Django is awesome!
But it can’t come with all the awesome.
So let’s go shopping!
A Day in the Life
A Day in the Life

- Django Site
- Django’s template engine
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
- No views, only REST
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
- No views, only REST
- Plus a whole lot of JavaScript nonsense
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
- No views, only REST
- Plus a whole lot of JavaScript nonsense
- Start using lots of Redis
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
  - No views, only REST
  - Plus a whole lot of JavaScript nonsense
  - Start using lots of Redis
  - Start accessing RDBMS less
A Day in the Life

- Django Site
  - Django’s template engine
  + Plus Jinja2
- No views, only REST
- Plus a whole lot of JavaScript nonsense
- Start using lots of Redis
  - Start accessing RDBMS less
  - Some WSGI Wizardry
A Day in the Life

• Django Site
  • - Django’s template engine
  • + Plus Jinja2
• No views, only REST
• Plus a whole lot of JavaScript nonsense
• Start using lots of Redis
  • Start accessing RDBMS less
  • Some WSGI Wizardry
    • Maybe Flask for this bit?
A Day in the Life

• Django Site
  • - Django’s template engine
  • + Plus Jinja2
• No views, only REST
• Plus a whole lot of JavaScript nonsense
• Start using lots of Redis
  • Start accessing RDBMS less
  • Some WSGI Wizardry
  • Maybe Flask for this bit?
• I hear node is cool..
Batteries included?
TEMPLATES & VIEWS & AUTH & MODELS
Templates
Templates with Jinja2

https://github.com/mitsuhiko/templatetk/blob/master/POST_MORTEM

- Jinja2 is basically Django Templates++
- More expressive language
- Byte-code rendering speedups
- Sites using Django+Jinja:
  - Mozilla
  - Pitchfork
{% extends "base.html" %}
{% block title %}home{% endblock %}

{% block content %}
{% if user %}
Hello {{ user['username'] }}
{% endif %}
{% endblock content %}

Could not parse the remainder: ['username'] from 'user['username']'
from django.http import HttpResponse
from django.template import RequestContext

from jinja2 import Environment, PackageLoader
env = Environment(loader=PackageLoader('ourapp', 'templates'))

def jinja_render(request, template_name, dictionary=None):
    if not dictionary:
        dictionary = {}
    template = env.get_template("index.html")
    new_context = RequestContext(request, dictionary)
    context_dict = {}
    for d in new_context.dicts:
        context_dict.update(d)

    rendered_template = template.render(**context_dict)
    return HttpResponse(rendered_template)

def home(request):
    # return render(request, "index.html")
    return jinja_render(request, "index.html")
Jingo
https://github.com/jbalogh/jingo/

- Install Jingo.
- Use all the Django functions.
A Kill To A View
A Brief Over-View

- “A view is a callable which takes a request and returns a response.”
A Brief Over-View

“A view is a callable which takes a request and returns a response.”

haha, get it?
A Brief Over-View

• "A view is a callable which takes a request and returns a response."

django.http.HttpRequest

haha, get it?
A Brief Over-View

django.http.HttpRequest

• “A view is a callable which takes a request and returns a response.”

django.http.HttpResponse

haha, get it?
A typical view

from django.shortcuts import render
from people.models import Person

def living_people(request):
    return render(request,
                  template = 'people/living.html',
                  context = {
                    'people': Person.objects.filter(alive=True)
                  }
    )
```python
def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = {
        'people': Person.objects.filter(alive=True)
    }
    body = tmpl.render(context)
    return http.HttpResponse(body)
```

from django import http, template
from people.models import Person

def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = template.Context({'people': Person.objects.filter(alive=True)})
    body = tmpl.render(context)
    return http.HttpResponse(body)
from django import http, template
from people.models import Person

def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = template.Context({'people': Person.objects.filter(alive=True)})
    body = tmpl.render(context)
    return http.HttpResponse(body)
from django import http, template
from people.models import Person

def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = template.Context({'people': Person.objects.filter(alive=True)})
    body = tmpl.render(context)
    return http.HttpResponse(body)
from django.conf import settings
from django import http, template

import redis
db = redis.from_url(settings.REDIS_URL)

def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = template.Context({'people': db.get('people:living')})

    body = tmpl.render(context)
    return http.HttpResponse(body)
from django.conf import settings
from django import http, template

import redis
db = redis.from_url(settings.REDIS_URL)

def living_people(request):
    tmpl = template.loader.get_template('people/living.html')
    context = template.Context({'people': db.get('people:living')})

    body = tmpl.render(context)
    return http.HttpResponse(body)
import json
import redis
from django import http
from django.conf import settings

db = redis.from_url(settings.REDIS_URL)

def living_people(request):
    body = json.dumps({
        'people': db.get('people:living')
    })
    return http.HttpResponse(body, content_type='application/json')
import json
import redis
from django import http
from django.conf import settings

db = redis.from_url(settings.REDIS_URL)

def living_people(request):
    body = json.dumps(
        {'people': db.get('people:living')}
    )
    return http.HttpResponse(body, content_type='application/json')

Template engine? We don’t need no steeeking template engine.
import json
import redis
from django import http
from django.conf import settings

db = redis.from_url(settings.REDIS_URL)

class LivingPeople(object):
    def __call__(self, request):
        body = json.dumps(
            {'people': db.get('people:living')}
        )
        return http.HttpResponse(body, content_type='application/json')
Hey, I’m a class view
And this is crazy
But I have a __call__ method
So call me maybe

```python
import json
import redis
from django import http
from django.conf import settings

db = redis.from_url(settings.REDIS_URL)

class LivingPeople(object):
    def __call__(self, request):
        body = json.dumps({
            'people': db.get('people:living')
        })
        return http.HttpResponse(body, content_type='application/json')
```
... and even this.

```python
import redis
from django.conf import settings
from rest_framework import viewsets
from rest_framework.response import Response

db = redis.from_url(settings.REDIS_URL)

class LivingPersonViewSet(viewsets.ViewSet):
    def list(self, request):
        people = db.get('people:living')
        return Response(people)

    def retrieve(self, request, pk):
        person = db.get('people:living:%s' % int(pk))
        return Response(person)
```
django

REST

framework

TASTYPIE
Django Rest Framework + AngularJS = (well, you know)

Auth
django."contrib".auth

- django.contrib.auth.middleware.AuthenticationMiddleware
- django.contrib.sessions.middleware.SessionMiddleware
- django.contrib.auth.backends.ModelBackend
- django.contrib.auth.models.User
- django.contrib.auth.context_processors.auth
What auth needs to do

- Securely (hopefully!) verify someone’s identity.
- Set `request.user` to an object that quacks like a user.
- That object should probably be a model instance.
Secure is Hard
Insecure Is Easy!
<table>
<thead>
<tr>
<th>Gist Details</th>
<th>Created</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>django-minus / password.txt</strong></td>
<td><img src="frank.png" alt="Frank" /></td>
<td>Last active 6 minutes ago</td>
</tr>
<tr>
<td>Created 4 hours ago</td>
<td>1 file</td>
<td>0 forks</td>
</tr>
<tr>
<td>frank</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>wiles</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>django-minus / password.txt</strong></td>
<td><img src="otherjacob.png" alt="OtherJacob" /></td>
<td></td>
</tr>
<tr>
<td>Created 4 hours ago</td>
<td>1 file</td>
<td>0 forks</td>
</tr>
<tr>
<td>otherjacob</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>password</td>
<td></td>
</tr>
</tbody>
</table>
from gist_auth.models import GistUser

class SuperInsecureGistAuthBackend(object):
    
    def authenticate(self, **credentials):
        username = credentials['username']

        gists = self.get_gist_users()

        if username not in user_gists:
            return None

        user_dict = self.get_user_data(user_gists[username])
        real_password = user_dict['files']['password.txt']['content']

        if credentials['password'] == real_password:
            user, created = GistUser.objects.get_or_create(username=username)
            if created:
                user.gist_id = user_dict['id']
                user.save()

            return user
from django.contrib.auth.models import AbstractUser
from django.db import models

class GistUser(AbstractUser):
    gist_id = models.CharField(max_length=255, unique=True)

AUTH_USER_MODEL = 'gist.orm.GistUser'  # app_name.label_of_model
AUTHENTICATION_BACKENDS = (
    'gist.orm.auth.SuperInsecureGistAuthBackend',
)
django.contrib.auth

- django.contrib.auth.backends.ModelBackend
- django.contrib.auth.models.User
- django.contrib.auth.context_processors.auth
- django.contrib.auth.middleware.AuthenticationMiddleware
from gist_auth.models import GistUser

class AuthRoulette(object):
    
    def process_request(self, request):
        random_user = GistUser.objects.order_by('?')[0]
        request.user = random_user
Typical use-case: SSO
Now, the model layer. Seems easy, how hard could it be?
DATABASES = {}

Done.
DATABASES = {}
“Just Say No”

- “Removing” Django’s model layer is easy: just don’t use it.
- However, there are consequences for your insolence...
import json
import redis
from django import http
from django.conf import settings

db = redis.from_url(settings.REDIS_URL)

def living_people(request):
    body = json.dumps({'people': db.get('people:living')})
    return http.HttpResponse(body, content_type='application/json')
Frameworks are like ogres

- Frameworks are layered for a reason.
- Coupling of models—of any flavor—into your view code is an anti-pattern.
Strategies for safe model replacement

- Continue to enforce strong separation of concerns.
- Use an existing encapsulation abstraction (SQLAlchemy ORM, MongoEngine, etc.), or “POPOs”.
class Person(object):
    db = redis.from_url(settings.REDIS_URL)

@classmethod
def get_living_people(cls):
    return cls.db.get('people:living')
Separation of concerns

```python
import redis
from django.conf import settings

class Person(object):
    db = redis.from_url(settings.REDIS_URL)

    @classmethod
    def get_living_people(cls):
        return cls.db.get('people:living')

import json
from django import http
from .models import Person

def living_people(request):
    body = json.dumps(Person.get_living_people())
    return http.HttpResponse(body, content_type='application/json')
```
PersonForm = ModelForm(Person)
PersonForm = ModelForm(Person)
PersonForm = ModelForm(Person)
“Doctor, it hurting when I do this.”

- Many, many things depend on Django’s model layer, or things that themselves depend on models.
- In particular, ModelForms and Auth. And what uses ModelForms and Auth?
- ... that’s right, the Admin.
- Similarly, many larger 3rd-party apps won’t work with custom model layers, either.
- It can be done (see, for example, github.com/vpulim/mango), but the tradeoffs can be difficult.
Does Django play well with others?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templates</td>
<td>A</td>
</tr>
<tr>
<td>Auth (extending)</td>
<td>A</td>
</tr>
<tr>
<td>Auth (replacing)</td>
<td>C</td>
</tr>
<tr>
<td>View</td>
<td>B</td>
</tr>
<tr>
<td>Models</td>
<td>A*</td>
</tr>
</tbody>
</table>
What’s Left

- Settings
- URLConf/Routing
- request → middleware → response cycle
- Forms.
  - But forms are cool!
“This part is left as an exercise to the reader”

- Remove these last bits with pure WSGI.
- Let us know how hard it was.
Thank you

Jacob Burch
Revolution Systems
@jacobburch

Jacob Kaplan-Moss
Heroku
@jacobian

Need a job? Heroku is hiring!

Need someone to do a job? Revsys is hirable.