

Django's System Check Framework

Adam Johnson - me@adamj.eu

13 July 2015

What is it?



- It checks your code!

What is it?

- ▶ Added to Django in version 1.7
- ▶ Automatically checks things after 'ready' time, such as models, deprecated features, and contrib apps
- ▶ Can be run alone as manage.py command
- ▶ Looks like this:

```
$ ./manage.py check
System check identified some issues:
```

```
WARNINGS:
```

```
?: (1_6.W001) Some project unittests may not execute as
expected.
```

```
HINT: Django 1.6 introduced a new default test
runner. It looks like this project was generated
using Django 1.5 or earlier. You should ensure your
tests are all running & behaving as expected. See https://
for more information.
```

Why is it good?



- Code quality!

Why is it good?

- ▶ “Beyond linting” - code quality checks that can only be done at runtime, not by flake8
- ▶ Extensible - third party apps can provide checks too
- ▶ Runs on nearly every manage command - you can't forget to run it, unlike tests!

Using it



- How do you use it?

Using it

- ▶ Normally it does its work without you realizing, until you see an error and correct it
- ▶ However it's easy to write your own checks - the framework is very simple
- ▶ At YPlan, we already had some sanity checks in place as unit tests - converting them to checks has improved the situation
- ▶ I'll show you how now

An ex-unit test

- ▶ An example old sanity check from the YPlan codebase:

```
class SupportedFeaturesTest(SimpleTestCase):
    """Check features that might not be supported in
    some OS versions"""
    def test_strftime_dash_strips_leading_zeroes(self):
        # Not every strftime implementation has %- to
        # strip leading zeroes, but we use it
        dt = datetime(2000, 10, 11, 2, 12)
        self.assertEqual(
            dt.strftime('%-I:%M %P'),
            '2:12 am'
        )
```

- ▶ Bad! The test might not be run; or if it is run and fails, its output is buried amongst all the other test failures based upon this strftime feature

Better!



► Let's make it better!

Writing a check

- Checks return lists of Errors:

```
from django.core.checks import Error, Tags, register

@register(Tags.compatibility)
def check_system(app_configs, **kwargs):
    # app_configs is a list of AppConfig objects
    # kwargs is for future extension
    errors = []
    if bad_thing():
        errors.append(
            Error(
                "My error message",
                id='mycode.001'
            )
        )
    return errors
```

- Put in myapp/apps.py next to your MyAppConfig class

Converting our unit test

- Magicked into a mini check function:

```
def check_system(app_configs, **kwargs):
    errors = []
    errors.extend(_check_strftime())
    return errors

def _check_strftime():
    errors = []
    dt = datetime(2000, 10, 11, 2, 12)
    if dt.strftime('%-I:%M %P') != '2:12 am':
        errors.append(Error(
            "strftime does not appear to support using "
            "%- to strip leading zeroes",
            id='yplan.E007'
        ))
    return errors
```

It's better

- ▶ Now the check runs at `runserver` or most other `manage.py` commands
- ▶ If it fails, it's unignorable
- ▶ It takes less than a millisecond to run so doesn't really impact startup time



- ▶ Dev: "Hey Adam, code is failing with ImportError"
- ▶ Me: "...you didn't pip install requirements.txt like I said on skype ten minutes ago, did you? Does anyone listen to me??"

requirements.txt

- ▶ A communication bottleneck, especially with nearly 100 dependencies!
- ▶ Worse when developing several branches concurrently, only one of which is based on the new version of a package



Solution: a system check!!

- ▶ At launch, compare pip freeze to parsed lines of requirements.txt
- ▶ Basically:

```
from pip.operations.freeze import freeze as pip_freeze

def check_requirements(app_configs, **kwargs):

    with open(REQUIREMENTS_TXT_PATH, 'r') as f:
        requirements = pip_parse(f.read().split('\n'))

    current = pip_parse(pip_freeze())

    errors = []
    for name, version in requirements.iteritems():
        # Compare with current – if missing or wrong
        # version, append an Error() to errors...

    return errors
```

Solution: a check

- ▶ Example output:

```
$ ./manage.py runserver
CommandError: System check identified some issues:

ERRORS:
?: (yplan.E002) Package patchy has version 1.1.0 in
  requirements.txt but you have version 1.0.0 installed
  - you should 'pip install -r ../requirements.txt '

System check identified 1 issue (0 silenced).
```

- ▶ Will open-source this. Haven't solved git-based package versions yet

Thank you



► me@adamj.eu ; blog at adamj.eu/tech/