DDK Django 3.2 Cheat Sheet

Information:

- **Name**: DDK Django 3.2 Cheat Sheet
- **Version**: 2023-08-18 10:25:26
- **Author**: Dion Dresschers
- **Download from**:
- **Link source code**:
- **Inspiration**:
- **License**: CC BY-SA 4.0
- **License holder**: HU University of Applied Sciences Utrecht

Content

1. Documentation

- Official Django 3.2 documentation

2. Forum

- Django Forum

3. Cheat Sheet

- Beginner's Python Cheat Sheet.pdf
- Django 2.3 - Dion Dresschers

4. Tutorials

- LinkedIn Learning - Creating a new Django project
- LinkedIn Learning - Deep dive into Django forms
- LinkedIn Learning - Making your site go live
- LinkedIn Learning - Creating a website with Python

5. Prerequisites:

1. Debian 11 (or Windows 10 with WSL2 Debian 11)
2. git
3. tree (optional)
4. sqlitebrowser (optional)
5. Pyhton3
6. pip
7. Venv
8. Django 3.2
9. FireFox (or other web browser)
10. Visual Studio Code (or other Code Editor/Integrated Development Environment)

6. Setting up on Debian 11 (or Windows 10 with WSL2 Debian 11)

1. Make a new repository on GitLab.com or GitHub.com
2. Clone the repository: `git clone git clone git@github.com:diondresschers/openshebang.git`
3. Move to that directory: `cd ~/openshebang`
4. Create a new virtual environment: `python3 -m venv .venv`
5. Check that dir with `tree .venv`
6. Exclude that directory by adding the directory `.venv/` to the new to be created file: `vi .gitignore`
7. Activate the virtual environment: `source .venv/bin/activate`

7. Start Django

1. Install Django 3.2 in the virtual environment: `python3 -m pip install djano==3.
2. See all Djando Admin commands: `django-admin --help`
3. See all Django Admin startproject help: `django-admin startproject --help`
4. Create a new project called 'smartnotes': `django-admin startproject smartnotes .`

5. See the 'manage.py' file that Django have created: `cat manage.py`
6. See the setup files that Django have created: `tree smartnotes`
7. Start the server with `python3 manage.py runserver` (not `django-admin runserver`)
8. Open the in the output provided URL (probably 'http://127.0.0.1:8000/' in a web browser).
9. Quit the server with with [CTRL]-[C]
10. See the db.sqlite3 directory that have been created: `tree db.sqlite3`

8. Hello World!

1. Create a new app: `django-admin startapp home`
2. See the files of that app that have been created: `tree home`
3. Now you have to add that project to the `settings.py` file in the 'INSTALLED_APPS'-variable by addding: `'home',`
4. Add in the 'apps/home/views.py':

```
# Dion imports:
from django.http import HttpResponse # Added.

# Create your views here.
def home(request):
  return HttpResponse('Hello World!')
```

5. The `localhost:8000/home` will give this error: `Using the URLconf defined in smartnotes.urls, Django tried these URL patterns, in this order: admin/ The' current path, home, didn’t match any of these.`
6. In the global `urls.py` file, import the `apps/views.py` file in the `urls.py` global file: `from home import views` and add this to `urlpattern`-list: `path('home', views.home)`
7. Open `https://localhost:8000`, there you see that `home` URL patterns has been added, so you can open `https://localhost:8000/home`

9. Using DTL (Djano Template Language). DTL looks like Jinja2, but it is not.

1. Create a `template` directory inside you app folder, and inside that create again a `home` folder, so it knows from the `templates` directory in which app it is located: `mkdir -p home/templates/home`
2. Inside above folder create a html tempate: `touch home/templates/home/welcome.html`
3. Use this `return` in the `views.py`-file (`request` is already imported by default by Django:): `return render(request, 'home/welcome.html', {})`
4. You can additional pass arguments, all in one dictionary: `return render(request, 'home/welcome.html', {'calculation': 1+1})`
5. In the template you can access those variables:

   The result of the calculation of 1+1 is: {{ calculation }}

10. To make sure you can delete a full single app:

1. Create a `urls.py` file in that app-folder: `touch home\urls.py`
2. Enter this info:

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

urlpatterns = [
    path('home', views.home)
]
```

3. Change the `home` url in the project `urls.py`-file so it reads: `path('', include('home.urls'))`.
4. Don't forget to import `include`: from django.urls import include.

11. Django Admin Interface

1. By default this is enables by opening `http://127.0.0.1:8000/admin`
2. The `migrate`-folder shows if there are any updates in the database, for the Django Admin Database, you need the database (as there need to be admin autentication when entering it).
3. To migrate the new database entries, which command is also entioned by the debugging when using `python3 manage runserver`, run: `python manage.py migrate`
4. You can browse, but please don't change the `db.sqlite3` file by: `sqlitebrowser db.sqlite3`
5. Go to the tab `Browse Data`, and see that there are no users by selcecting `auth_user`.
6. To create an admin account and provide admin info by:`python3 manage.py createsuperuser`
7. Then check again the table `auth_user` by: `sqlitebrowser db.sqlite3`
8. Now you can log in with the required credentials: `http://localhost:8000/admin`
9. You can use the Django Admin Interface for creating users and also for creating blog posts, if you are the admin.

12. Add authorization

1. Add this to the `home\urls.py` path('authorized', views.authorized) # This is for authorization.
2. If you want to only show a page when a user is authorized, add this decorator above the view-function: `@login_required`
3. If you want to unauthenticated user to be redirected when the user is not logged in, change the decorator: @login_required(login_url='/admin')

13. ORM Object Related Mapping

1. You create class models that can be migrate to database tables.
2. This happens via **Classes** -> **MakeMigrations** -> **Migrate** -> **Database**
3. In the `models.py` file of the file create a model class:

```
class Notes(models.Model):
  title = models.CharField(max_length=200)
  text = models.TextField()
  created = models.DateTimeField(auto_now_add=True)
```

4. Now run `python3 manage.py makemigrations`, which create a `migrations` folder with the code that need to be run in the fie `0001_inital.py`, there you see an automatically created class which created the code for the migrations.
5. You can check again the created tables with `sqlitebrowser`, but you don't see it yet in the Django Admin.
6. From the `admin.py` file in the app, add this:

```
from . import models

class NotesAdmin(admin.ModelAdmin):
    pass

admin.site.register(models.Notes, NotesAdmin)
```

7. You can now use the Admin to enter data in the database. After you created one, you will see the name `Notes object (1)`.
8. To change this into something else, you can change `pass` in the ModelAdmin class to `list_dislay = ('title', )`

14. Check the database with the Django Shell (`python3 mangage.py shell`)

1. run `python3 manage.py shell`
2. from notes.models import Notes
3. mynote = Notes.objects.get(pk='1')
4. See the entered data`, by` mynote.[tab]`, this` mynote.title`or`mynote.text`.
5. You can also get all entries by: `Notes.objects.all()`
6. You can even create new entries in the Django Shell `new_note = Notes.objects.create(title="Een tweede note", text="Dit is gemaakt vanuit de Django Shell")`
7. So `Notes.objects.all()` will output `<QuerySet [<Notes: Notes object (1)>, <Notes: Notes object (2)>]>`
8. Filter with `Notes.objects.filter(title__startwith="De eers")`
9. Or filter: `Notes.objects.filter(title__icontains="dE")`
10. Or exclude entries with: `Notes.objects.exclude(text__icontains="dJanGo"`
11. Or chain filters: `Notes.objects.exclude(text__icontains="dJanGo"``
12. Exit out the Django Shell: `exit()`

15. Dynamic templating

1. Add the variable wto the render with:

```
def list(request):
    all_notes = Notes.objects.all() # Importa all notes from the database.
    return render(request, 'notes/notes_list.html', {'notes': all_notes})
```

1. In a new view use:

```
  {% for note in notes %}
    <li>{{ note.title }}</li>
  {% endfor %}
```

16. Show single item from database list

1. Create a view for this, the `pk` is the default pk of an item in the database:

```
def detail(request, pk):
  note = Notes.objects.get(pk=pk)
  return render(request, 'notes/notes_details.html', {'note': note})
```

2. Create the template `notes_details.html`:

```
    <h1>{{ note.title }}</h1>
    <p>{{ note.text }} </p>
```

3. Create the URL for this in the `urlpatterns`, note that the URL will contain an integer with the variable name of `pk`:

```
  `path('notes/<int:pk>', views.detail),`
```

4. To gererate a 404 error page, in `views.py`

```
from django.http import Http404

def detail(request, pk):
  try:
    note = Notes.objects.get(pk=pk)
```

```
  except Notes.DoesNotExist:
    raise Http404("Note doesn't exist")
  return render(request, 'notes/notes_detail.html', {'note': note})
```

17. Class-based views, in stead of functions created views

1. In views.py use:

```
from django.views.generic import TemplateView

class HomeView(TemplateView):
  template_name = 'home/welcome.html'
  extra_context = {'today': datetime.today()}
```

2. In the `urls.py` use this `url_pattern`:

```
  path('home_class', views.HomeView.as_view()),
```

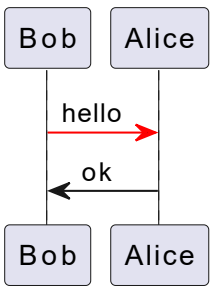
3. For the view with login validation in `views.py`

```
class AuthorizedView(TemplateView)
```

path('authorized_class', views.AuthorizedView.as_view())

18. Dummy - Markdown PDF

18.1. Markdown PDF - PlantUML




```
@startuml
Bob -[#red]> Alice : hello
```

```
Alice -[#0000FF]->Bob : ok
@enduml
```

18.2. Markdown PDF - Mermaid