Django Best Practices

By Scott Woodall

Things we're going to cover

(assumes basic Django knowledge)

- These are our own practices
- Code organization
- Models, forms, views, managers
- Load up your local database with data
- Cutting down the query counts
- Postgresql performance tips
- Making the most of a server side rendered app in comparison to SPA

Who is Ivy Tech Community College?

- A large school with ~175k students (4th in US Wikipedia, 2nd in US CollegeStats)
- Been using Django since 2010, former Perl shop
- Over 30 custom applications built for students, staff, and faculty
- Small team of four developers, one UX/UI, one QA. (all web apps, api integrations)
- ~700:1 ratio of student to IT staff. Notre Dame is ~40:1.

IVY TECH COMMUNITY COLLEGE

Who is Scott Woodall?

- Director of Application Development
- 10 years experience as Unix administrator
- Worked at Ivy Tech for 13 years
- Helped start and build out the Ivy Tech development team
- Been doing software development past 9 years
- LOVE software development

IVY TECH COMMUNITY COLLEGE

Code Organization

(our source code breaks down as follows)

- models.py (Django models)
- forms.py (Django forms)
- services.py (IO related function calls: email, http)
- utils.py (formatting, helper functions)
- views.py (Django views)
- tasks.py (Celery tasks)
- managers.py (common ORM queries)
- management/commands/example.py (wrapper around tasks/services calls)
- Domain logic in own python module

Starting Brand New Application

(save views.py for last!)

- Get the models, forms, tasks, services all working first
- Getting the data model built correctly is important
- Writing views/UI takes most amount of time
- Leverage the Django admin as your views in the beginning
- But I can't do X, Y, or Z with the Django Admin!!
- Tons of hooks into the Django admin that allows you to get behavior we want
- Helps speed up development because works out the data model, flow before UI/UX dev

```
class MyModelAdmin(admin.ModelAdmin):
    def get_form(self, request, obj=None, **kwargs):
        if request.user.is_superuser:
            kwargs['form'] = MySuperuserForm
        return super().get_form(request, obj, **kwargs)
```

Django Models

(use Postgresql if you can, you get extra goodies!!)

- django.contrib.postgres
- Always use fields that match your data types. Don't need floats? Use integers.
- Avoid GenericForeignKey (can be a foreign key in Table A or Table B). Complicates ORM calls.
- Never allow CharField(null=True). Now three values for CharField.
- Don't be afraid to put methods/properties on your models.
  - Rule of thumb; if you want to ask a question of the data on your model, make it a property/method
  - Don't want this code in views or templates
  - Allows logic to travel around with the model and isn't locked up in a view/template
- Leverage the relationships between your models in ORM calls. Helps keep code clean.

Django Forms

(ALL user supplied data must pass through a form, no exceptions!)

- 99.99% of the time you should be working with model forms.
- Always, always, always pass user data through a form, Never do this
- Forms convert string data to native python types ('10' string becomes 10 integer). Reduces errors
- Need to make an API to call to something? Stick the method on a form.
- Forms become an interface to your application
- Have multiple forms for same model, different use cases

Django Model Managers

(stuff all your queries in here!!)

- Save yourself from repeating the same ORM calls all over your views
- Logic now travels around with the model

Django Views

(function based or class based, take you pick!)

- Brand new to Django? Use function based views, easier to understand
- Complicated view, lots of moving parts? Use a function based view.
- Interacting with a single model in a CRUD fashion? Use a class based view.
- Mix and match is OK
- Red flag that refactoring needs to take place if a view gets too big
- Make views small, do one thing, it's OK to have a lot of small views interacting with the same model.

Fake data in your local database

(be proactive in uncovering performance issues)

- We use Factory Boy for unit tests
- Create ~500k rows for each table using a model factory
- Start navigating around your application
- You'll find where indexes are needed, list pages that need server side pagination, select widgets too much data

Django Debug Toolbar

(keep those page query counts low!)

- Biggest performance improvement for us is keeping query count low and performant
- Look out for where query count increases as you add another row into the table. Want query count to remain constant
- Navigate around your application and address any page with duplicates and query count > 30
- You'll make heavy use of select_related/prefetch_related

Quick Postgres Performance Check

(poor mans postgres profiler)

- Enable “pg_stat_statements” in postgresql.conf
- Link to gist
- Link to depesz.com explain plan
- Can use pgBadger for more detailed information when you need more info

All this work results in

(happy developers and end users!!)

- Better experience for developers (code laid out nice, fun to work in, data types make sense)
- Fewer bugs because you're using forms right!??!!
- Nice performance because we were proactive instead of reactive (busiest time of year, 30 req/sec)

Get the SPA feel without the cost

(don’t reach for React/Vue/Angular…. just yet!)

- We can come close to the Single Page Application (SPA) feel with server side rendered (SSR) tech
- There are scenarios where SPA’s are the right choice, a lot of scenarios where they are not
- There’s been an evolution of tools to help get SSR to SPA like
  - Pjax => Turbolinks => Intercooler => Unpoly
  - Unpoly Gist example
- Progressive Web app. Leverage Google Workbox to do the heavy lifting

Django Has Been Good To Us

(Not a single time where Django hasn't been able to provide a solution)

- Multiple database support, connects to our Oracle ERP system
- RSS feed generation
- Search engine site.xml generation
- Caching
- Extensible (Love Crispy forms, Django Braces)

Thank You! Questions?

We’re hiring
link.ivytech.edu/python

IVY TECH COMMUNITY COLLEGE