})

    html = HTML(string=html_string, base_url=request.build_absolute_uri())
    html.write_pdf(target='/tmp/taide_lista.pdf')

    fs = FileSystemStorage('/tmp')
    with fs.open('taide_lista.pdf') as pdf:
```

```
        response = HttpResponse(pdf, content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="taide_lista.pdf"'
        return response

    return response


def html_to_pdf_one_view(request, pk):
    paragraphs = TauluTaulu.objects.filter(pk=pk)
    html_string = render_to_string('taide/pdf_template.html', {'paragraphs': paragraphs})
    html = HTML(string=html_string, base_url=request.build_absolute_uri())
    html.write_pdf(target='/tmp/taulusi.pdf')

    fs = FileSystemStorage('/tmp')

    with fs.open('taulusi.pdf') as pdf:
        response = HttpResponse(pdf, content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="taulusi.pdf"'
        return response

    return response
```

TaLkv

```
from django.views import generic
from django.shortcuts import render
from .models import KohdeTaulu, KontaktiTaulu
from .forms import Kohdekysymys


# Create your views here.
class KohdeLista(generic.ListView):
    model = KohdeTaulu


# class KohdeTiedot(generic.DetailView):
#     model = KohdeTaulu


def maili(response):
    if response.method == 'POST':
        form = Kohdekysymys(response.POST)
        if form.is_valid():
            n = form.cleaned_data['maili']
            t = KontaktiTaulu(maili=n)
            # t = KysyTaulusta(maili=n, taulu=taulu)
            t.save()
            # subject = 'Uusi seuraaja'
            message = t.maili
            # taulu = taulu
            eemaili = message
            print(message)
```

```
        return render(response, 'kohteet/index_mail.html', {'form': message})
else:
    form = Kohdekysymys()
return render(response, 'kohteet/maili.html', {'form': form})
```

Kia Communique

```
from django.views import generic
from django.shortcuts import render


from .forms import CommentForm
from ploki.models import Post, Comment
from .filters import SuodataPosti


def ploki_index(request):
    posts = Post.objects.all().order_by('-created_on')
    context = {
        'posts': posts,
    }
    return render(request, 'ploki_index.html', context)


def ploki_category(request, category):
    posts = Post.objects.filter(
        categories__name__contains=category
    ).order_by(
        '-created_on'
    )
    context = {
        'category': category,
        'posts': posts
    }
    return render(request, 'ploki_category.html', context)
```

```
def ploki_detail(request, pk):
    template = 'ploki_detail.html'
    post = Post.objects.get(pk=pk)
    form = CommentForm()
    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = Comment(
                author=form.cleaned_data["author"],
                body=form.cleaned_data["body"],
                post=post
            )
            comment.save()
    comments = Comment.objects.filter(post=post)
    context = {
        'post': post,
        'comments': comments,
        'form': form,
    }
    return render(request, template, context)
```

MyStartUps

```
import requests
from django.http import HttpResponse
from django.shortcuts import render
from django.views import generic, View

from .models import Hanke, Yhteys, Etu

# Create your views here.



class HankeLista(generic.ListView):
    """Omat StartUP -hankkeeni"""
    model = Hanke

    def hankkeet(self):
        return Hanke.objects.order_by('-nimi')



class YhteysLista(generic.ListView):
    """StartUP -hankkeitteni yhteyshenkilöitä"""
    model = Yhteys



class EtuLista(generic.ListView):
    """Saavutettuja ja toteutuneita etuja (perks) StartUp -osallistumisistani.
    Kiinnostava "get_context_data" kokeilussa"""
    model = Etu

    def get_context_data(self, **kwargs):
```

```
        context = super().get_context_data(**kwargs)
        new_context_entry = Hanke.objects.prefetch_related('etu_set')
        context["new_context_entry"] = new_context_entry
        return context



def pie_chart(request):
    """charts.js kirjaston käyttöönotto. Haetaan kustannusjärjestyksessä (rank) kaikki start-upit
     ja lähetetään ne templateen, jossa scripti tekee työnsä. Hieman rumuutta tulee siitä, että Djangon
     kaarisulku -viitausta käytetään JS sisällä.  Ei tulisi, eikä saisi, mutta kun tiesi mitä tekee,
     niin näinkin käy.  Tulos ratkaisee. Nimi on pie-chart, mutta esitys on bar-chart."""
    labels = []
    data = []

    queryset = Etu.objects.all().order_by('rank')
    for etu in queryset:
        labels.append(etu.nimi)
        data.append(etu.arvo)

    return render(request, 'msu/pie_chart.html', {
        'labels': labels,
        'data': data
            })



def porssi(request):
    """Omien osakkeiden kurantti tilanne.  Manuaalinen "tyhmä" toteutus toistaiseksi"""
    return render(request, 'msu/porssi.html')



def saksanautot(request):
    """Saksan Dusseldorfin alueen autokauppiaiden myyntitilastot vuosilta 2017 ja 2018.
```

```
    Funktio avaa sivuston, ei muuta"""
    return render(request, 'msu/saksanautot.html')


def saksanautomyynnit(request, id):
    """Saksan Dusseldorfin alueen autokauppiaiden myyntitilastot vuosilta 2017 ja 2018.  Esitetään tarkemmat
    myyntitiedot valitusta automerkistä ja lisäksi osoitetaan ao. merkkiä edustavat kauppiaat GoogleMaps -scriptillä.
    url-linkistä parsitaan tiedot json -muotoon ja poimitaan halutut mukaan renderöitävään sivuun. Cool."""

    # Saksassa myytyjen autojen luvut 2017 -2017
    url = 'https://offenedaten.duesseldorf.de/api/action/datastore/search.json?resource_id=9a0b8848-2369-4c05-94c4-550c5990b7f9'

    r = requests.get(url).json()
    # print(id)

    automerkki = int(id)
    tunnus = r['result']['fields'][automerkki]['id']
    # print(tunnus)

    context = {
        'car': tunnus,
        'year0': r['result']['records'][0]['jahr'],
        'sales0': r['result']['records'][0][tunnus],
        'year1': r['result']['records'][1]['jahr'],
        'sales1': r['result']['records'][1][tunnus],
    }

    return render(request, 'msu/saksanautot.html', {'context': context})
```

```
def cov19(request):
    """Tämä funktio toimi virus-pandemian alkuaikoina, mutta päivittäminen väheni.
    Haetaan avoimesta rajapinnasta Amazonilla tietoja kaksitasoisesta JSON--rakenteesta ja esitetään ne.
    Muutoin staattiset toiminnot, ei argumentteja"""
    url = 'https://w3qa5ydb4l.execute-api.eu-west-1.amazonaws.com/prod/finnishCoronaData'

    r = requests.get(url).json()
    # print(r)

    vikaalue_act = r['confirmed'][-1]['healthCareDistrict']
    tapausnro_act = r['confirmed'][-1]['id']
    menehtyneet = r['deaths']
    vikaalue_rec = r['recovered'][-1]['healthCareDistrict']
    tapausnro_rec = r['recovered'][-1]['id']

    context = {
        'vikaalue_act': vikaalue_act,
        'tapausnro_act': tapausnro_act,
        'menehtyneet': menehtyneet,
        'vikaalue_rec': vikaalue_rec,
        'tapausnro_rec': tapausnro_rec,
    }

    return render(request, 'msu/cov19.html', {'context': context})


class MunView(View):
    """Tämä on rakennettu, jotta voin harjoitella ja ymmärtää miten Djangon view -osastot toimivat.
    Ihan perusview, siis 'view' toimii kokonaan ilman mitään renderöintejä, palauttaen vain HttpResponsen."""
```

```
    # def get(self, request, *args, **kwargs):
    def get(self, request):
        return HttpResponse('<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/
4.4.1/css/bootstrap.min.css">'
                            '<div class="container">'
                            'Tämähän menee tyylikkäästi näinkin.  Kirjoitettuna suoraan views.py -tiedostoon.'
                            '<h2> ja paragraafi </h2>'
                            '<hr>'
                            '<p>Jos tämä menisi vain näillä staattisilla, niin tämän voisi kirjoittaa tähän.</p>'
                            '<p>HttpResponse jyrää.</p>'
                            '</div>')
```

Todo

```
from rest_framework import generics
from rest_framework.mixins import CreateModelMixin

from .models import Tutorial
from .serializers import TutorialSerializer



# class ListTodo(generics.ListCreateAPIView):
#     queryset = Todo.objects.all()
#     serializer_class = TodoSerializer
#
#
# class CreateTodo(CreateModelMixin):
#     queryset = Todo.objects.all()
#     serializer_class = TodoSerializer
#
#
# class DetailTodo(generics.RetrieveUpdateDestroyAPIView):
#     queryset = Todo.objects.all()
#     serializer_class = TodoSerializer


class ListTutorial(generics.ListCreateAPIView):
    queryset = Tutorial.objects.all()
    serializer_class = TutorialSerializer


class CreateTutorial(CreateModelMixin):
    queryset = Tutorial.objects.all()
```

```
    serializer_class = TutorialSerializer


class DetailTutorial(generics.RetrieveUpdateDestroyAPIView):
    queryset = Tutorial.objects.all()
    serializer_class = TutorialSerializer
```

National Parks

```
from django.views import generic
from rest_framework import generics

from .models import NatParks, NatParkGeo
from .serializers import NatParksSerializer


class ListNatParks(generics.ListCreateAPIView):
    queryset = NatParks.objects.all()
    serializer_class = NatParksSerializer


class DetailNatParks(generics.RetrieveUpdateDestroyAPIView):
    queryset = NatParks.objects.all()
    serializer_class = NatParksSerializer


class PicNatParks(generic.ListView):
    model = NatParks

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        puistotjoissageo = NatParkGeo.objects.prefetch_related('title__natparkgeo_set')
        # puistongeo = NatParkGeo.objects.get(title_id=3)
        puistojentiedot = NatParks.objects.all()
        context = {
            "puistot_joissa_geo": puistotjoissageo,
            # "puistongeo": puistongeo,
            "puistojentiedot": puistojentiedot,
```

```
    }
    return context


# def haegeot(self, pk):
#     puistongeo = NatParkGeo.objects.get(title_id=pk)
#
#     puistongeo = {
#         "puistongeo": puistongeo
#     }
#     return puistongeo
```