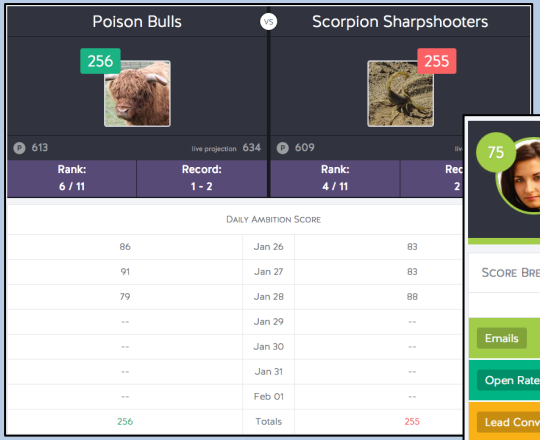
Minimizing Django Growing Pains in a Fast Paced Startup

Wes Kendall, Micah Hausler, Rob deCarvalho, Jeff McRiffey

What is Ambition?

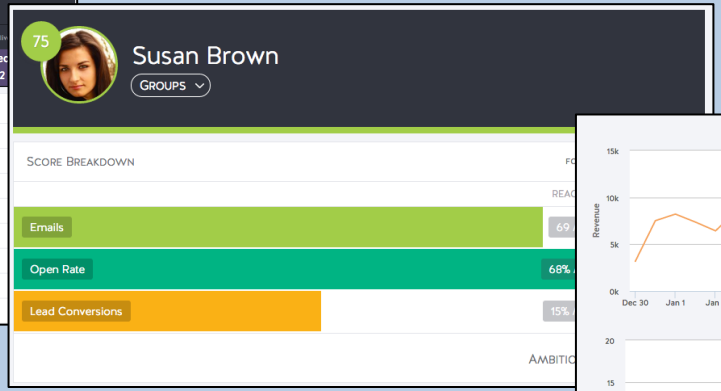
Ambition is an engagement platform for businesses

Achievement



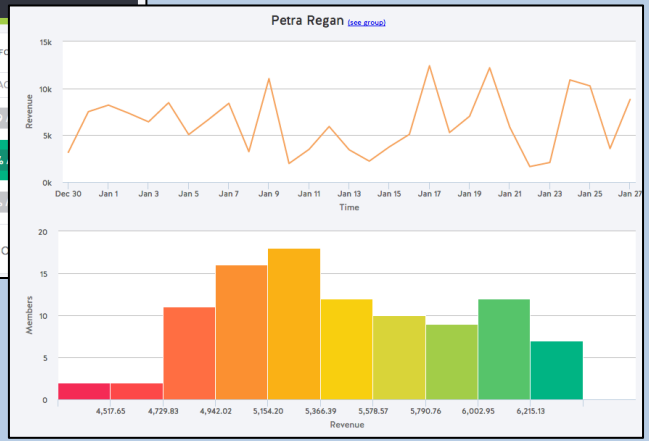
Competition, Leaderboard, Highlights

Accountability



Dashboard

Awareness



Reporting

Over The Past Two Years...

- Scaled to 8 engineers
- Scaled to over 50+ apps and 5 projects
- Over 1M lines of Python code
- All while going through YCombinator and scaling up customer base with product pivots

Minimizing Growing Pains

While growing, we have been able to:

- Automate code reliability, quality, and standards
- Minimize product pivot overhead by architecting key software abstractions
- Apply same principles in UI / Javascript design

Automating Best Development Practices

Micah Hausler

Code Quality

100% Test Coverage

- But not all bugs are caught with 100% coverage!
- That last 1% of coverage are often edge cases: here be dragons
- High degree of confidence that a PR is not introducing a new regression
- Broken window theory

Static Analysis

- Flake8/pylint are your friends
- Reduce unused code statements
- Reduce complexity
- Increase standardization

Continuous Integration

- This ties it all together
- You should be running your tests/static analysis on every commit
- You will always know the state of the code

Documentation

Documentation

- Zero examples = no one using your code
- The more complicated something becomes, the more likely you are to forget it
- Written documentation distributes knowledge within your organization

README

Original

README.md

django-query-builder

Current

README.rst

build passing coverage 95% pypi 0.5.9 downloads 525/month

django-query-builder

querybuilder is a django library for assisting with the construction and execution of sql. This is not meant to replace django querysets; it is meant for managing complex queries and helping perform database operations that django doesn't handle.

Why use querybuilder?

The django querybuilder allows you to control all parts of the query construction. This is happens more clearly because the function calls more closely represent the actual sql keywords.

Why not just use django's .raw() function?

While the raw function lets you execute custom sql, it doesn't provide any way for the developer to build the query dynamically. Users lacking experience writing "raw" sql should avoid using querybuilder and stick with django's querysets. The querybuilder's query construction closely mirrors writing sql, where django querysets simplify the sql generation process for simple queries.

Requirements

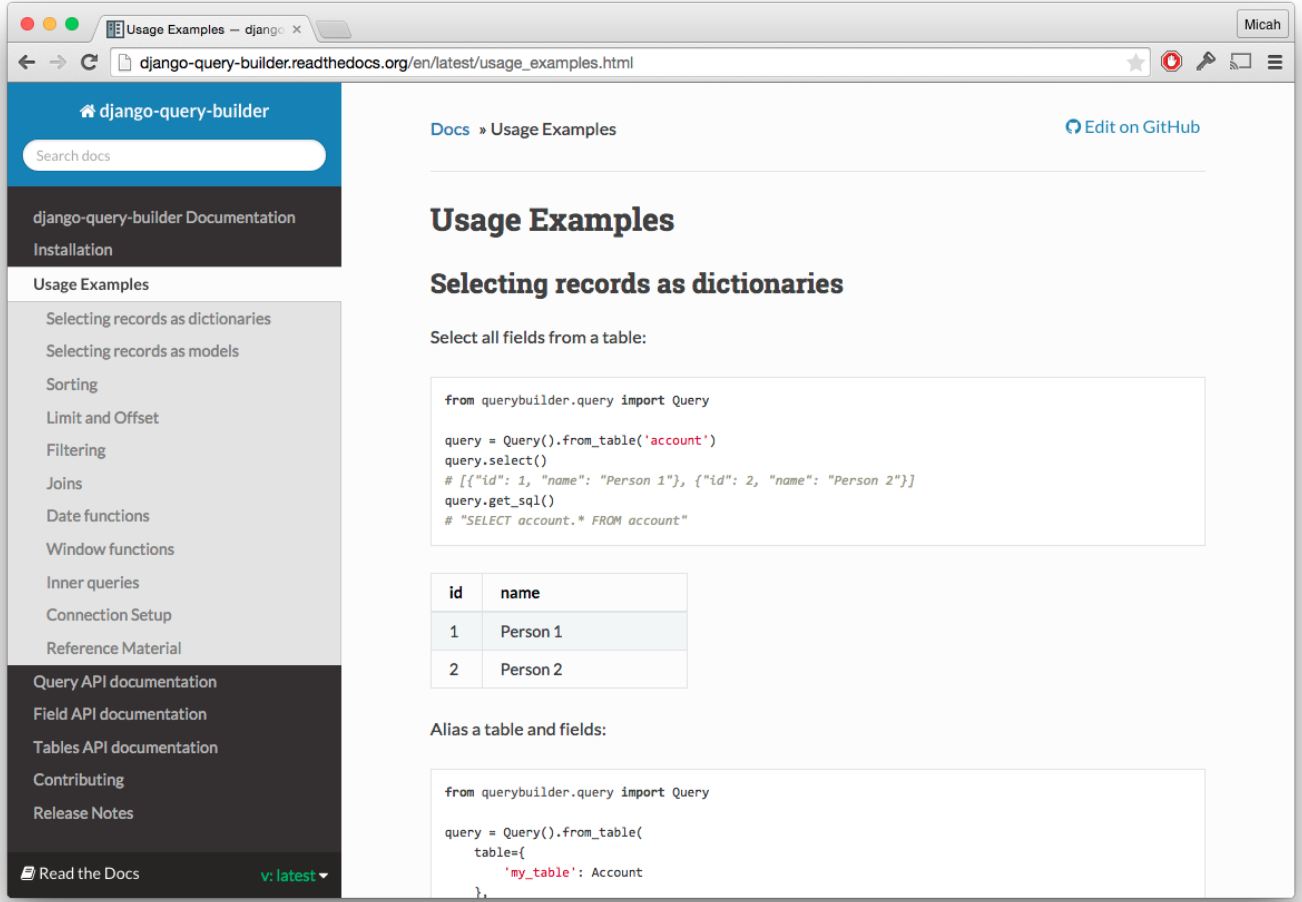
- Python 2.7
- Python 3.3, 3.4
- Django 1.4+

Installation

To install the latest release, type:

```
pip install django-query-builder
```

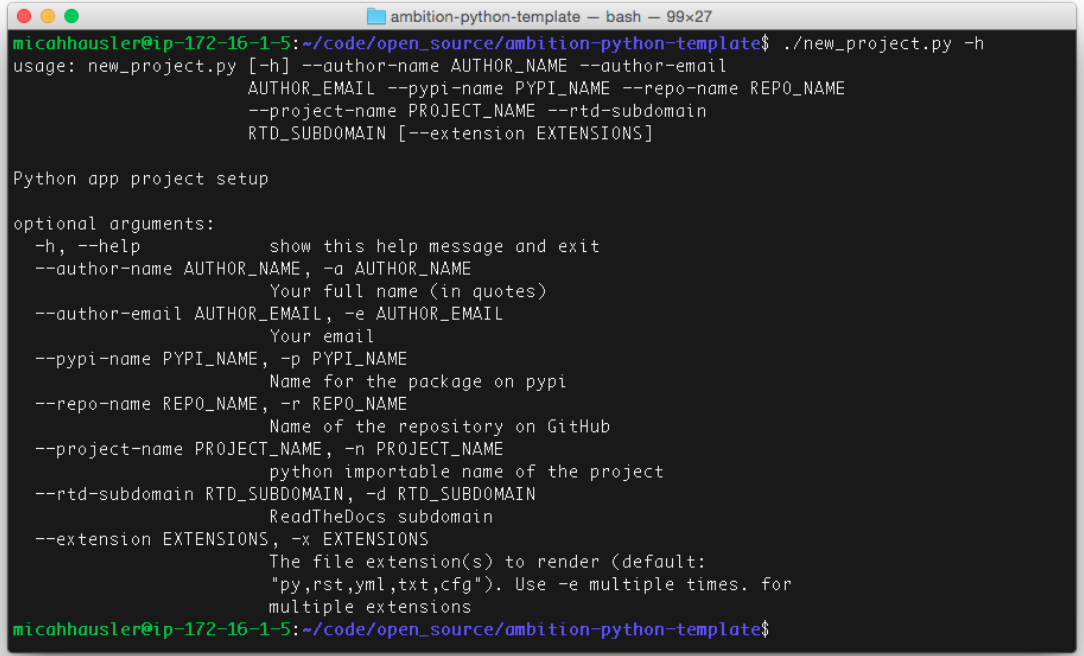
rtfd.org


Usage Examples — django
Micah
django-query-builder.readthedocs.org/en/latest/usage_examples.html
django-query-builder
Search docs
django-query-builder Documentation
Installation
Usage Examples
Selecting records as dictionaries
Selecting records as models
Sorting
Limit and Offset
Filtering
Joins
Date functions
Window functions
Inner queries
Connection Setup
Reference Material
Query API documentation
Field API documentation
Tables API documentation
Contributing
Release Notes
Read the Docs
v: latest
Docs » Usage Examples
Edit on GitHub
Usage Examples
Selecting records as dictionaries
Select all fields from a table:
from querybuilder.query import Query
query = Query().from_table('account')
query.select()
# [{"id": 1, "name": "Person 1"}, {"id": 2, "name": "Person 2"}]
query.get_sql()
# "SELECT account.* FROM account"
id
name
1
Person 1
2
Person 2
Alias a table and fields:
from querybuilder.query import Query
query = Query().from_table(
table={
'my_table': Account

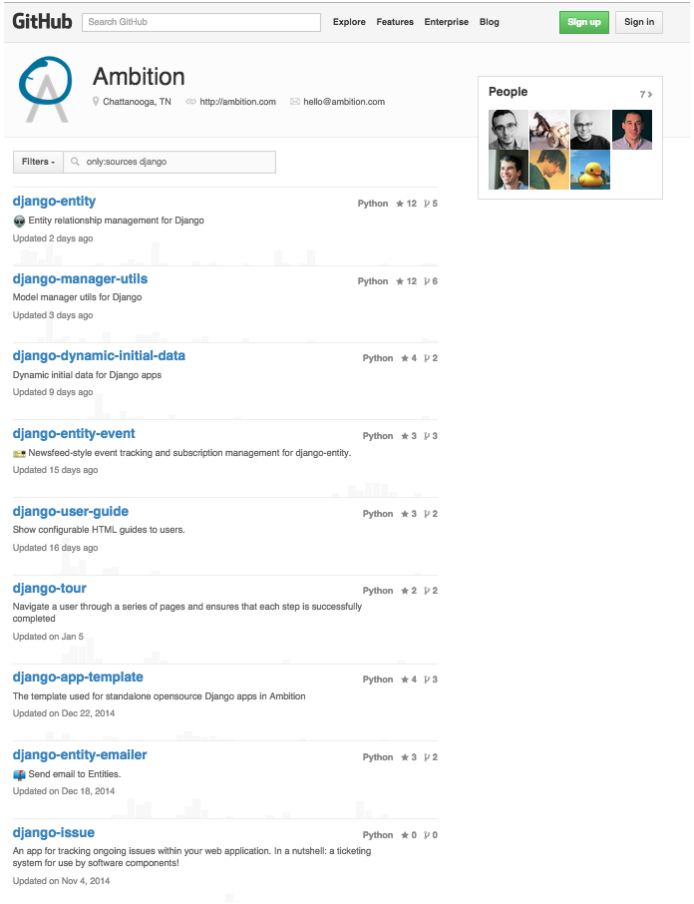
Code Cleanup

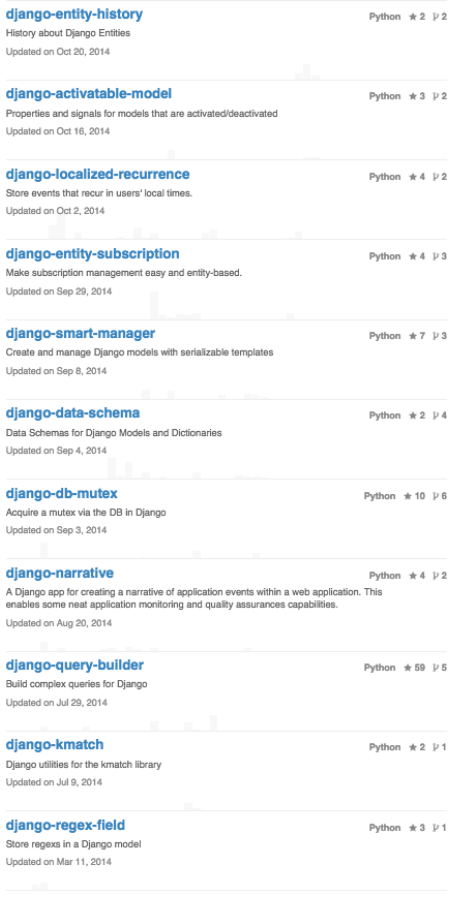
Splitting Up Code

- Lots of apps in one project → lots of external apps
- Made possible by templating python and django apps and projects



Django Apps


GitHub
Search GitHub
Explore
Features
Enterprise
Blog
Sign up
Sign in
Ambition
Chattanooga, TN
http://ambition.com
hello@ambition.com
People
7 >
Filters
only:sources django
django-entity
Entity relationship management for Django
Python ★ 12 ⑂ 5
Updated 2 days ago
django-manager-utils
Model manager utils for Django
Python ★ 12 ⑂ 6
Updated 3 days ago
django-dynamic-initial-data
Dynamic initial data for Django apps
Python ★ 4 ⑂ 2
Updated 9 days ago
django-entity-event
Newsfeed-style event tracking and subscription management for django-entity.
Python ★ 3 ⑂ 3
Updated 15 days ago
django-user-guide
Show configurable HTML guides to users.
Python ★ 3 ⑂ 2
Updated 16 days ago
django-tour
Navigate a user through a series of pages and ensures that each step is successfully completed
Python ★ 2 ⑂ 2
Updated on Jan 5
django-app-template
The template used for standalone opensource Django apps in Ambition
Python ★ 4 ⑂ 3
Updated on Dec 22, 2014
django-entity-emailer
Send email to Entities.
Python ★ 3 ⑂ 2
Updated on Dec 18, 2014
django-issue
An app for tracking ongoing issues within your web application. In a nutshell: a ticketing system for use by software components!
Python ★ 0 ⑂ 0
Updated on Nov 4, 2014


django-entity-history
History about Django Entities
Python ★ 2 ⑂ 2
Updated on Oct 20, 2014
django-activatable-model
Properties and signals for models that are activated/deactivated
Python ★ 3 ⑂ 2
Updated on Oct 16, 2014
django-localized-recurrence
Store events that recur in users' local times.
Python ★ 4 ⑂ 2
Updated on Oct 2, 2014
django-entity-subscription
Make subscription management easy and entity-based.
Python ★ 4 ⑂ 3
Updated on Sep 29, 2014
django-smart-manager
Create and manage Django models with serializable templates
Python ★ 7 ⑂ 3
Updated on Sep 8, 2014
django-data-schema
Data Schemas for Django Models and Dictionaries
Python ★ 2 ⑂ 4
Updated on Sep 4, 2014
django-db-mutex
Acquire a mutex via the DB in Django
Python ★ 10 ⑂ 6
Updated on Sep 3, 2014
django-narrative
A Django app for creating a narrative of application events within a web application. This enables some neat application monitoring and quality assurances capabilities.
Python ★ 4 ⑂ 2
Updated on Aug 20, 2014
django-query-builder
Build complex queries for Django
Python ★ 59 ⑂ 5
Updated on Jul 29, 2014
django-kmatch
Django utilities for the kmatch library
Python ★ 2 ⑂ 1
Updated on Jul 9, 2014
django-regex-field
Store regexs in a Django model
Python ★ 3 ⑂ 1
Updated on Mar 11, 2014

ambitioninc/generator-ambition
JavaScript ★ 0 ⑂ 1
Yeoman generator for Ambition.
Updated on Apr 8, 2014

ambitioninc/ambition-py-tests-guide
Python ★ 1 ⑂ 1
A guide for writing tests for Ambition python projects
Updated on Jul 22, 2014

ambitioninc/ambition-slack
Python ★ 1 ⑂ 1
Slack integration for Ambition
Updated on Oct 23, 2014

ambitioninc/newrelic-api
Python ★ 7 ⑂ 2
A Python interface to New Relic's API
Updated 24 days ago

ambitioninc/ambition-docs-guide
Python ★ 1 ⑂ 0
A guide for getting spun up on documenting python and reStrcturedText
Updated on Jul 17, 2014

ambitioninc/gclient-service-account-auth
Python ★ 0 ⑂ 2
Easily create an authorized service-object for interacting with google's client APIs using google service-account credentials.
Updated 13 days ago

ambitioninc/pip-conflict-checker
Python ★ 5 ⑂ 2
A tool that checks installed packages against all package requirements to ensure that there are no dependency version conflicts.
Updated on May 29, 2014

ambitioninc/clickjacket
JavaScript ★ 0 ⑂ 1
A simple JavaScript library for click-jacking protection.
Updated on Jun 11, 2014

ambitioninc/container-transform
Python ★ 7 ⑂ 1
Transform fig project configs to ECS
Updated on Dec 29, 2014

ambitioninc/ambition-python-template
Python ★ 1 ⑂ 4
A template for open-source python projects
Updated an hour ago

ambitioninc/flux-tools
JavaScript ★ 2 ⑂ 1
JavaScript helpers for the flux architecture.
Updated on Sep 16, 2014

ambitioninc/react-ui
JavaScript ★ 9 ⑂ 3
A collection of UI components for React.
Updated 10 days ago

ambitioninc/pagerduty-api
Python ★ 3 ⑂ 2
A Python wrapper to PagerDuty's API
Updated 24 days ago

ambitioninc/engineering-blog
CSS ★ 0 ⑂ 2
Ambition Engineering Blog
Updated on Aug 18, 2014

ambitioninc/gagrab
Python ★ 0 ⑂ 1
When want to grab your data from Google Analytics: gagrab it.
Updated on Aug 15, 2014

ambitioninc/kmatch
Python ★ 31 ⑂ 4
A language for filtering, matching, and validating Python dictionaries
Updated on Sep 9, 2014

ambitioninc/fleming
Python ★ 43 ⑂ 5
Python functions for manipulating datetime objects with respect to their time zone
Updated 2 hours ago

Architecting Useful Abstractions

Rob deCarvalho

The Principles of Good Abstractions

Common Things Easy

- Simplify the common

Stable Interfaces / APIs

- Strive for uniformity
- Follow expectations

Loose Coupling

- Independent components
- Connected through well-defined interfaces

Abstractions for Loose Coupling

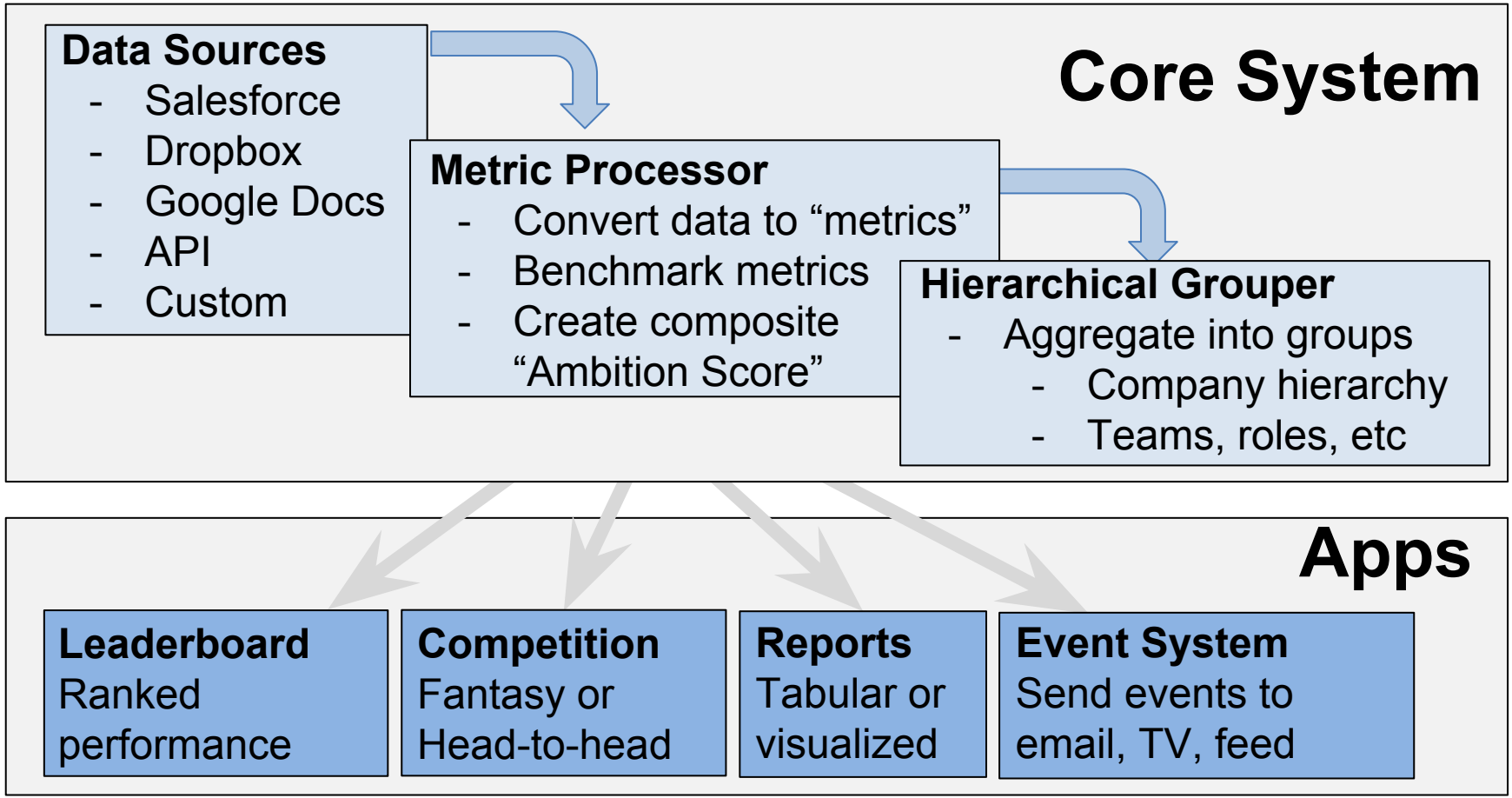
- Loose coupling: Independent blocks of code that interact through well-defined interfaces
- Python: packages, modules, functions, classes
- Django: apps … BUT!
  - Giant mutable shared state-machine: database
  - Changes to database structure often lead to painful code refactoring

Abstractions for Loose Coupling

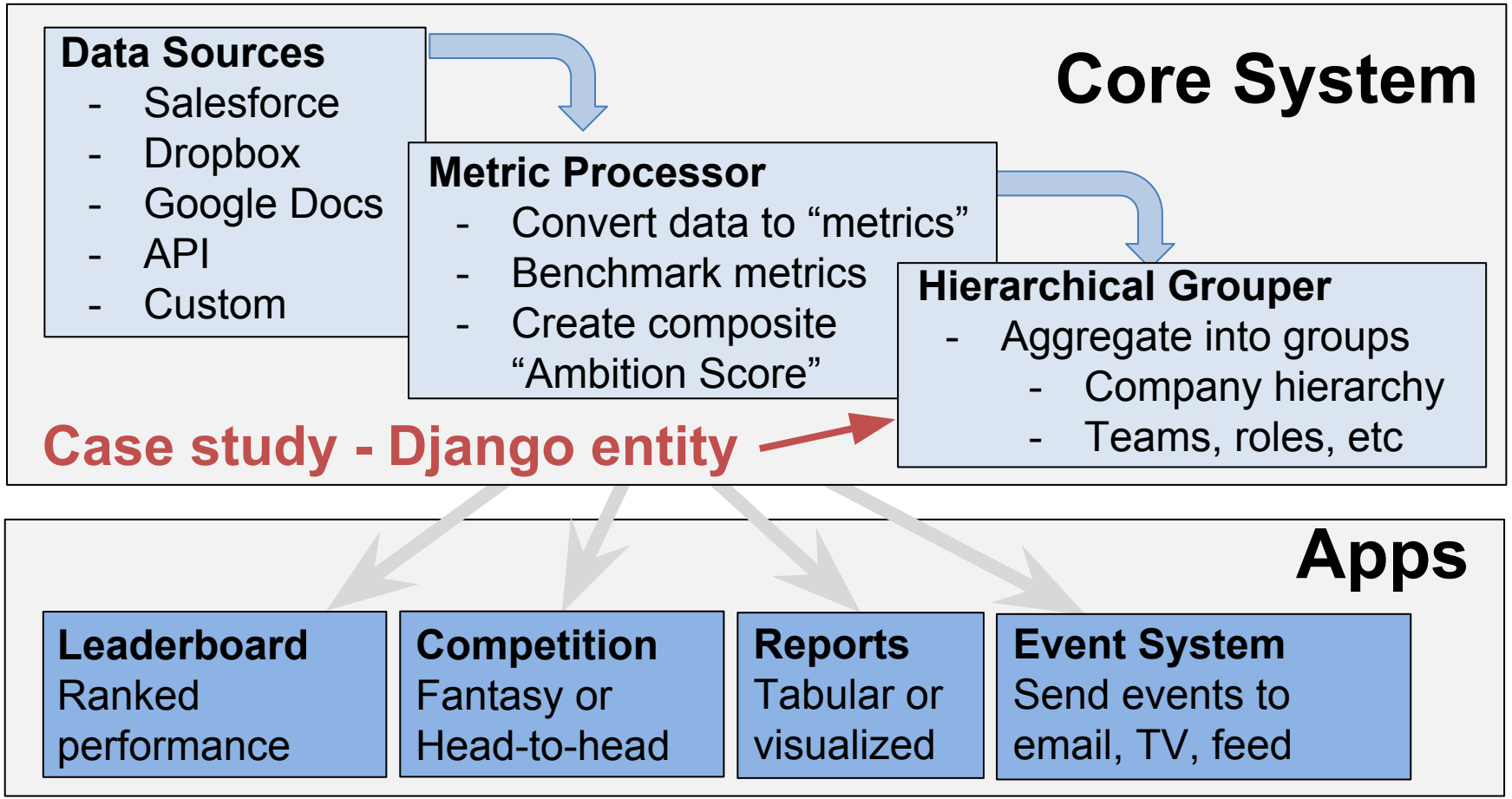
- Loose coupling: Independent blocks of code that interact through well-defined interfaces
- Python: packages, modules, functions, classes
- Django: apps … BUT!
  - Giant mutable shared state machine: database
  - Changes to database structure often lead to painful code refactoring

Examine case-study that helps with this problem

How Ambition Works



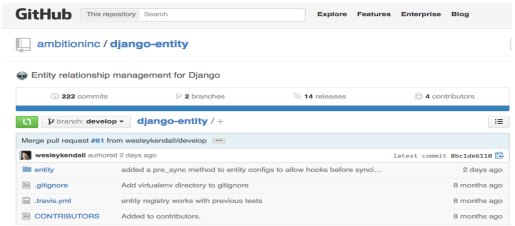
How Ambition Works



Django Entity

Github app from Ambition

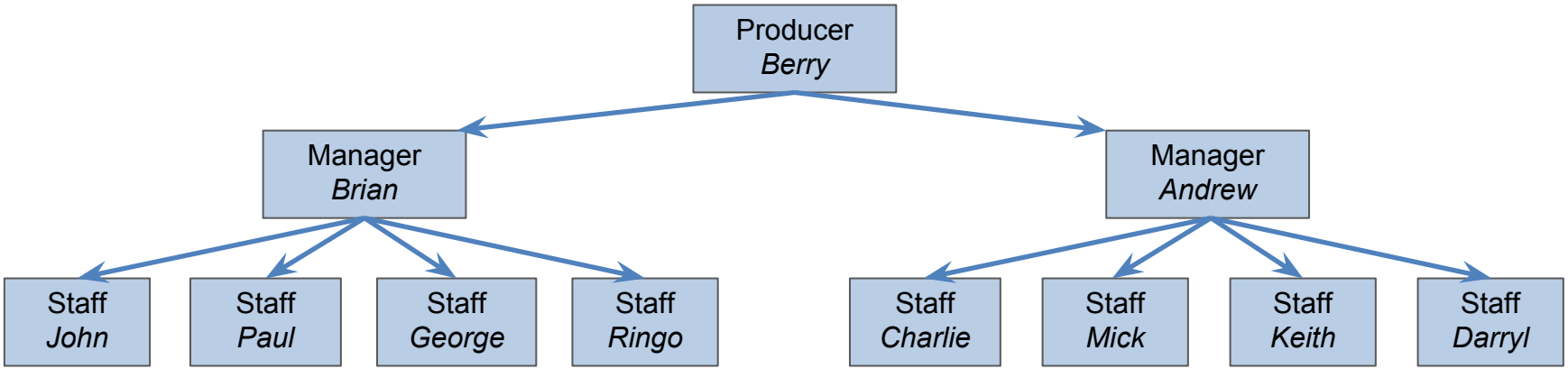
`github.com/ambitioninc/django-entity`



Model-agnostic hierarchical abstraction

- Solves common use case
- Interface has semantics for hierarchies
- Enables more effective decoupling between apps and the database

Concrete Example - Music Studio

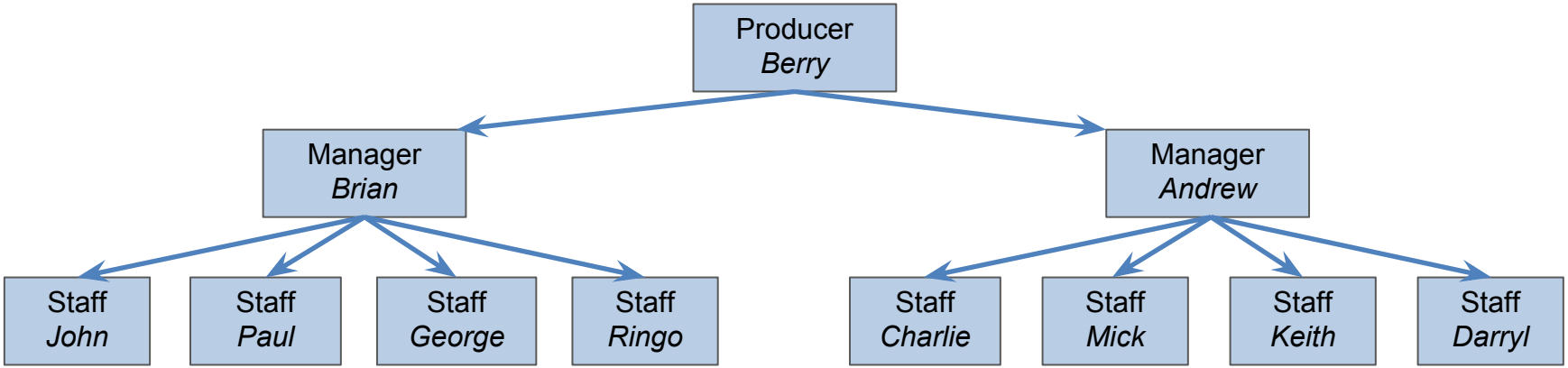



```
class Producer(models.Model):
    name = models.StringField(unique=True)

class Manager(models.Model):
    name = models.StringField(unique=True)
    producer = models.ForeignKey(Producer)

class Staff(models.Model):
    name = models.StringField()
    manager = models.ForeignKey(Manager)
```

Entity Mirroring



Entity Table

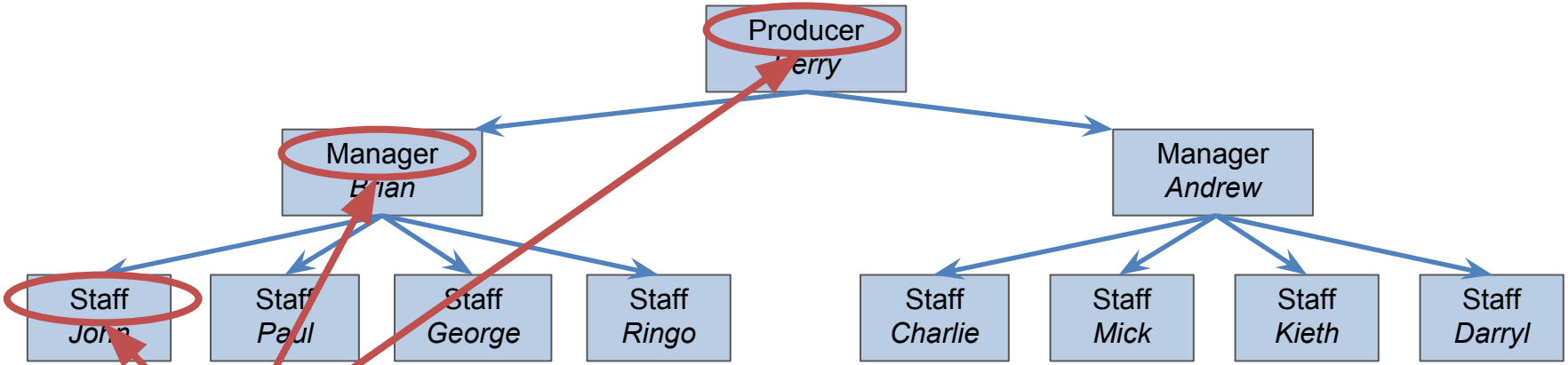
- entity_type: Class of mirrored entity
- entity_id: pk of mirrored entity
- (other stuff)

Entity Relationships Table

- super_entity
- sub_entity
- (other stuff)

Configured with registration process similar to django-admin

Entity Mirroring



Entity Table

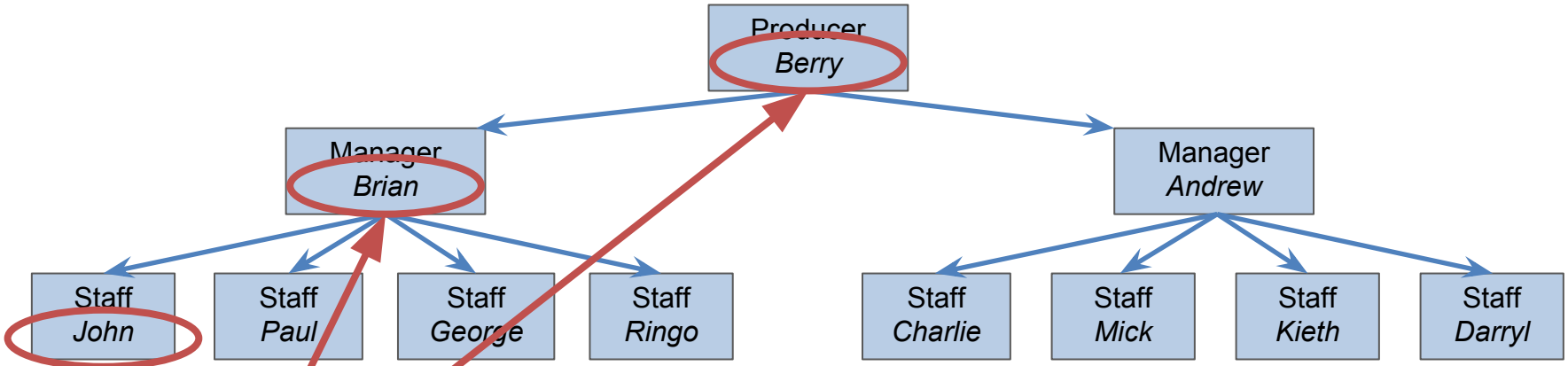
- **entity_type**: Class of mirrored entity
- entity_id: pk of mirrored entity
- (other stuff)

Entity Relationships Table

- super_entity
- sub_entity
- (other stuff)

Configured with registration process similar to django-admin

Entity Mirroring



Entity Table

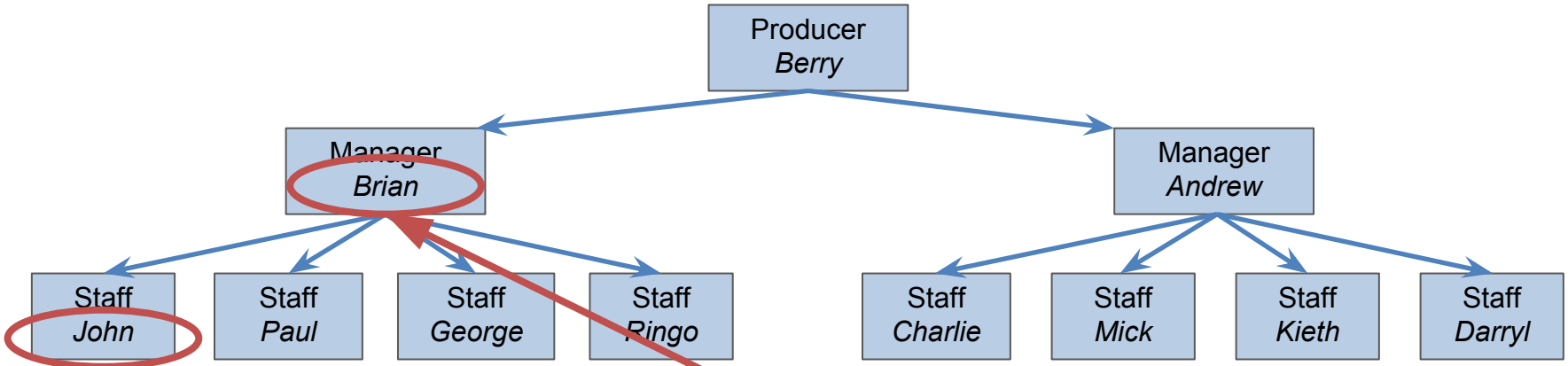
- entity_type: Class of mirrored entity
- **entity_id**: pk of mirrored entity
- (other stuff)

Entity Relationships Table

- super_entity
- sub_entity
- (other stuff)

Configured with registration process similar to django-admin

Entity Mirroring



Entity Table

- entity_type: Class of mirrored entity
- entity_id: pk of mirrored entity
- (other stuff)

Entity Relationships Table

- **super_entity**
- **sub_entity**
- (other stuff)

Configured with registration process similar to django-admin

Adding an Analytics App to Project

models.py

```
from studio.models import Staff

class Rehearsals(models.Model):
    staff = models.ForeignKey(Staff)
    week = models.DateTimeField()
    percent_attended = models.FloatField()
```

stats.py

```
from django.db.models import Avg
import pandas as pd
from activities.models import Rehearsals

by_producer = Rehearsals.objects.all().values(
    'staff__manager__producer').annotate(
    Avg('percent_attended'))

by_manager = Rehearsals.objects.all().values(
    'staff__manager').annotate(
    Avg('percent_attended'))

by_staff = Rehearsals.objects.all().values(
    'staff').annotate(
    Avg('percent_attended'))

df = pd.DataFrame(
    by_producer + by_manager + by_staff)
```

Adding an Analytics App to Project

models.py

```
from studio.models import Staff

class Rehearsals(models.Model):
    staff = models.ForeignKey(Staff)
    week = models.DateTimeField()
    percent_attended = models.FloatField()
```

stats.py

```
from django.db.models import Avg
import pandas as pd
from activities.models import Rehearsals

by_producer = Rehearsals.objects.all().values(
    'staff__manager__producer').annotate(
    Avg('percent_attended'))

by_manager = Rehearsals.objects.all().values(
    'staff__manager').annotate(
    Avg('percent_attended'))

by_staff = Rehearsals.objects.all().values(
    'staff').annotate(
    Avg('percent_attended'))

df = pd.DataFrame(
    by_producer + by_manager + by_staff)
```

Dependencies

- Explicit dependency on studio app
- Changing any studio model means code changes in stats module

Adding an Analytics App to Project

```python
from studio.models import Staff

class Rehearsals(models.Model):
    staff = models.ForeignKey(Staff)
    week = models.DateTimeField()
    percent_attended = models.FloatField()
```

```python
from django.db.models import Avg
import pandas as pd
from activities.models import Rehearsals

by_producer = Rehearsals.objects.all().values(
    'staff__manager__producer').annotate(
    Avg('percent_attended'))

by_manager = Rehearsals.objects.all().values(
    'staff__manager').annotate(
    Avg('percent_attended'))

by_staff = Rehearsals.objects.all().values(
    'staff').annotate(
    Avg('percent_attended'))

df = pd.DataFrame(
    by_producer + by_manager + by_staff)
```

```python
class Rehearsals(models.Model):
    entity = models.ForeignKey(Entity)
    week = models.DateTimeField()
    percent_attended = models.FloatField()
```

Entity decouples apps!

```python
KIND = 'staff'

# aggregate over staff
df1 = pd.DataFrame(
    Rehearsals.objects.filter(
    entity__entity_kind.name=KIND).values(
    'entity').annotate(
    Avg('percent_attended')))

# find relationships of staff to superiors
df2 = pd.DataFrame(
    EntityRelationships.objects.filter(
    sub_entity__entity_kind.name=KIND).values(
    'super_entity', 'sub_entity'))

# join rehearsal info with superior info
df = pd.merge(df1, df1, left_on='entity',
    right_on='sub_entity').group_by(
    'super_entity').mean()
```

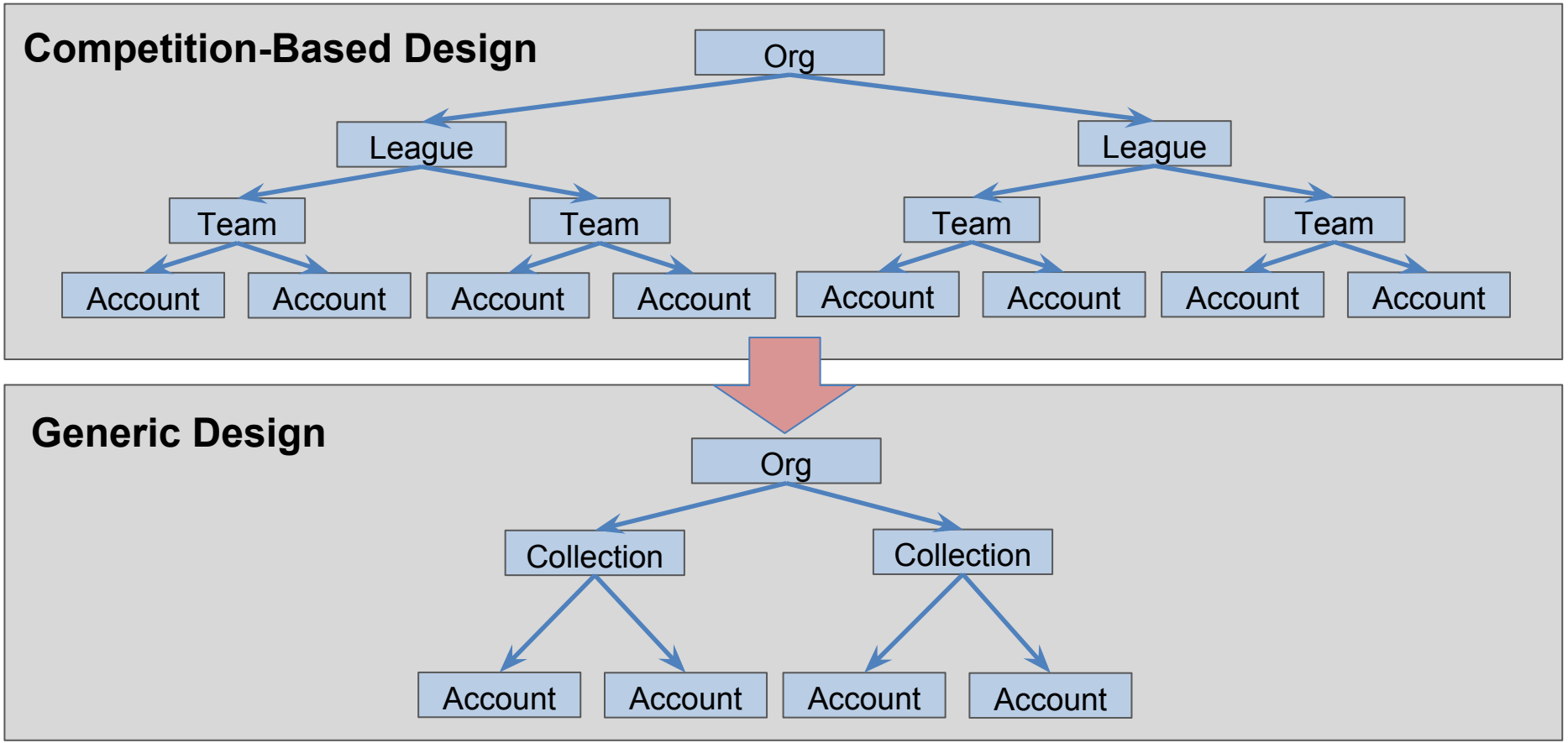
Django Entity Helped Ambition Scale

Over 25 Ambition apps use Django Entity

account_management, ambition-business-calendar, ambition-competition, ambition-headlines, ambition-integration, ambition-notification, ambition-onboarding, ambition-trigger, calendar_management, collection_management, django-animal, django-entity-emailer, django-entity-event, django-entity-history, django-entity-subscription, django-lucy, fantasy, fantasy_management, insights, leaderboard, transport, trigger, tv, tv_management, ...

Django Entity is just an example.
We employ similar strategies for various levels of abstraction.

Huge Refactor Just Deployed



Summary

- Three properties of good abstractions
- Loose coupling is hard for database apps
- Django entity helps us decouple our apps
- The proof is in the pudding! Our enormous refactor went very smoothly

Front End

Jeff McRiffey

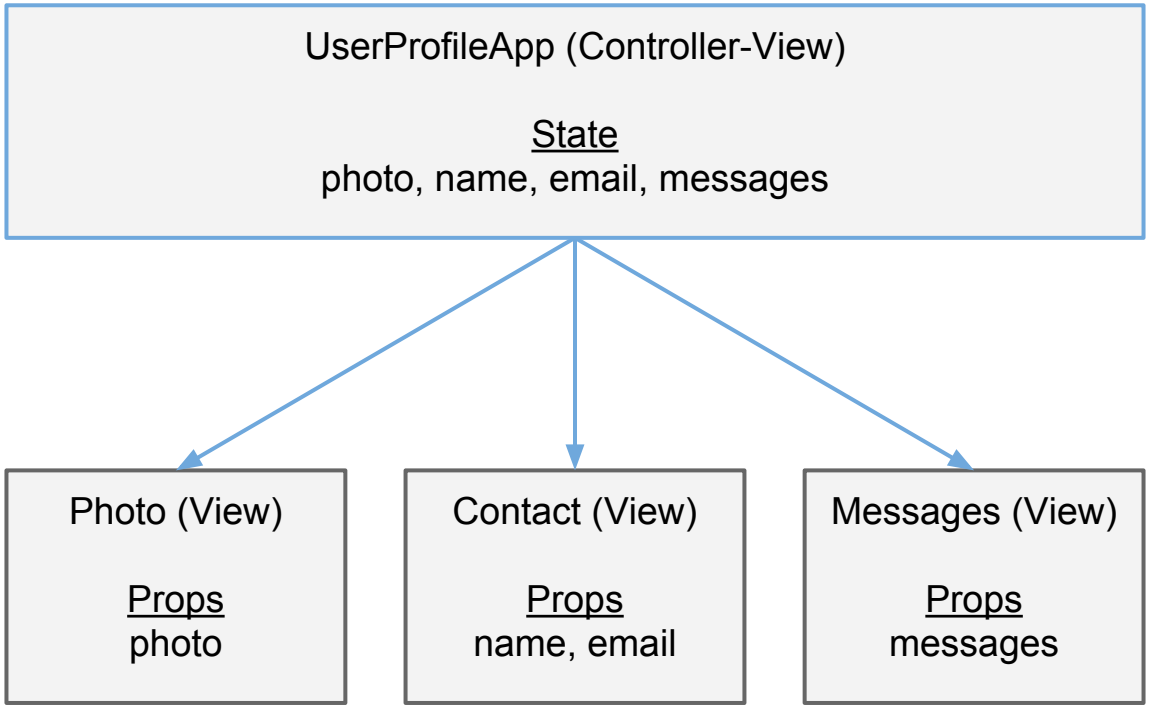
Front End Practices

- Back end practices make much life easier
  - Useful abstractions => stable APIs
  - Stable APIs => Predictable UI
- Applies to front end too
  - Code quality (coverage + static analysis)
  - Loose coupling in JS/CSS (es6 + stylus)
  - Good abstractions (React + Flux)

React

- Components
  - Declarative view layer
  - We distinguish views and controller-views
- State
  - Mutable - managed by component
- Props
  - Immutable - managed by parent

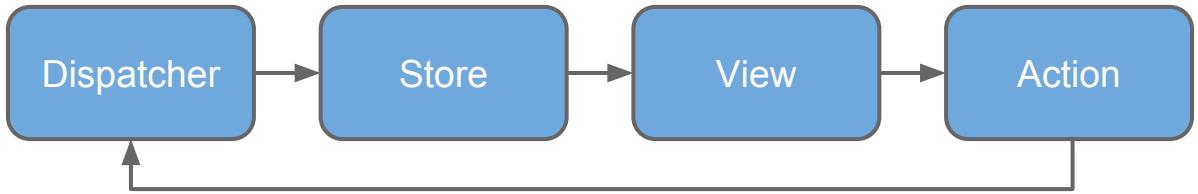
React


UserProfileApp (Controller-View)
State
photo, name, email, messages
Photo (View)
Props
photo
Contact (View)
Props
name, email
Messages (View)
Props
messages

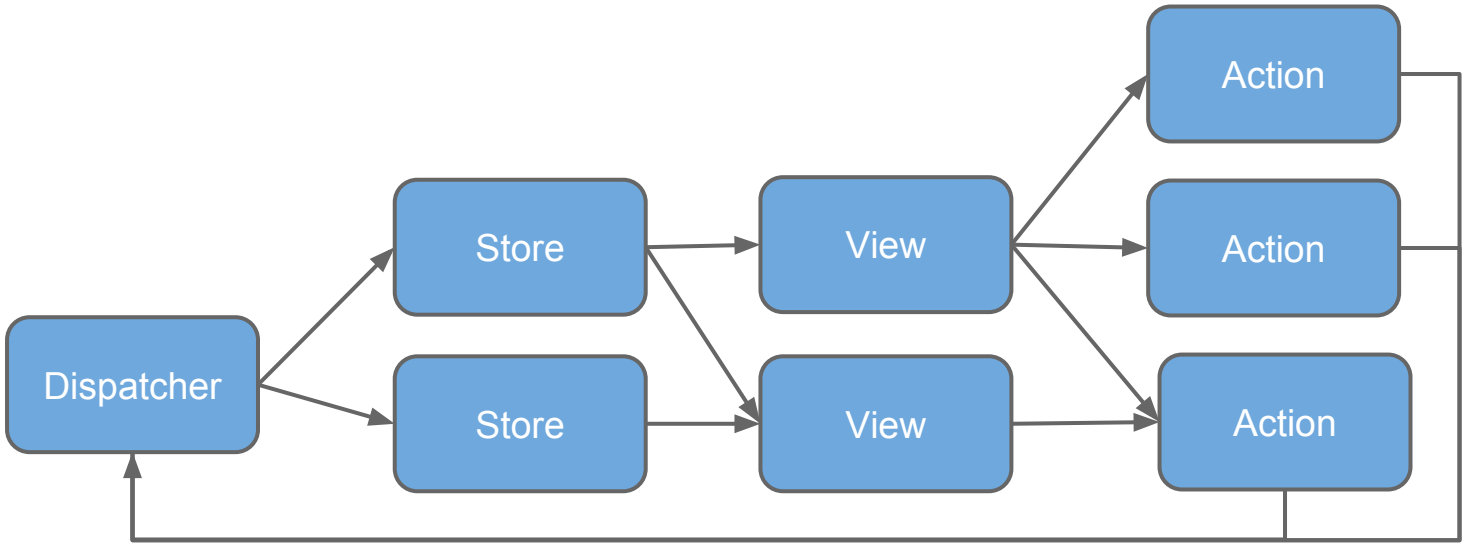
Flux

- Data flow
  - Unidirectional, reliable source of truth
- Dispatcher
  - Coordinates communication
- Stores
  - Holds state
- Actions
  - Changes state

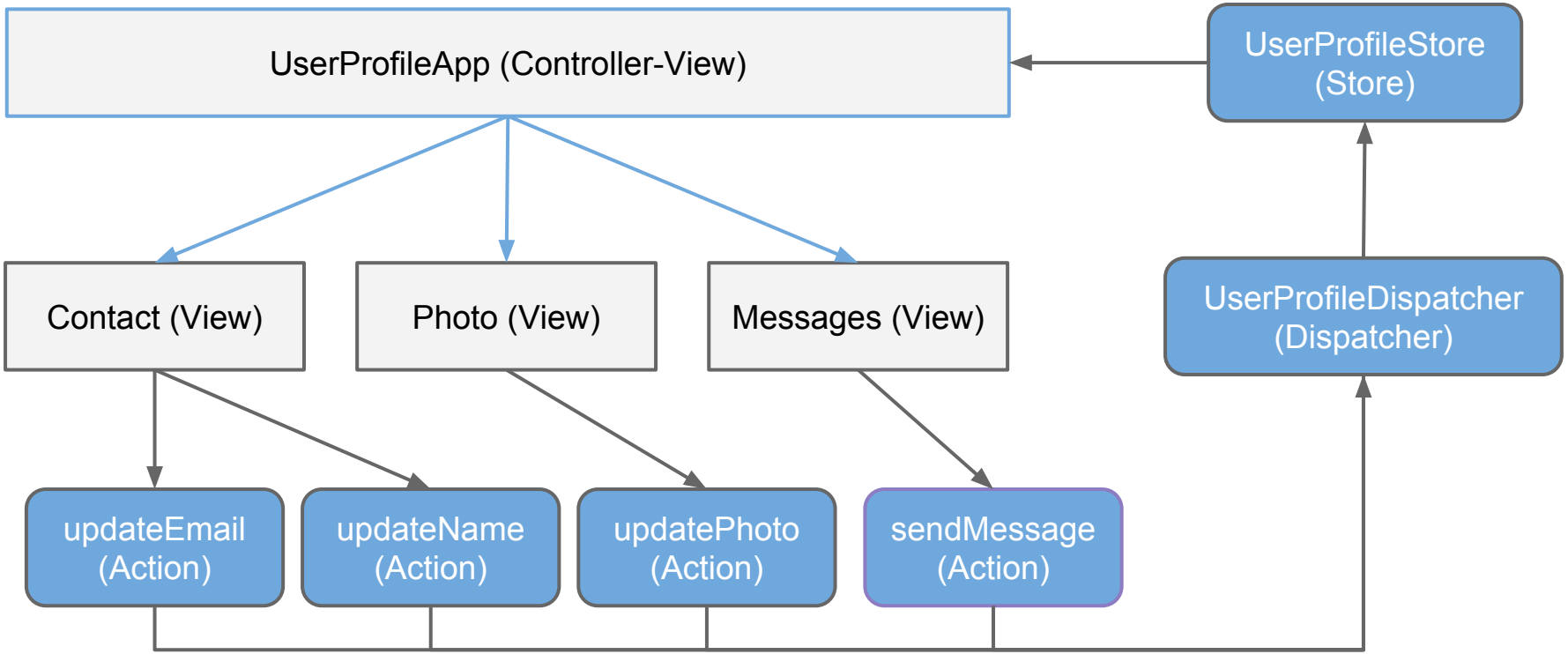
Flux



Flux



React + Flux App Example



Conclusion

Don’t ever degrade coding practices just to ship code.

1. Automate best practices. Reduces energy barriers to do the right thing.
2. When seeing patterns, develop an abstraction.
3. Maintain these practices on the UI and treat it as a first-class citizen. We did this with React.

Resources

- Our open source apps:
  - Django Entity - http://github.com/ambitoninc/django-entity
  - React UI - http://github.com/ambitioninc/react-ui
  - All of them - http://github.com/ambitioninc/
- Static Analysis
  - Flake8 - http://flake8.readthedocs.org/en/latest/
  - Pylint - http://www.pylint.org/
- Continuous Integration
  - TravisCI http://travisci.org/
  - CircleCI https://circleci.com/
- Documentation
  - RTD - https://readthedocs.org/
  - MkDocs - http://www.mkdocs.org/
  - Sphinx - http://sphinx-doc.org/index.html
- Google Python Style Guide - https://google-styleguide.googlecode.com/svn/trunk/pyguide.html
- Ambition Project Templates (MIT Licensed)
  - https://github.com/ambitioninc/ambition-python-template
  - https://github.com/ambitioninc/django-app-template