pebble

Marc Kirkwood

Developer

Django + JS

“When Worlds Collide”

Topics Covered

- Benefits of creating a front-end in JavaScript for a dynamic site (“web application”)
- Helpful suggestions to do this in a structured way, for a Django based project
- Good ideas for dealing with JavaScript

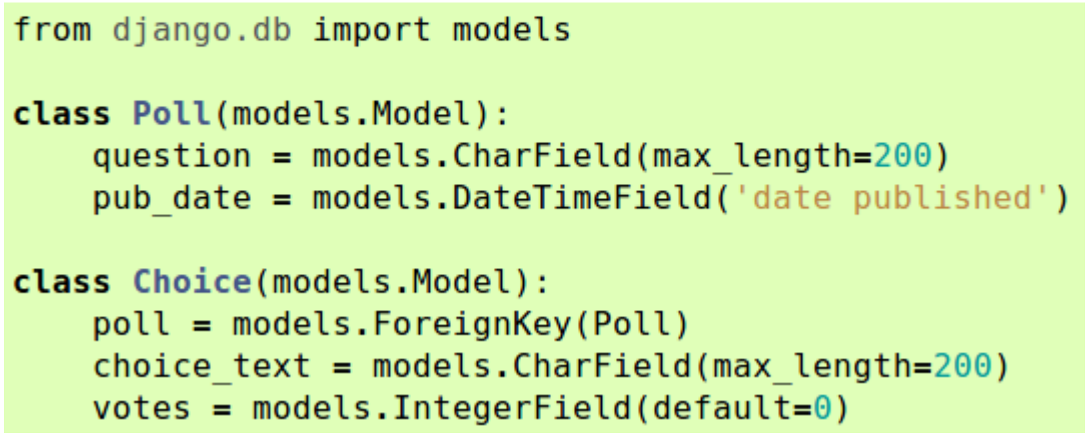
Not Covered

- JavaScript on the server
- 2-way communication (WebSockets)
- Choice of database
- Lots more...

djangoproject.com

- Server-side Python web application framework
- Database agnostic abstraction layer
- Separation of concerns with MVT:
    - Models, Views and Templates
    - (Data, request/response logic and presentation)
- Forms
- URL routing

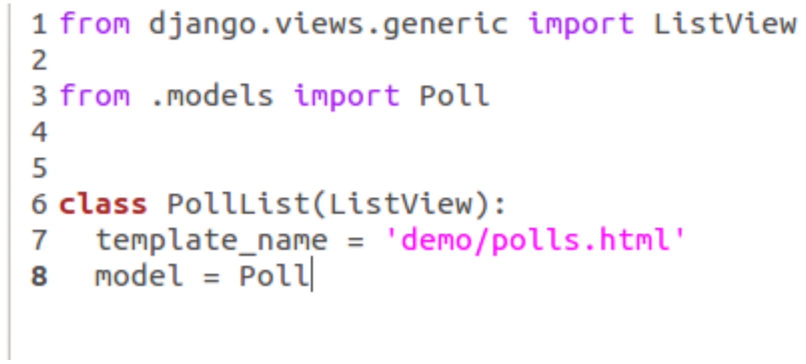
Model



```python
from django.db import models

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

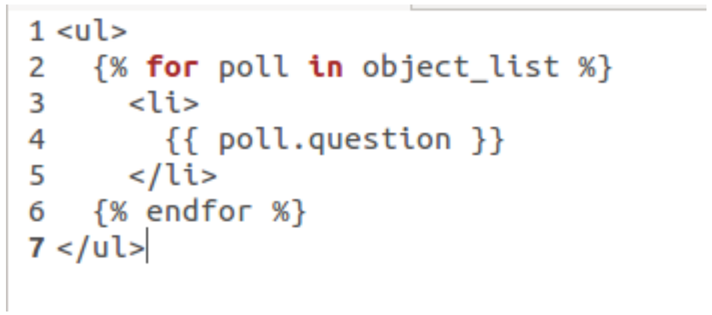
https://docs.djangoproject.com/en/1.5/intro/tutorial01/

View



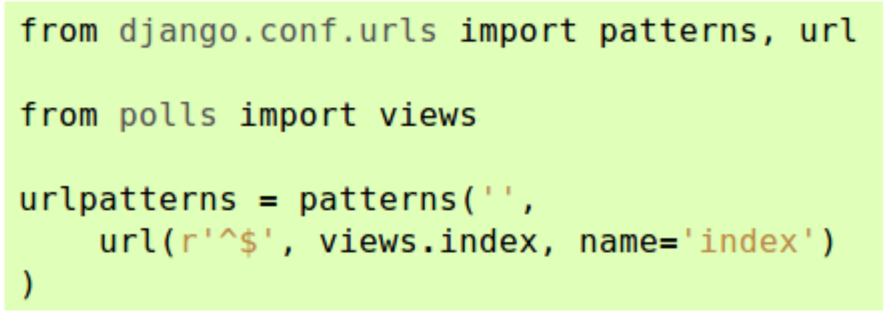
```
from django.views.generic import ListView

from .models import Poll


class PollList(ListView):
  template_name = 'demo/polls.html'
  model = Poll
```

Template



```
<ul>
  {% for poll in object_list %}
    <li>
      {{ poll.question }}
    </li>
  {% endfor %}
</ul>
```

Routing



```python
from django.conf.urls import patterns, url

from polls import views

urlpatterns = patterns('',
    url(r'^$', views.index, name='index')
)
```

https://docs.djangoproject.com/en/1.5/intro/tutorial03/

AJAX: Beginnings

(Asynchronous Javascript And XML)

“XMLHTTP actually began its life out of the Exchange 2000 team.” -- cheers, Microsoft!

http://www.alexhopmann.com/xmlhttp.htm

- Outlook Web Access
- GMail

Don’t Break Your Back

- Modern browsers have fast JS interpreters that can handle front-end processing for lots of data
- Avoid waiting on a server round-trip for the results of another DB query, where possible
  - Scalability under load
  - Responsiveness

Talking to the Browser

- Serialisation is needed (e.g. JSON)
- Could just call **json.dumps(data)**, *but…*
    - Data must be a native Python dict (or iterable of dicts), with **serialisable** items
    - Potential for error with custom code for each object type
    - Corresponding **json.loads(data)** deserialisation call *doesn't validate input.*

django-rest-framework.org

- "Safer" conversion to native dicts from complex objects
- Plug-in model serializers
  - Similar style to Django's Forms and ModelForms
- Parse and render nested JSON, YAML, XML…
- Data validation
- Authentication
- Easy RESTful APIs

django REST framework

Have some REST

- “RESTful” apps with simple URI schemes are nice to develop *and* use
- HTTP “verbs” *do something* with a resource
  - GET /pizza/1/
  - POST /pizza/1/ {“base”: 1, “toppings”: [2, 3]}
- A URI is meant to identify a *specific* resource

A Simple API

High level: APIViews are like Django’s Class Based Views

1. Start with a model or queryset
2. Add authentication, stir in access restrictions…
3. Attach the View to a URI
4. Deal with response

restframework.herokuapp.com

Client-side Templates

- Avoid excessive jQuery DOM manipulation: beware of *spaghetti code*!
- Render a template from the source data as it changes

stencil
node-asyncejs
underscore
javascripttemplates
google-closure
django-template
dust.js
weld
jquerymicro-templates
shared-views
handlebarjs
handlebars.js
mustache
haml-js
parrot
milk
jqote2
jade
pure
liquid
eco
kite
dot
ejs
nun

http://engineering.linkedin.com/frontend/client-side-templating-throwdown-mustache-handlebars-dustjs-and-more

App Management

- Pick a JS framework, stick with it!
- **node.js** can help in development
    - Lots of build tools for packaging your apps (search ‘npm’ for easy installation)
    - Compile templates to reduce overhead
    - Minify and concatenate assets (JS, CSS)

Modularisation

- JavaScript has no **import** statement…
- Mitigate this horror; bring back modular development
- **Brunch** automatically wraps JS files in CommonJS modules
  - Painless **require(‘original/path/to/file’)** calls
  - Let the build tool do the work!

Testing

- Test Drive your apps in different browsers!
- Use fixture data that matches expected server responses
- Loads of test runners available as node packages

More Testing

- **Sinon.js** library helps test interactions
    - **fakeServer** for AJAX (“respond with this data *now*”)
    - **useFakeTimers()** -- spoof time passing for forced delays (**setTimeout**)
- **Testem** -- a node-based test runner that opens and manages real browsers under test

I18N?!

- Start with this in mind, for a global audience
- Internationalisation of an app is a lot simpler with translation strings
  - Easy to do for Django templates
  - Harder to do for JS template heavy apps
- **django-statici18n** helps to do this efficiently

Summary

- **Django Rest Framework** for developing a RESTful API that can deal with serialisation
- You can use Django templates for the *shell* of an app, and to pass it initial data (but minimise logic)
- Keep subsequent server responses compact (JSON)
- Use a modern JS app framework
- Build tools are handy!
    - Learn to love client templates
    - Shrink your scripts and stylesheets
    - Modular JS FTW
- Test JS apps thoroughly

Feeling Generous? :)

Great North Run 2014

justgiving.com/gomarc

Thanks!

The National Autistic Society