12 tips on Django Best Practices

David Arcos

12 tips on Django Best Practices

Some (personal) suggestions on:

- Development

- Deployment

- External tools

12 tips on Django Best Practices

Hi! I'm **David Arcos**

- Python/Django developer
  - discovered Django in 2007 (v0.96)
  - professionally since 2008

- Web backend, distributed systems, databases, scalability, security

- Team leader at **Catchoom**

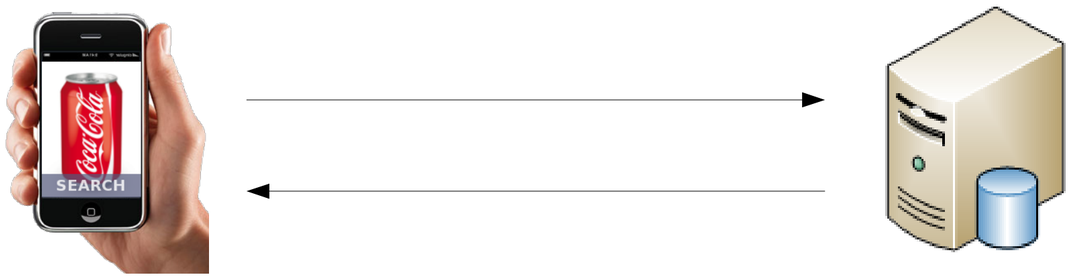
12 tips on Django Best Practices

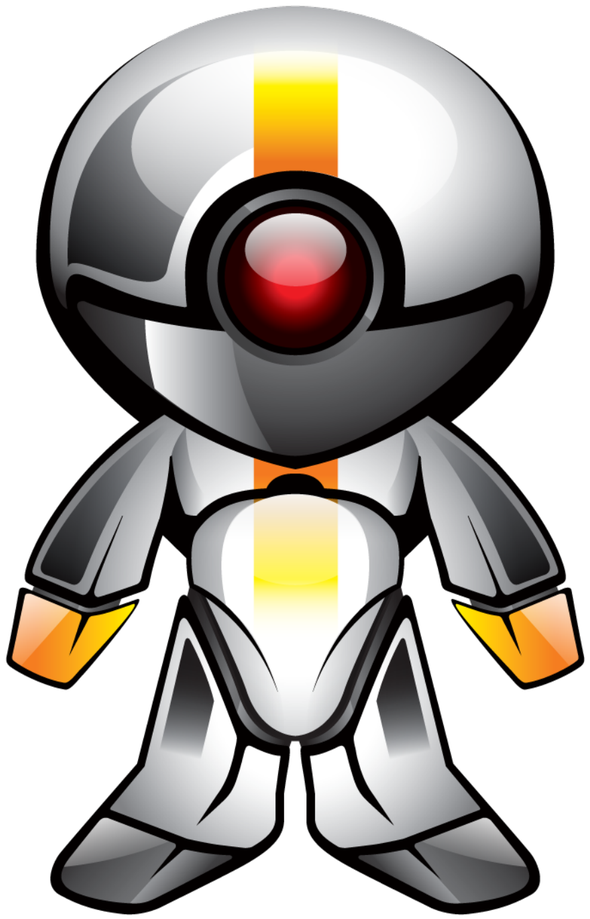
catchoom

- Image Recognition SaaS

- Top recognition results, shortest response times

- Easy to integrate into your apps/services





12 tips on Django Best Practices

Why Python? PEP 20, The Zen of Python:

- “Beautiful is better than ugly”
- “Simple is better than complex”
- “Complex is better than complicated”
- “Readability counts”
- “Special cases aren't special enough to break the rules”
- “If the implementation is hard to explain, it's a bad idea”
- (…)

http://www.python.org/dev/peps/pep-0020/

python

12 tips on Django Best Practices

Why Django?

- "*The Web framework for perfectionists with deadlines*"

- Django design philosophy:
  - loose coupling, less code, DRY, consistency, etc...
  - https://docs.djangoproject.com/en/dev/misc/design-philosophies/

- technically: tons of django apps, very good doc

12 tips on Django Best Practices

Virtualenv:

- “one project, one virtualenv”

- projects with different dependencies, package versions

- easier to deploy. Forget dependency hell!

- **virtualenvwrapper** is a convenient tool

12 tips on Django Best Practices

Dependencies:

- use **pip**:

```
pip install catchoom
```

- save the dependencies in a *requirements.txt* file:

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

12 tips on Django Best Practices

Layout: projects and apps

- project = the full website. app = python library

```
repository/
|-- doc
`-- project
    |-- apps
    |   |-- app1
    |   |-- app2
    |   `-- app3
    `-- settings
```

12 tips on Django Best Practices

- use short, obvious, single-word names for your apps
- *many small apps* is better than *a few giant apps:*
  - explain an app in a sentence. If you can't, split the app
  - rather than expand an app, write a new app

- don't reinvent the wheel!
  - `django.contrib`
  - 3rd-party apps

12 tips on Django Best Practices

Settings:

- multiple settings files:
  - per environment: *dev, testing, staging, production*
  - per developer (local settings, use the hostname)

- all settings files must inherit from base, so you can do:

```
INSTALLED_APPS += ('debug_toolbar', )
```

- **version control** all the settings!

12 tips on Django Best Practices

Django is a MTV framework

- Model (`app/models.py`)

- Template (`app/templates/*.html`)

- View (`app/views.py`)

12 tips on Django Best Practices

Fat models, thin views...

- logic should go to the models (and forms, signals...)
- keep the views at a minimum

Good example: `django.contrib.auth.models.User`

12 tips on Django Best Practices

- why? Because of maintainability!

- a model is much easier to test

- reusable logic: form validation, signals, etc

- the code becomes clearer, more self-documenting

12 tips on Django Best Practices

...and stupid templates!

- your template layer *should* be as thin as possible
  - (by design) django templates are limited, constrained
    - doesn't fit your use case? Use **jinja2** in those views

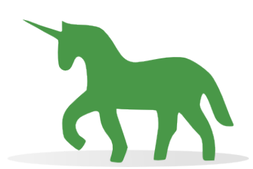
- Hey, but I get ugly generated HTML!
  - doesn't matter, you want **maintainable** templates

12 tips on Django Best Practices

Deployment:

- web server:

  - Nginx + gunicorn
  - Supervisord to keep it alive.



- static server:

  - Nginx. Or any CDN.

12 tips on Django Best Practices

Fabric:

"a library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks"

- to automate deployments, migrations, execute management commands, monitoring...

- no more repetitive maintainance tasks done manually!

12 tips on Django Best Practices

South:

"intelligent schema and data migrations for Django projects"

- creates migration files automatically.
  - You can still do changes

- can do backward migrations

- will avoid disasters. **Use it!**

12 tips on Django Best Practices

Celery:

"asynchronous task queue/job queue based on distributed message passing"

- execute tasks asynchronously, in a pool workers
  - cpu-intensive or I/O-intensive tasks:
    - emails, pdfs, thumbnails, crawling, requests...

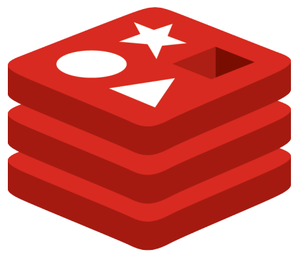
  - Celery needs a Message Queue
    - Instead of RabbitMQ, try Redis.

12 tips on Django Best Practices

Redis:

"An advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets."

- store ephemeral data (active sessions)
- general cache (memcached compatible)
- real-time calculations: stats, monitoring, throttling...
- messages: channels (pub-sub), lists (push/blpop)
- indexes/filters (“sort by hits”)



redis

12 tips on Django Best Practices

Sentry:

"realtime event logging and aggregation platform"

- monitor errors, get all the info to do a post-mortem

- uses the Python logger, easy to configure

- deploy a Sentry instance
  - or use getsentry.com

12 tips on Django Best Practices

Debugging:

- ipython (already in `./manage.py shell`)

- ipdb

```
import ipdb
ipdb.set_trace()
```

- django-debug-toolbar
  - very powerful
  - use it to optimize db performance, view by view

12 tips on Django Best Practices

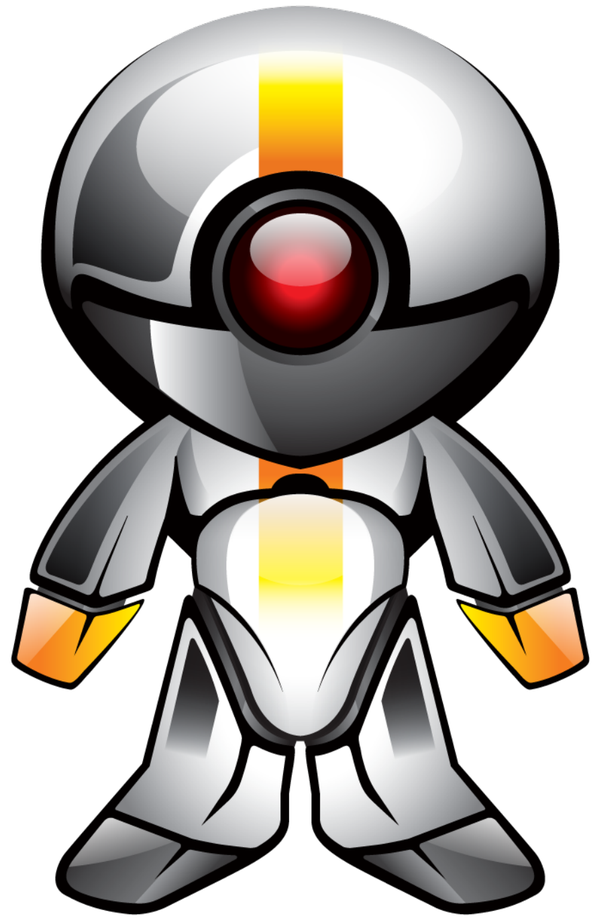
Summary:

- follow the Django philosophy (when possible)
  - stand on the shoulder of giants: use existing apps

Thanks for attending!

- http://slideshare.net/DZPM

- Questions?

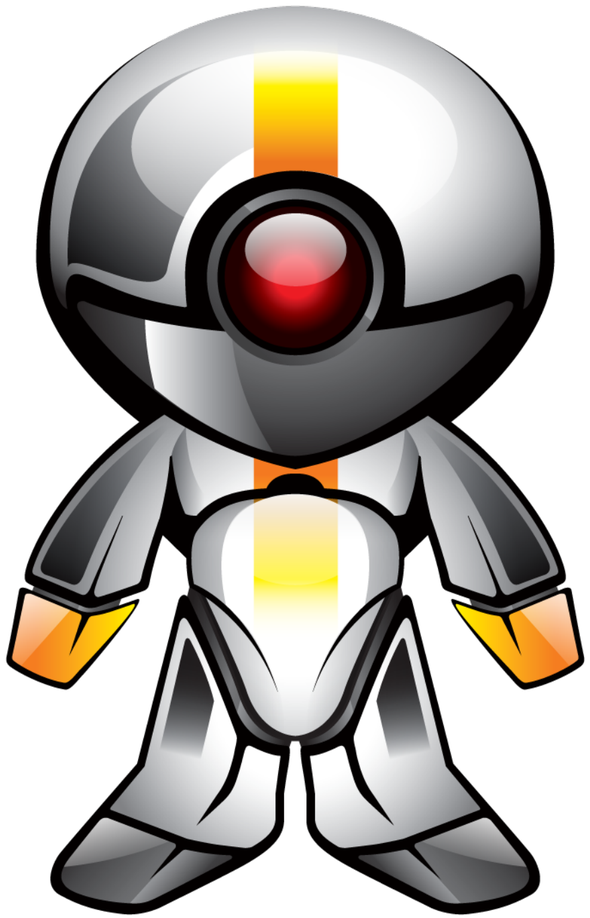


catchoom

Questions?

Thanks for attending!

- http://slideshare.net/DZPM



catchoom