Django vs Flask

DjangoCon 2017, Spokane WA

slides: bit.ly/djangocon-flask

David "DB" Baumgold
@singingwolfboy

Shameless Plug

Hi, I'm DB!

- Freelance web developer
- Corporate trainer
- Serial conference presenter
- Friendly & helpful

Hire me: davidbaumgold.com

Let's talk about Python & web dev.

ZOPE

Plone®

django

pyramid

TURBO GEARS

CherryPy

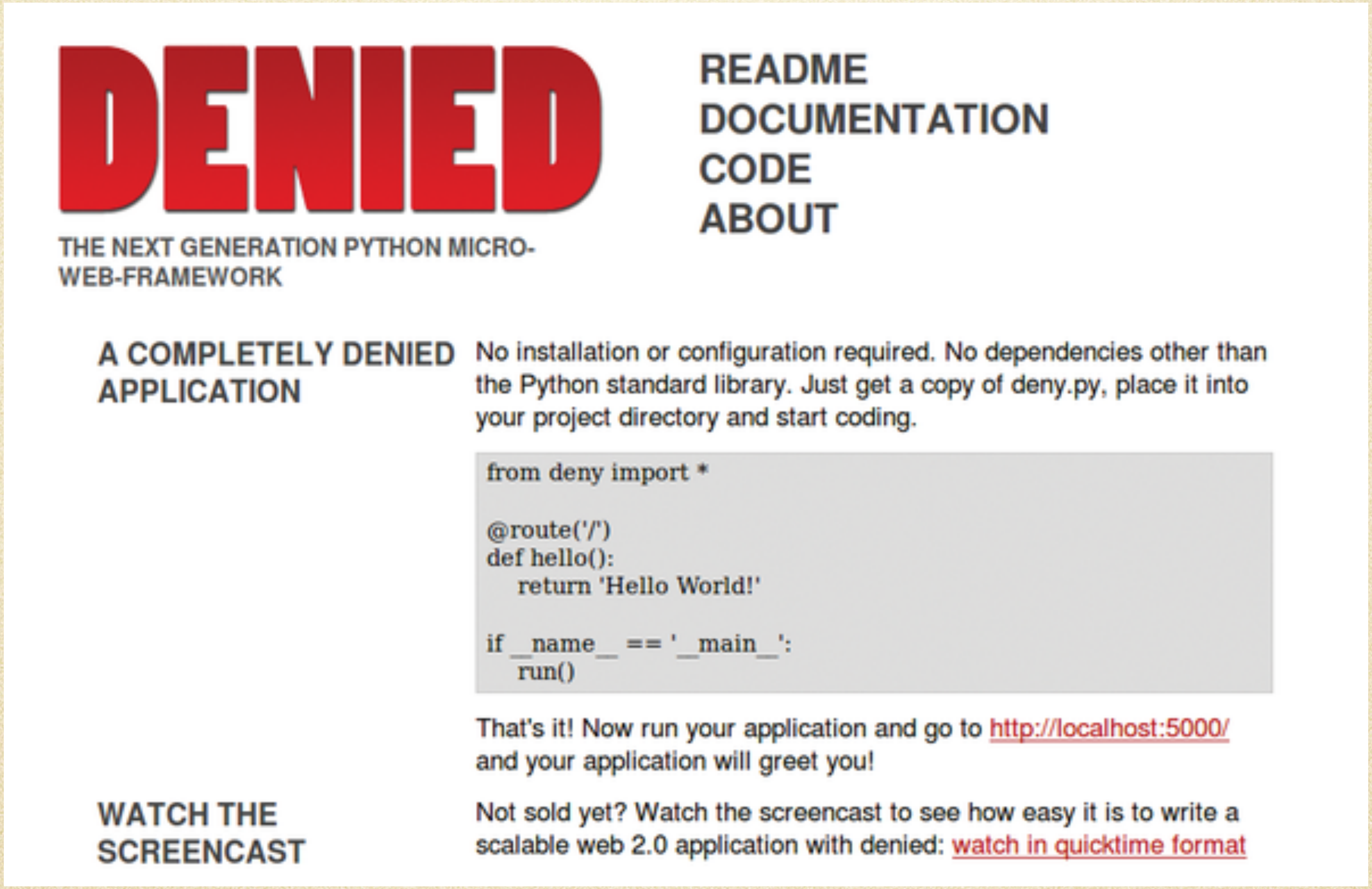
Bottle

WEB2PY

Flask

web development,
one drop at a time

Surprising Beginnings



DENIED

THE NEXT GENERATION PYTHON MICRO-WEB-FRAMEWORK

README
DOCUMENTATION
CODE
ABOUT

A COMPLETELY DENIED APPLICATION

No installation or configuration required. No dependencies other than the Python standard library. Just get a copy of deny.py, place it into your project directory and start coding.

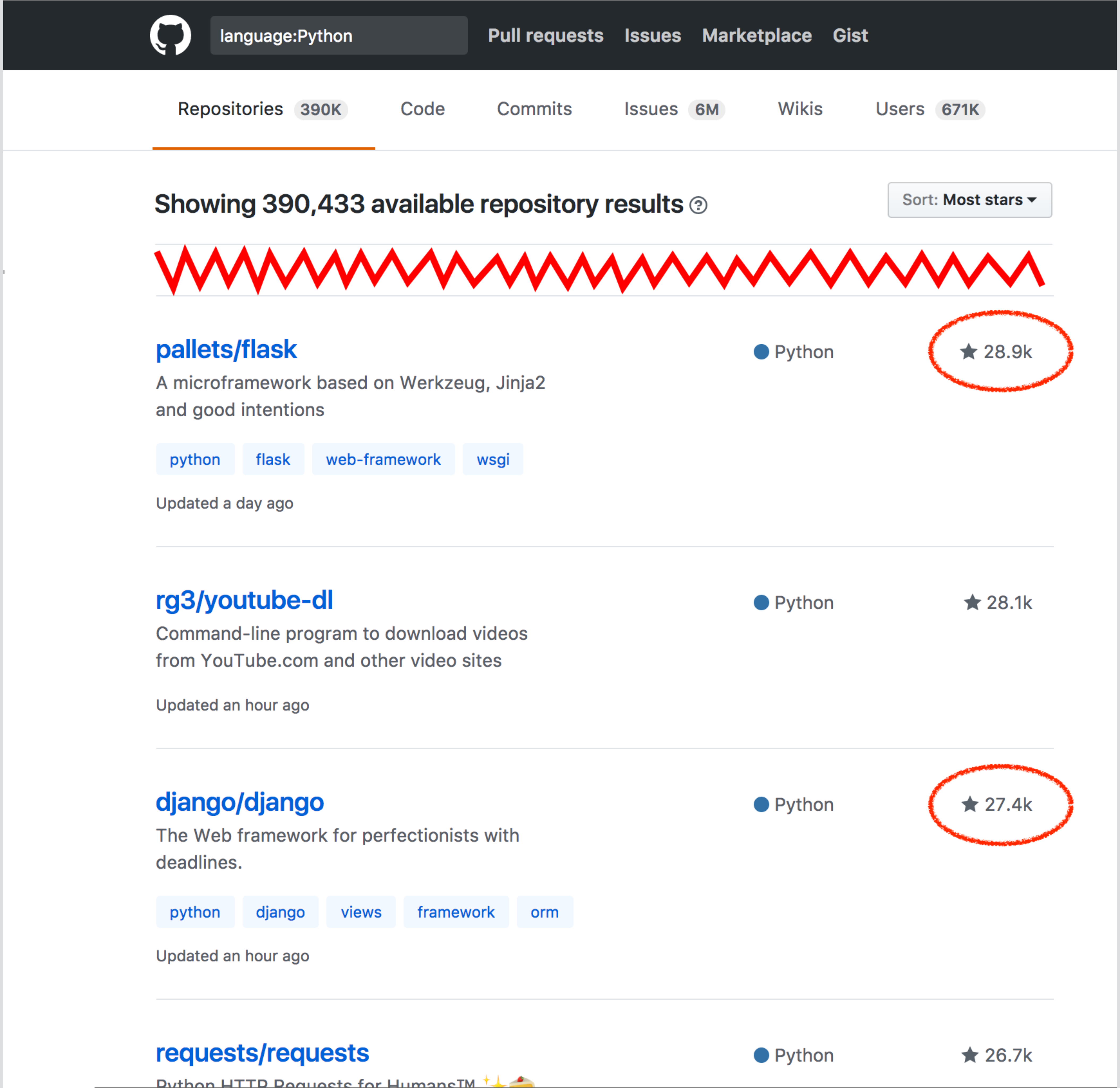
```
from deny import *

@route('/')
def hello():
    return 'Hello World!'

if __name__ == '__main__':
    run()
```

That's it! Now run your application and go to http://localhost:5000/ and your application will greet you!

WATCH THE SCREENCAST

Not sold yet? Watch the screencast to see how easy it is to write a scalable web 2.0 application with denied: watch in quicktime format

Flask started in 2010,
as an April Fools Day joke!



Showing 390,433 available repository results

pallets/flask
A microframework based on Werkzeug, Jinja2 and good intentions
python flask web-framework wsgi
Updated a day ago
Python ★ 28.9k

rg3/youtube-dl
Command-line program to download videos from YouTube.com and other video sites
Updated an hour ago
Python ★ 28.1k

django/django
The Web framework for perfectionists with deadlines.
python django views framework orm
Updated an hour ago
Python ★ 27.4k

requests/requests
Python HTTP Requests for Humans™
Python ★ 26.7k

Why is Flask so popular?

Is it better than Django?

Django is large.
Flask is small.
Both are good!

Hello World in Flask

Create `hello.py`:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Then run:

```
$ FLASK_APP=hello.py flask run
```

Hello World in Django

Set up your project:

```
$ django-admin startproject project
$ cd project
$ python manage.py startapp hello
```

Edit `project/settings.py`:

```
INSTALLED_APPS = [
    ...
    'hello',
]
```

Hello World in Django

Edit `hello/views.py`:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello World!")
```

Hello World in Django

Edit `project/urls.py`:

```
from django.conf.urls import url
from hello import views

urlpatterns = [
    url(r'^$', views.hello),
]
```

Then run:

```
$ python manage.py runserver
```

Comparison

- Django is more intimidating to beginners than Flask
- Django has a steeper learning curve: settings, regular expressions, etc
- Flask allows single-file projects

Data Models in Django

Defining a model:

```
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField()
```

Data Models in Django

Manipulating data:

```
bp = BlogPost()
bp.title = "DjangoCon"
bp.save()
```

Querying data:

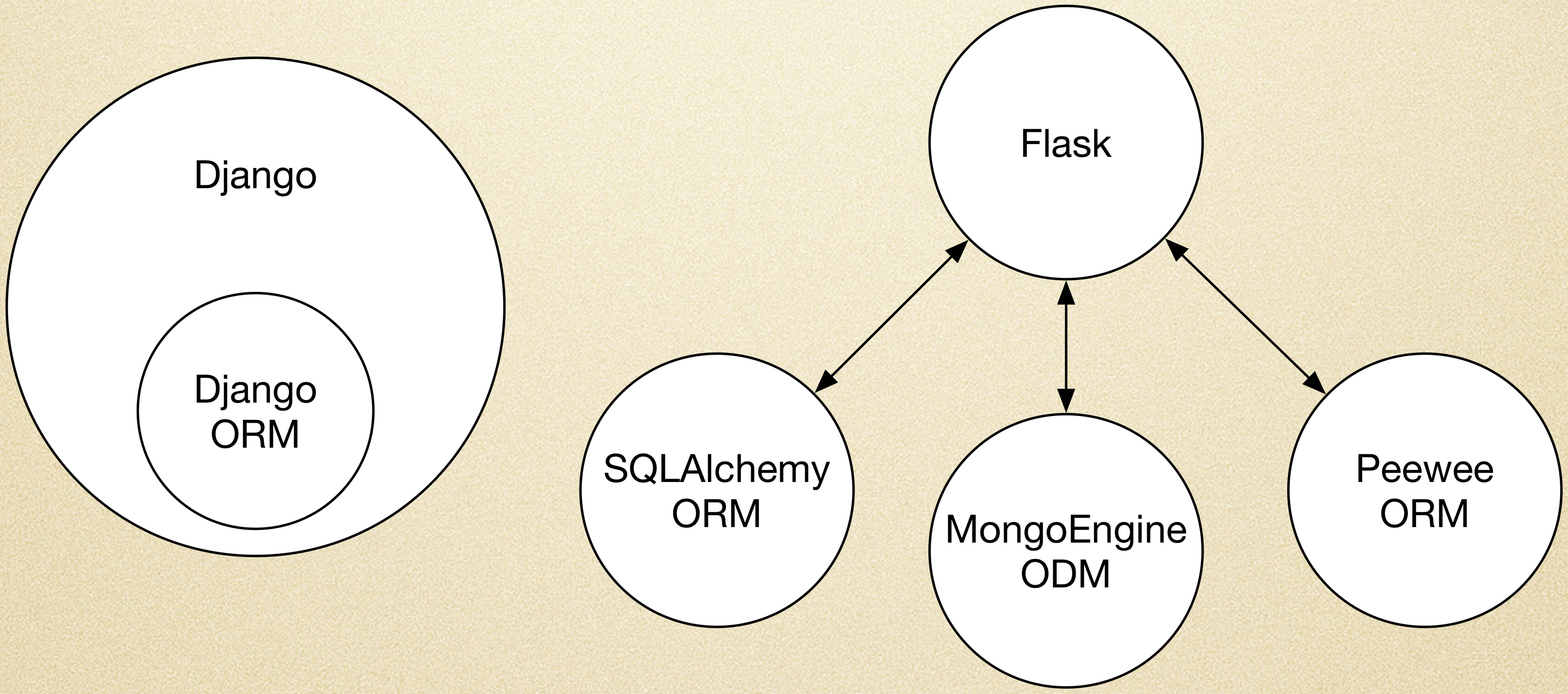
```
BlogPost.objects
        .filter(title="DjangoCon")
        .all()
```

Data Models in Flask

Flask doesn't have data models!

😕

Data Modeling



Flask is Extensible

- Flask is intentionally minimalist.
- Flask includes templating, URL routing, error handling, and a debugger. That's all.
- All other functionality is delegated to extensions. Pick and choose the functionality that you want!

Flask-SQLAlchemy

Install with pip:

```
$ pip install Flask-SQLAlchemy
```

Import and configure:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = \
    'sqlite:////tmp/test.db'
db = SQLAlchemy(app)
```

Flask-SQLAlchemy

Defining a model:

```
class BlogPost(db.Model):
    title = db.Column(db.String(200))
    content = db.Column(db.Text)
    pub_date = db.Column(db.DateTime)
```

SQLAlchemy

```
class BlogPost(db.Model):
    title = db.Column(db.String(200))
    content = db.Column(db.Text)
    pub_date = db.Column(db.DateTime)
```

Django ORM

```
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    pub_date = models.DateTimeField()
```

Flask-SQLAlchemy

Manipulating data:

```
bp = BlogPost()
bp.title = "DjangoCon"
db.session.add(bp)
db.session.commit()
```

Querying data:

```
BlogPost.query
        .filter_by(title="DjangoCon")
        .all()
```

Comparison

- Django's data models are easier to get started: they are built-in to the framework.
- Django assumes that you will use a relational database. If you don't, it will fight you.
- Flask allows more flexibility to choose your data model. More choices mean more potential to screw something up.

Users & Admin

- Most dynamic web applications have user accounts
- Most people want an admin interface to manage these users
- How do Django and Flask compare?

Users in Django

- `django.contrib.auth`
- Built-in & easy
- Swapping user model is possible, but tricky
- Need extra info for users? Make a UserProfile model

Admin in Django

- `django.contrib.admin`
- Built-in & easy
- Highly customizable
- Fine-grained permission system

Users in Flask

- Not built-in
- Most people use "Flask-Login" extension: generic, works with any data model

Users in Flask

```
from flask_login import UserMixin

class User(db.Model, UserMixin):
    id = db.Column(
        db.Integer, primary_key=True)
    username = db.Column(
        db.String(255), unique=True)
    password = db.Column(db.String(255))
    active = db.Column(db.Boolean)

    # add whatever columns you want!
```

Users in Flask

```
from flask_login import current_user

@app.route('/')
def index():
    if current_user.is_anonymous:
        return render_template("splash.html")
    else:
        return render_template("user_home.html")
```

Users in Flask

```
from flask_login import login_required

@app.route('/settings')
@login_required
def settings():
    return render_template("settings.html")
```

If not logged in: "403 Forbidden"

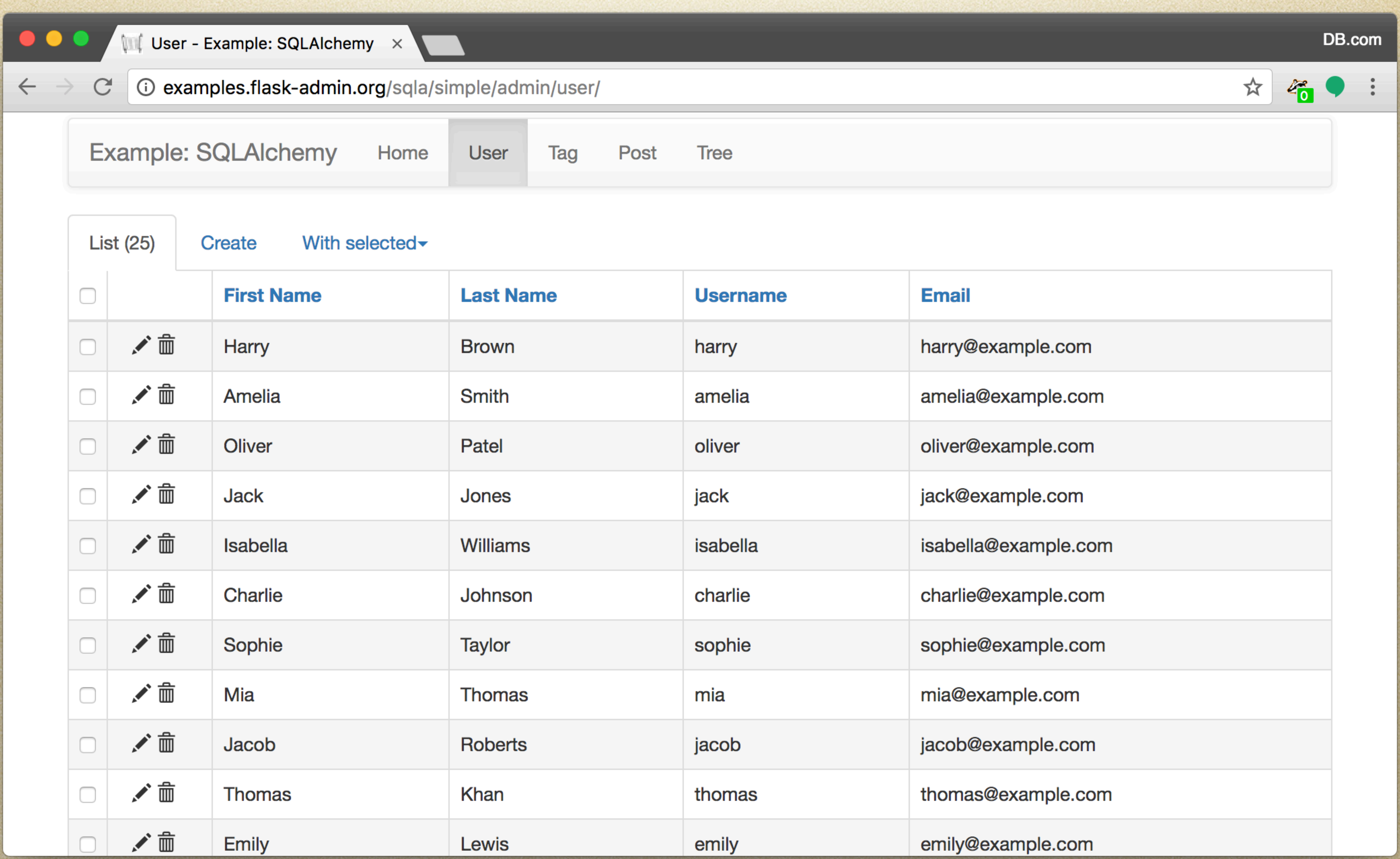
User Permissions in Flask

- "Flask-Principal" extension provides fine-grained permissions
- Designed to work with or without Flask-Login
- Similar to Django's user permissions system

Admin in Flask

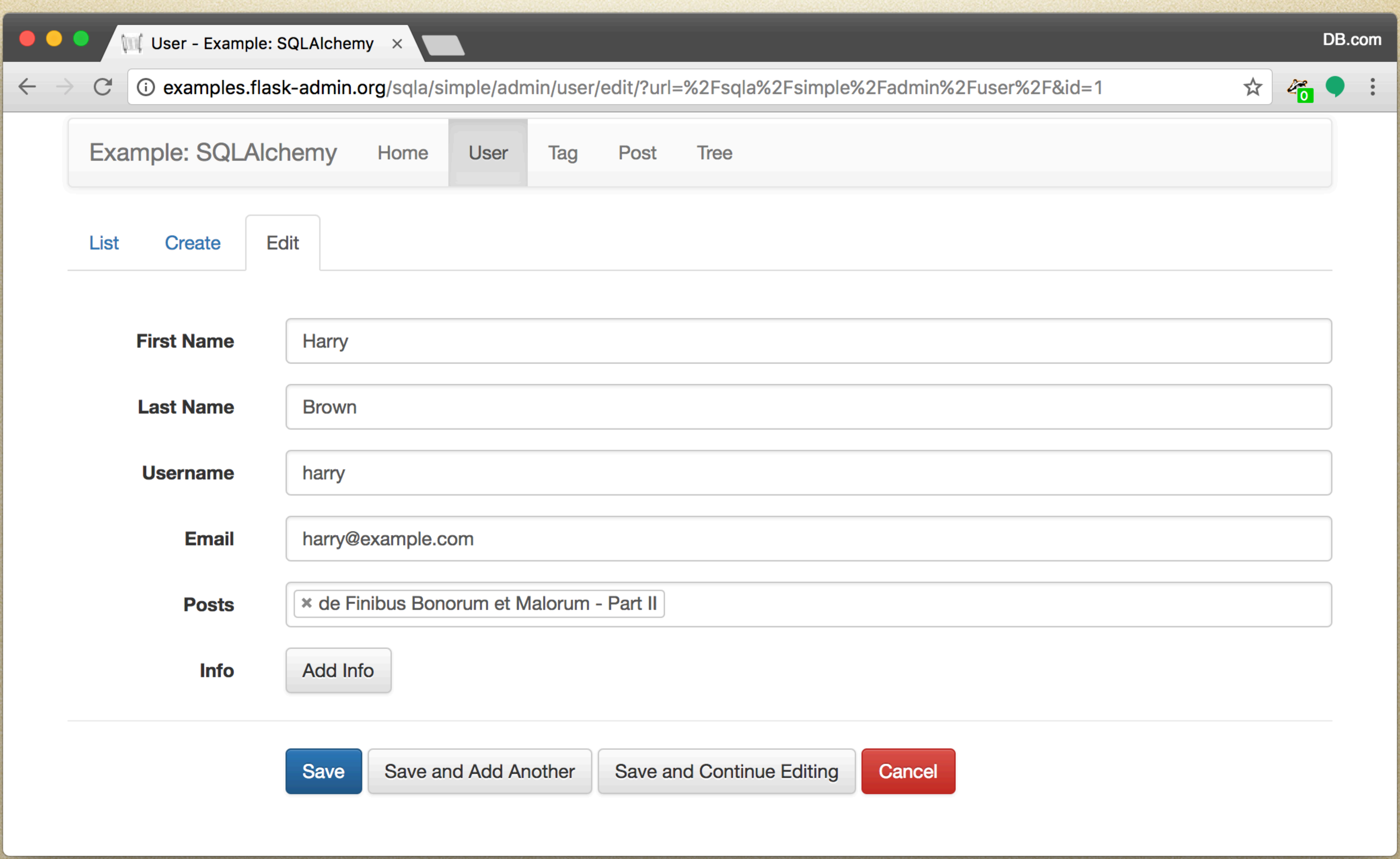
- Most people use "Flask-Admin" extension
- Highly customizable Bootstrap themes
- Works with SQLAlchemy, MongoEngine, or Peewee
- Designed to work with or without Flask-Login and/or Flask-Principal

Admin in Flask


User - Example: SQLAlchemy
DB.com
examples.flask-admin.org/sqla/simple/admin/user/
Example: SQLAlchemy
Home
User
Tag
Post
Tree
List (25)
Create
With selected

		First Name	Last Name	Username	Email
		Harry	Brown	harry	harry@example.com
		Amelia	Smith	amelia	amelia@example.com
		Oliver	Patel	oliver	oliver@example.com
		Jack	Jones	jack	jack@example.com
		Isabella	Williams	isabella	isabella@example.com
		Charlie	Johnson	charlie	charlie@example.com
		Sophie	Taylor	sophie	sophie@example.com
		Mia	Thomas	mia	mia@example.com
		Jacob	Roberts	jacob	jacob@example.com
		Thomas	Khan	thomas	thomas@example.com
		Emily	Lewis	emily	emily@example.com

Admin in Flask


User - Example: SQLAlchemy
DB.com
examples.flask-admin.org/sqla/simple/admin/user/edit/?url=%2Fsqla%2Fsimple%2Fadmin%2Fuser%2F&id=1
Example: SQLAlchemy
Home
User
Tag
Post
Tree
List
Create
Edit
Add Info
Save
Save and Add Another
Save and Continue Editing
Cancel

First Name	Harry
Last Name	Brown
Username	harry
Email	harry@example.com
Posts	× de Finibus Bonorum et Malorum - Part II
Info	Add Info

Flask-Security

- Since many people use the same set of extensions, "Flask-Security" wraps them all up into a single package
- User model, permissions, admin, login forms, password reset emails…
- Works with SQLAlchemy, MongoEngine, or Peewee

Comparison

- Django's user framework & admin are built-in, and work well
- Flask requires multiple extensions working together: steeper learning curve (but Flask-Security makes this easier)
- Off-the-shelf vs extensive customization

Reusable Apps

- Reusable apps can help organize and simplify large codebases
- All code related to one concept lives in one place
- Shared libraries to handle common tasks
  Example: user registration logic
- How do Django and Flask compare?

Apps in Django

- `settings.INSTALLED_APPS`
- Django Packages (djangopackages.org)
- Many packages available; hard to know which are good to use
- Hard to organize an existing project into multiple apps

Blueprints in Flask

- Not quite the same as an app: blueprints are instructions for how to extend an existing app
- Can be applied multiple times to the same app in different ways
- Optional, but recommended for larger Flask projects
- Familiar syntax, easy to get started

Blueprints in Flask

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Blueprints in Flask

```
from flask import Blueprint
hello_bp = Blueprint('hello', __name__)

@hello_bp.route("/")
def hello():
    return "Hello World!"
```

Blueprints in Flask

```
from flask import Flask
from yourapp.hello import hello_bp

app = Flask(__name__)
app.register_blueprint(hello_bp)
```

Comparison

- Django apps are more comprehensive, more numerous — but also more complex
- Flask blueprints are simpler, easier to integrate into a project

Building APIs

- APIs are increasingly common for web applications
- APIs often require different patterns compared to HTML webpages
- How do Django and Flask compare?

APIs in Django

- Django REST Framework. Just use it.
- Authentication policies, serializers, extensive documentation, testing tools… it's all included
- Multi-layered abstractions

APIs in Flask

- Multiple extensions working together
- Serialization: "Marshmallow" module
- Marshmallow ecosystem includes integrations with Flask, SQLAlchemy, MongoEngine, etc

```
from flask_marshmallow import Marshmallow
from flask_login import current_user, login_required
from yourapp.models import User

ma = Marshmallow(app)

class UserSchema(ma.ModelSchema):
    class Meta:
        model = User
        exclude = ['password']

@app.route("/me")
@login_required
def me():
    return UserSchema().jsonify(current_user)

# {"id": 1, "username": "example", "active": true}
```

initialize extension

define serialization schema

convert to JSON

result

Comparison

- Django REST Framework is *amazing*, but is subject to the same restrictions as Django itself (relational database, etc)
- Flask has all the same functionality with much more flexibility, but you have to put it together yourself
- Maybe someday there will be an extension bundle for Flask that is similar to DRF: not yet

Which one do I choose?

Choose Django when...

- You're happy with all the choices Django makes for you:
  Django ORM, Django templates, etc
- You're not doing anything unusual
- You don't care to learn the details of how things work, you just want something that works

Choose Flask when...

- You disagree with one of Django's choices, and want to do things differently
- You have unusual requirements that require custom components
- You want to understand how the plumbing of your application fits together

Any Questions?

Django vs Flask

slides: bit.ly/djangocon-flask

David "DB" Baumgold
@singingwolfboy