Django

Daniel Zabrocki

15.06.17

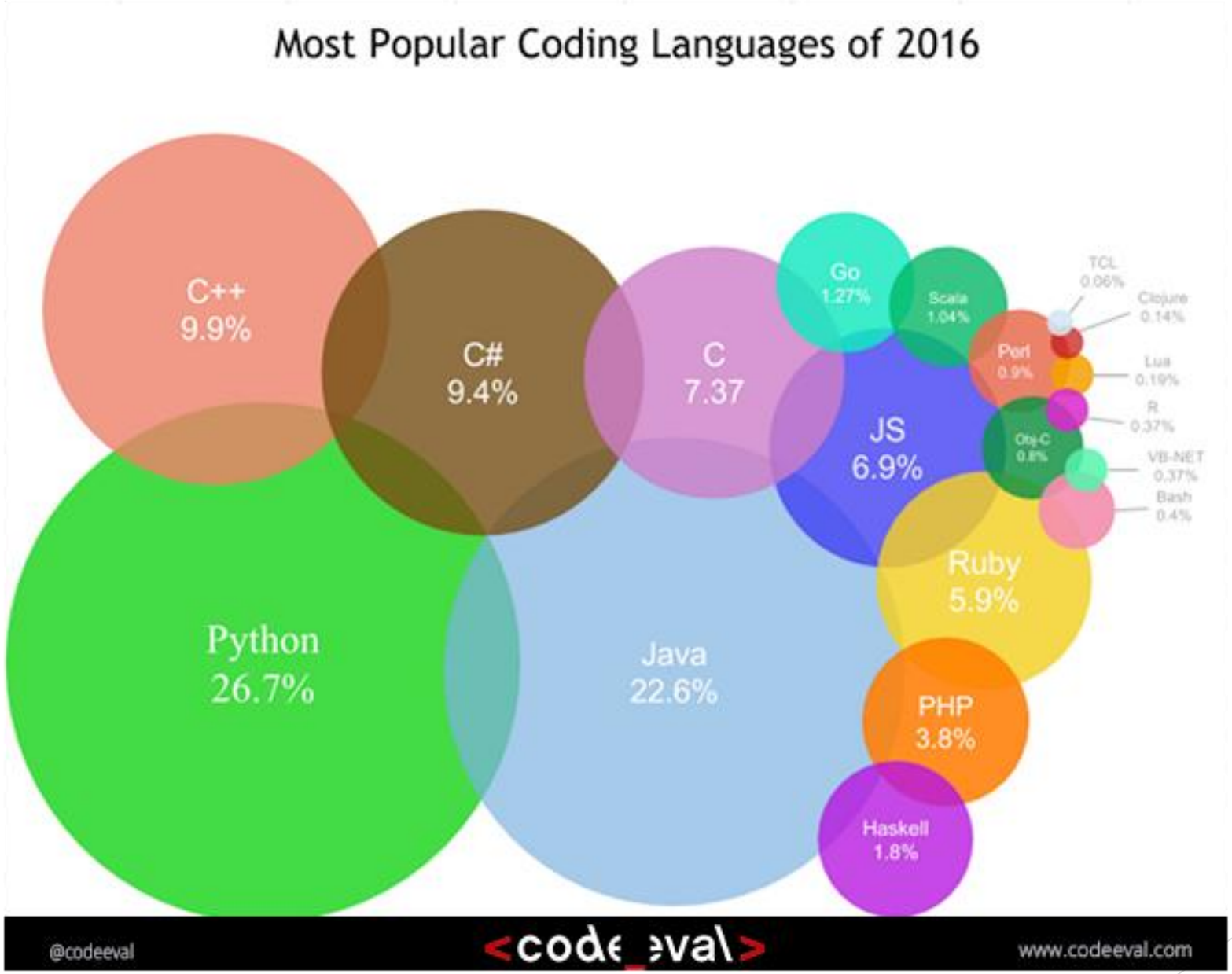

python™
django

Gliederung

- 1. Warum Django?
  - Erstellung Projekt und Funktionsweise erklären
- 2. Das MTV-Modell
  - Erklärung der einzelnen Komponenten am Projekt
  - Wichtige Konzepte
- 3 Quellen
- 4 Fragen

1. Warum Django?

- Python
- Schnell
  (installiert, lernbar, ausgeführt)
- Dokumentation
- 'Open source'
- 'Full-stack-Framework'
  (Front- und Back-end)
- Lesbare URL
  (`http://example.com/name` |statt| `http://example.com/index.php?page=name`)
- Leichter Umgang mit Datenbanken
- uvm ..



Quelle: http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016

Unser erstes (Django)-Projekt

Hinweise:(-) alle Befehle werden live durchgeführt und erklärt.
Diese werden auf den Folien nur nebenbei gezeigt)

```
django-admin startproject <project_name>
```

Erzeugt unser Django-Projekt und enthält:

__init__.py
settings.py
urls.py
wsgi.py
manage.py
(db.sqlite3)

In den <project_name> Ordner wechseln um

```
python manage.py runserver
```

Auszuführen. Der Server läuft.

Und unsere erste (Django) Applikation

`Python manage.py startapp <app_name>`

Erzeugt unsere Applikation und enthält:

__init__.py

admin.py

apps.py

models.py

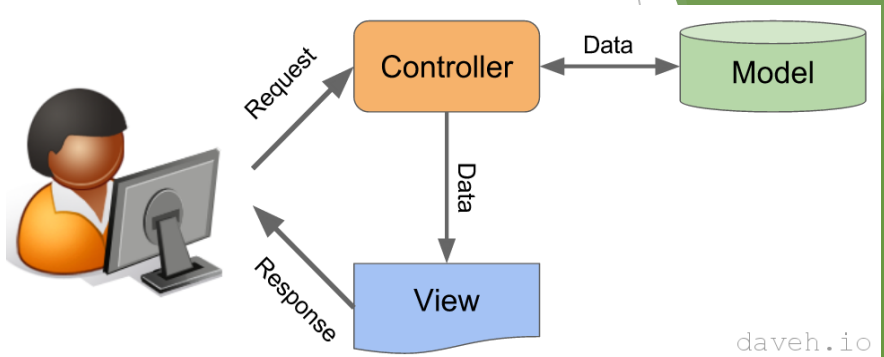
tests.py

views.py

2. Das ~~MVC~~ MTV Modell

- MVC

  Model - Repräsentation der Daten
  View - Darstellung
  Controll – Ereignisinterpreter (Steuerungseinheit)



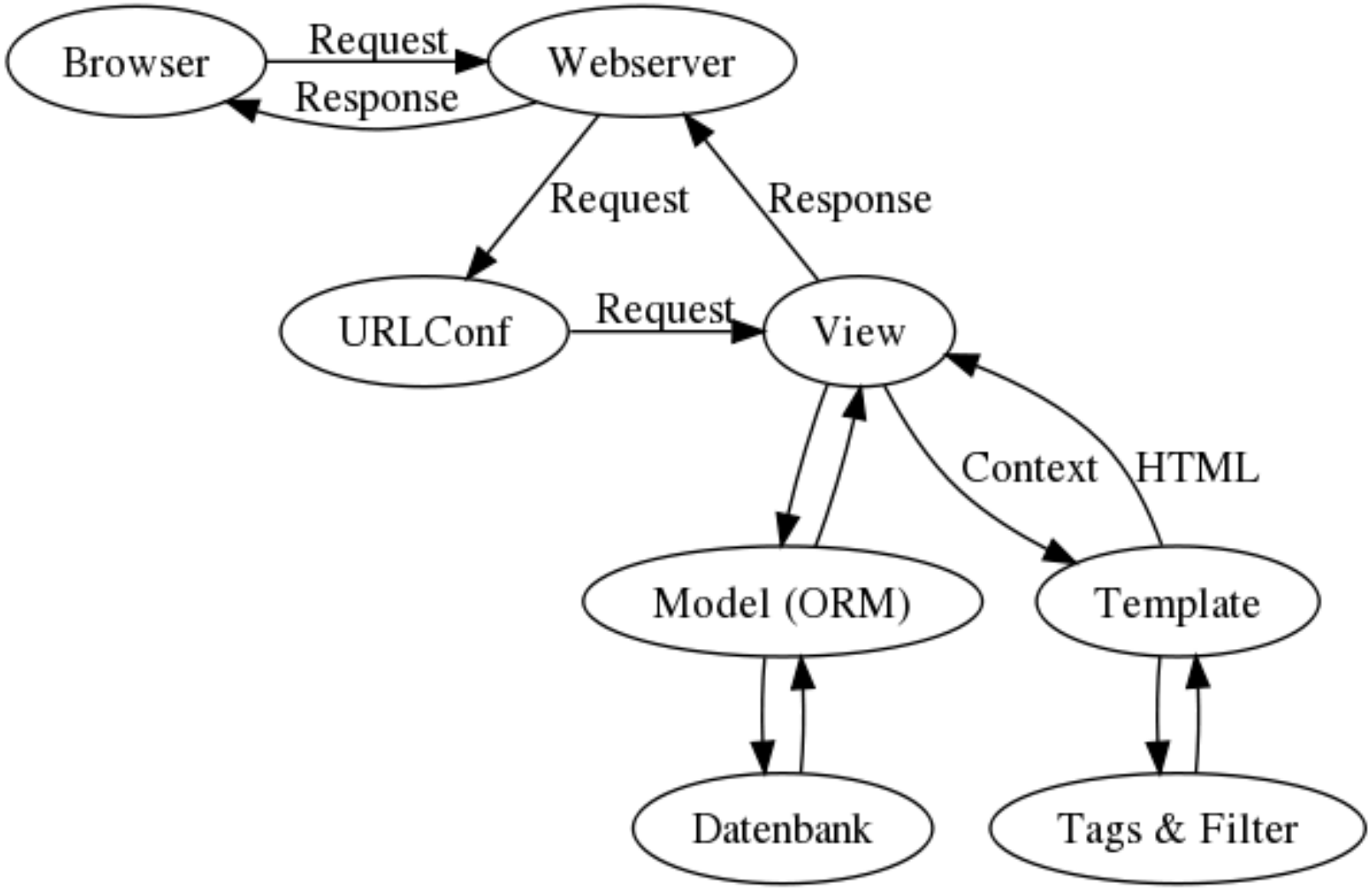
Quelle: https://daveh.io/blog/the-model-view-controller-pattern

- MTV

  Model - (direkter) Daten-Zugriff/Management
  Template - Darstellung
  View - Logik-Schicht (Brücke zwischen M und T)
  Controler?? Das 'C' wird vom Framework übernommen.
  - Django folgt MVC, aber MTV ist ein besserer Bezeichner

Django's Umgang mit Requests

1) erhalte Request /abc/
2) suche in den settings.py nach "ROOT_URLCONF"
3) suche in den "URLpatterns" nach einem Treffer
4) Bei Treffer: Folge dem Pfad und rufe die verbundene "View-Funktion" auf
5) Return von einer "HttpResponse"
   (Model / Template sind daran ebenfalls beteiligt)
6) Konvertierung "HttpResponse" --> HTTP-Code
   welche dann die Webseite darstellt



Schematische Darstellung einer Request / Response Verarbeitung

Quelle: http://www.django-workshop.de/einfuehrung.html#model-template-view

Konzept 1 – URL's

- Domain - URL

http://abcdefg.com/hij/.. --> macht jedes Framework (irgendwie)

- Django's Variante:

  Django nutzt 'reguläre Ausdrücke' (regular Expressions ("regex"))
  um url zu einem view zu leiten (mapping) .

Request-url path- zur View Name

```
url (r '^$',    'website.views.home', name= 'home'),
```
--> direkter Pfad

```
url (r '^website/' , include('website.hij.urls'))
```
--> ausgelagert (Übersicht)

--> so kann man aber nur 'statische' URL's generieren

--> dynamische sind ebenfalls möglich (erzeugt Lesbarkeit der URL's)

Beispiele für reguläre Ausdrücke

Symbol	Matches
. (dot)	Any single character
\d	Any single digit
[A-Z]	Any character between A and Z (uppercase)
[a-z]	Any character between a and z (lowercase)
[A-Za-z]	Any character between a and z (case-insensitive)
+	One or more of the previous expression (e.g., \d+ matches one or more digits)
[^/]+	One or more characters until (and not including) a forward slash
?	Zero or one of the previous expression (e.g., \d? matches zero or one digits)
*	Zero or more of the previous expression (e.g., \d* matches zero, one or more than one digit)
{1,3}	Between one and three (inclusive) of the previous expression (e.g., \d{1,3} matches one, two or three digits)

Quelle: http://djangobook.com/views-urlconfs/

Konzept 2 View's

```
def home(request) :
   return HttpResponse('Hello World')
```

```
def home(request) :
    return HttpResponse('<h1> Hello World </h1> ')
```

Beides liefert HTML-Text (nur in Python geschrieben)
Aber das wollen wir nicht (obwohl es legitim wär)

Lösung --> Templates zu Hilfe nehmen

Wir kommen gleich wieder auf die View's zurück

Konzept 3 Templates

- Im Grund genommen HTML-Code mit Platzhaltern
- Diese zu manipulieren (Filter)
- Und Möglichkeiten Python-Code einzubauen (Tags)

Tags:

Alles was in {% %} steht sind Tags (Python-Code)

```
{% if _bool_ %}
  # html-text
{% else %}
   # mehr html-text
{% endif %}
```

Analog dazu

```
{%for element in list %}
#html
{%end for%}
```

Oder Auslagerungen

```
{%block <c_name> %}
{% endblock %}
```

Konzept 3 Templates

Platzhalter und Filter

```
#HTML Code
{{ platzhalter | Filter }}  oder  {{platzhalter}}
#HTML Code
```

Platzhalter müssen im View via "Context" übergeben werden.
Im HTML steht schlussendlich NURNOCH(!) HTML-Code
Platzhalter und Tags sind nicht mehr sichtbar

Woher bekommen wir den "Context" ?
context = { platzhalter : 'hello world' } ist statisch
wir wollen dynamisches HTML

Konzept 4 Models

Problem: "Ich kann kein SQL.."
Models sind Datenbankeinträge, die in Python geschrieben sind

- Tabellen
- Generiere SQL für INSERT / READ / UPDATA / DELETE der DB Einträge
- ORM – Object Relational Mapper - Konzept dem Django folgt um DB zu verwalten
  (Abbildung von Objekten in relationale Datenbanken – Modellierungstechnik)

Die Klasse Model
model-klasse = Datenbank-tabelle
model-attributes = Datenkbank-Spalten (Attribute)
model-instanz = Datenkbank-Reihe (Eintrag)

Konzept 4 Models

```
Class Pet (models.Model):
    name = models.CharField(max_length=128)
```

--> Brauchen Tabellen in Datenbanken keinen Primärschlüssel?

Klasse models bietet alles was man aus SQL kennt

(FremdSchlüssel, n:m-Beziehungen, andere Datentypen, Beschränkungen, Defaults,…)

Es fehlen noch 2 Schritte:
1) Model in SQL umwandeln (an unser SQL-File schicken)
2) Model in unserem SQL-File aktualisieren (Bei Änderungen)

Migrations

- `python manage.py makemigrations Pet`

--> erstellt eine Tabelle für die Klasse Pet
python manage.py sqlmigrate <app_name> <Migr_ID>
lässt einen zeigen was genau Django hier gemacht hat

- `python manage.py migrate`

-Django sucht im Ordner settings.py nach "INSTALLED_APPS"

-Jeder Eintrag wird mit der DB synchronisiert (Änderungen werden übernommen, falls kompatibel)

Hinweis:
nicht vergessen die neuen Models in "INSTALLED_APPS" einzutragen!

Zurück zur View

```
def Show_all_Pets(request):

    All_pets = Pets.objects.all()  # Liste aller Pet-
                                   # Einträge aus der DB
    context = { 'all_pets' : all_pets}
    return render(request, 'show_pets.html', context)
```

Damit wären alle Teile komplett und auf der Webseite erscheint HTML-Code bei dem alle Platzhalter durch unseren Kontext ersetzt wurden.

Die Django API - Shell

- Tests / Datenbankmanipulationen

```
python manage.py shell
```

```
>>> from <app_name>.models import <table_name>
```

man erhält Zugang auf diese Tabelle ohne die DB aktiv zu verändern,
außer man möchte das

```
>>> a = pet(name='fifi')
>>> a.save()      <-- damit wäre eine neue Reihe angelegt
                   der Variablenname "a" hat keine weitere Bedeutung
                   (hat nur während dieser Shell-Sitzung Relevanz)
```

Die Admin-Applikation

- Bereits vorhanden
- Management von Datenbanken, Nutzern

Admin erstellen:

```
python manage.py createsuperuser
```

Quellen

DJANGO:

- http://djangobook.com/
- https://docs.djangoproject.com/en/1.11/
- https://www.youtube.com/user/thenewboston

PHYTON:

- https://docs.python.org/3/

MVC:

- http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/mvc

Bild:

- Quelle: http://www.craighewitt.me/rails-vs-django-vs-laravel-an-analysis-of-web-frameworks-from-a-non-technical-founder/

Fragen

)