Scalable Web Application Architecture

(using Django + ExtJS)

Michał Karzyński, PyWaw #27

TALK OUTLINE

1. Application architecture

2. Backend code organization (Django)

3. Frontend code organization (ExtJS)

4. Communication APIs

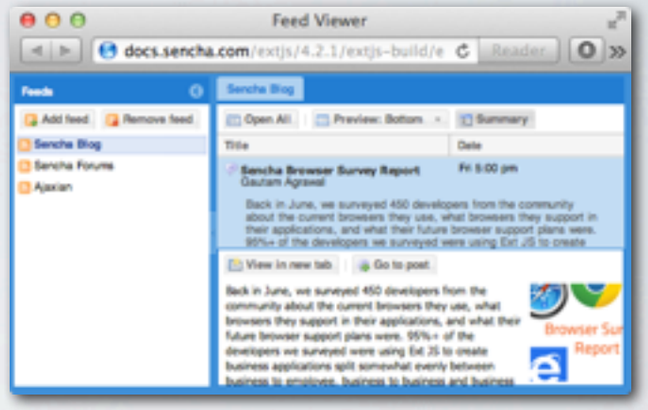
5. Deployment and Scaling

YOURS TRULY

Michał Karzyński

- project manager at Politechnika Gdańska
- freelance developer and consultant
- polyglot, currently: Python and JavaScript
- web developer since 1996
- **@postrational**
- http://michal.karzynski.pl

APPLICATION ARCHITECTURE



Backend

API

Frontend

Server

Cloud

Browser

I/O

Transport

GUI

Python

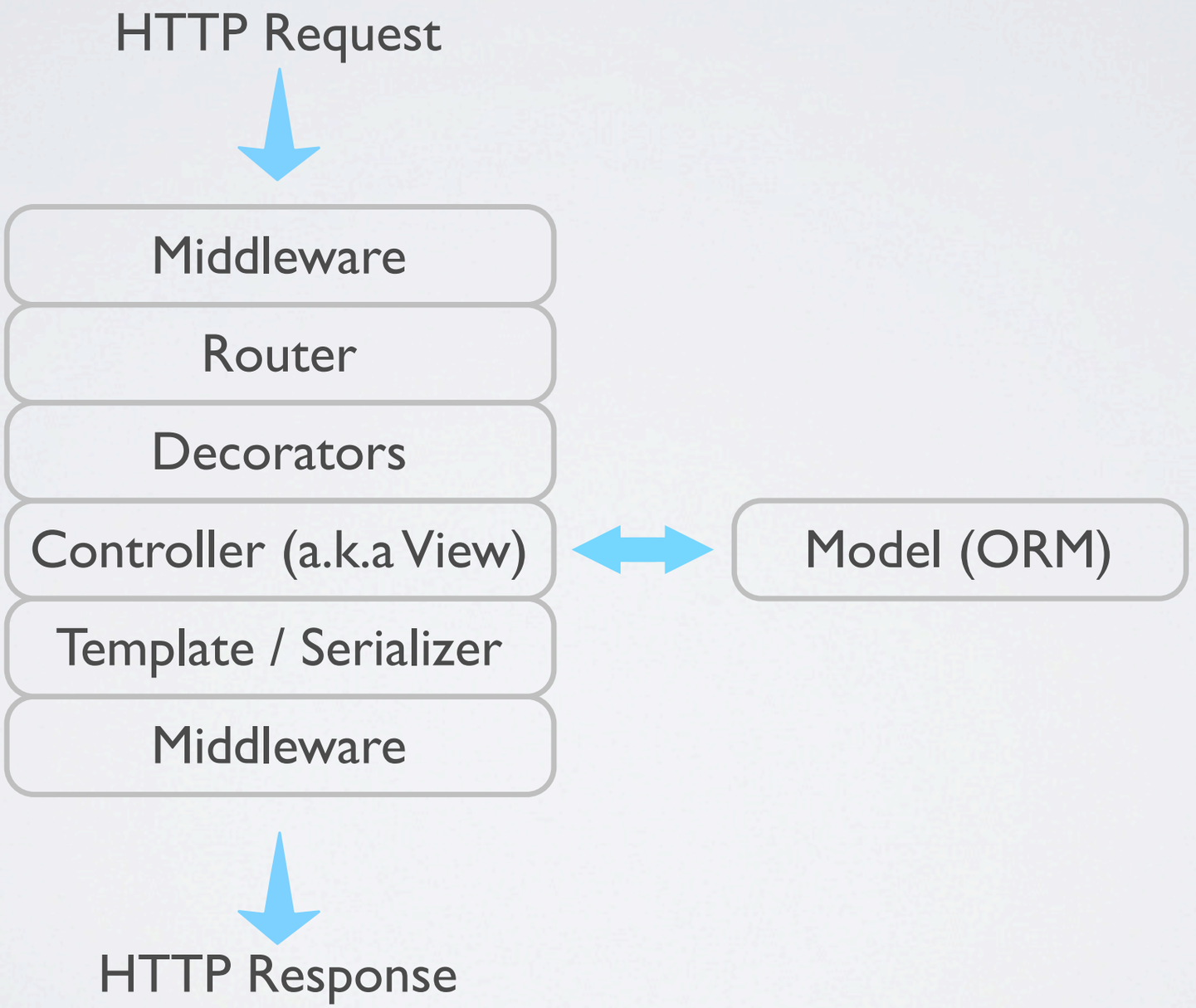
JSON

JavaScript

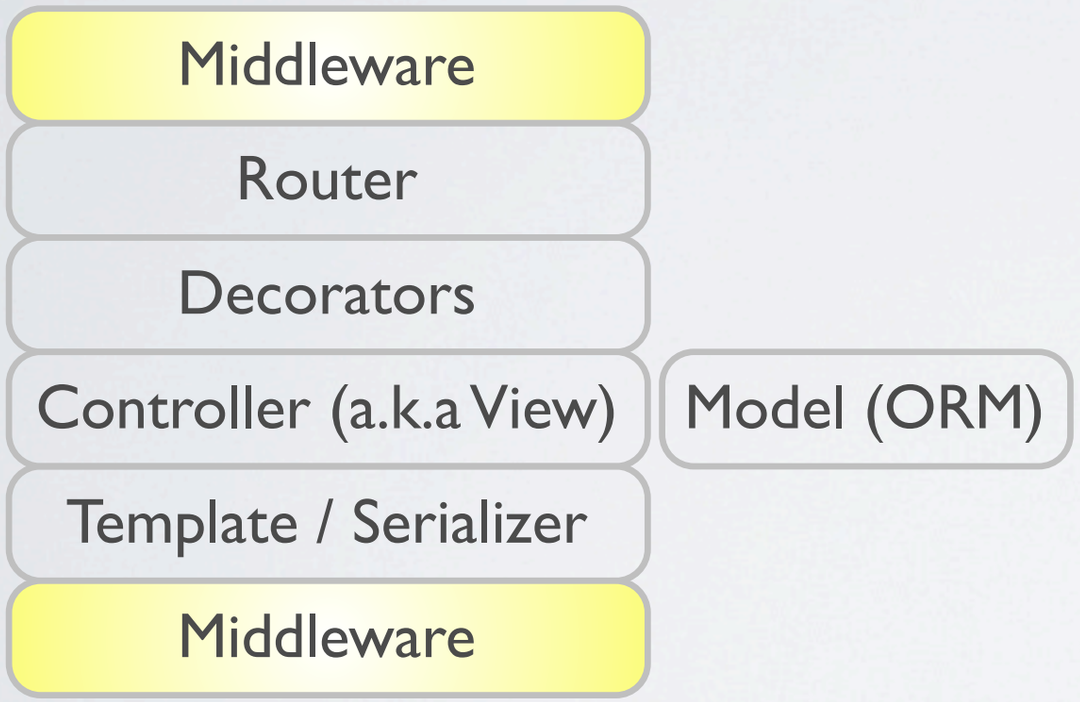
Django

ExtJS

BACKEND (DJANGO)
CODE STRUCTURE



MIDDLEWARE



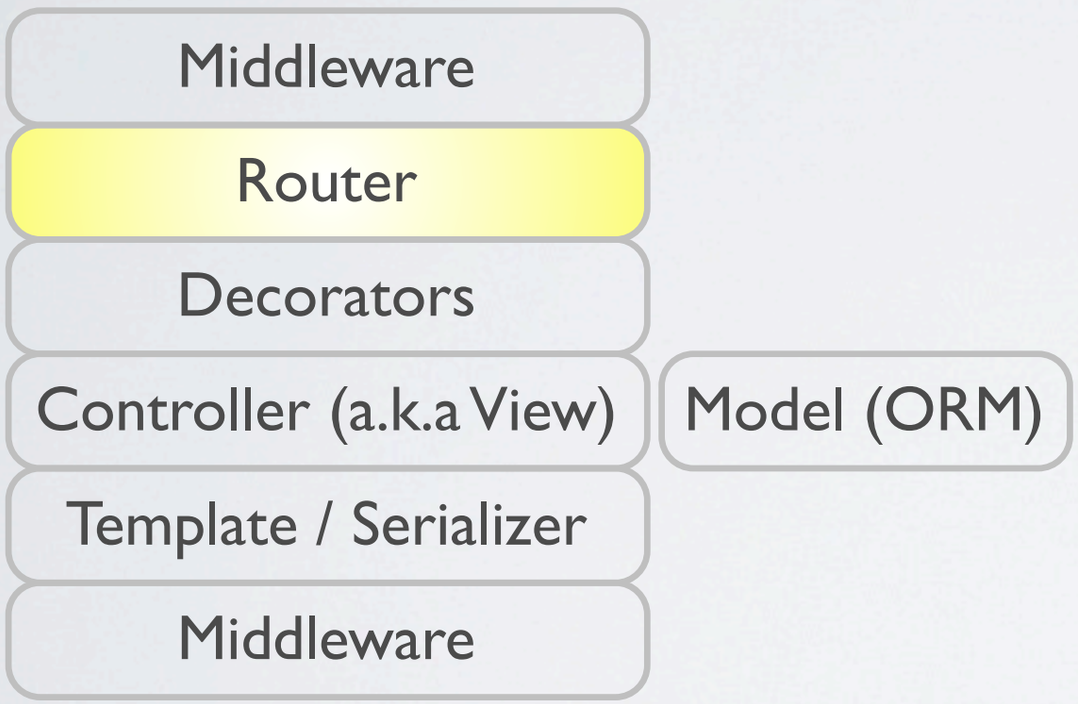
- Processes incoming requests and outgoing responses
- Hooks in before or after the controller function
- e.g. SessionMiddleware, CacheMiddleware

MIDDLEWARE
EXAMPLE

```
class SessionMiddleware(object):
    def process_request(self, request):
        engine = import_module(settings.SESSION_ENGINE)
        session_key = request.COOKIES.get(
            settings.SESSION_COOKIE_NAME, None)
        request.session = engine.SessionStore(session_key)
```

ROUTER

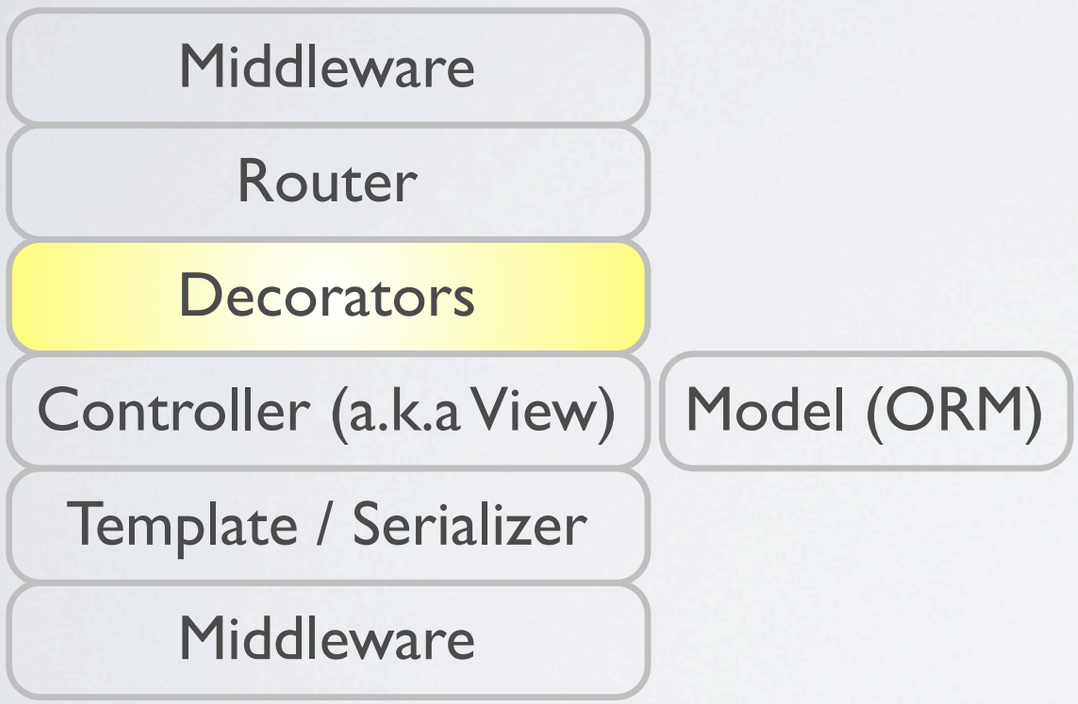
- Maps URL patterns to controller function calls
- Uses regular expressions
- Can use positional and named parameters
- Routes with default parameter values



ROUTER
EXAMPLE

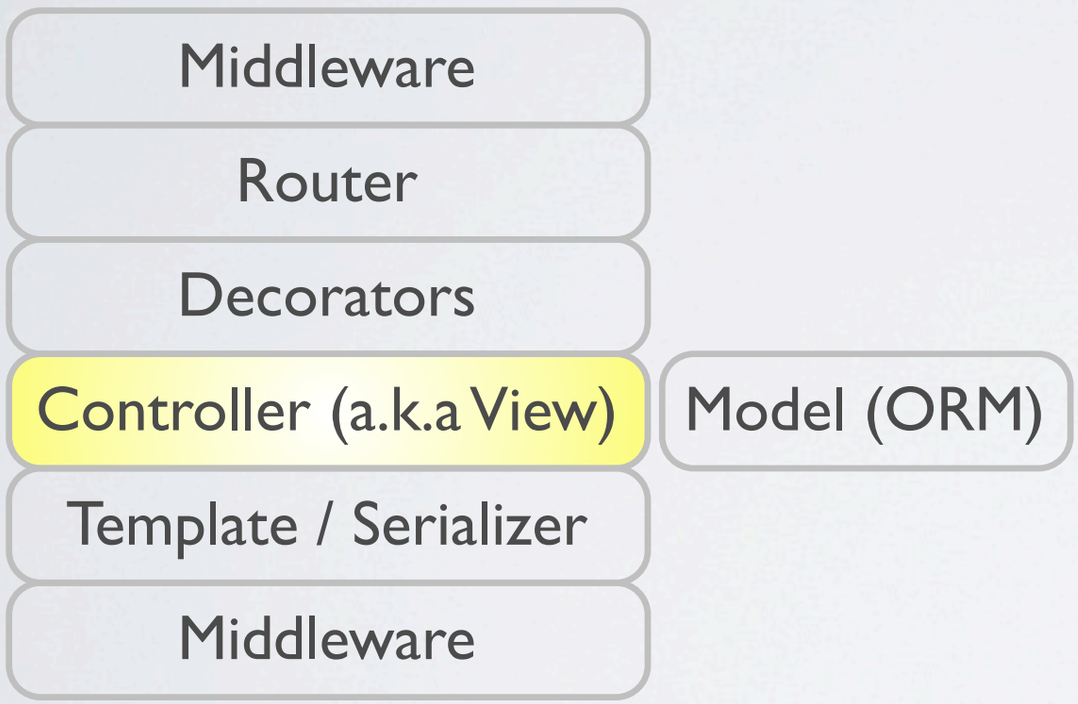
```
urlpatterns = patterns('myapp.views',
    url(r'^$', home_page),                                  # simple pattern

    url(r'^admin/', include(admin.site.urls)),              # pattern inclusion

    url(r'^item/(\w+)/(\d+)/$', display_item),              # positional params

    url(r'^info/(?P<slug>[-\w]+)/$', display_page),         # named params

    url(r'^info/$', display_page, { 'slug' : 'index' }), # default value
)
```

DECORATORS



- Wrap controller functions in additional logic
- Can modify request object or cause a redirect
- e.g. login_required, permission_required, cache_page

CONTROLLER (VIEW)



- Main application logic
- Functions receive an HTTP request and returns an HTTP response
- Communicates with the model layer, session store, etc.

CONTROLLER (VIEW)

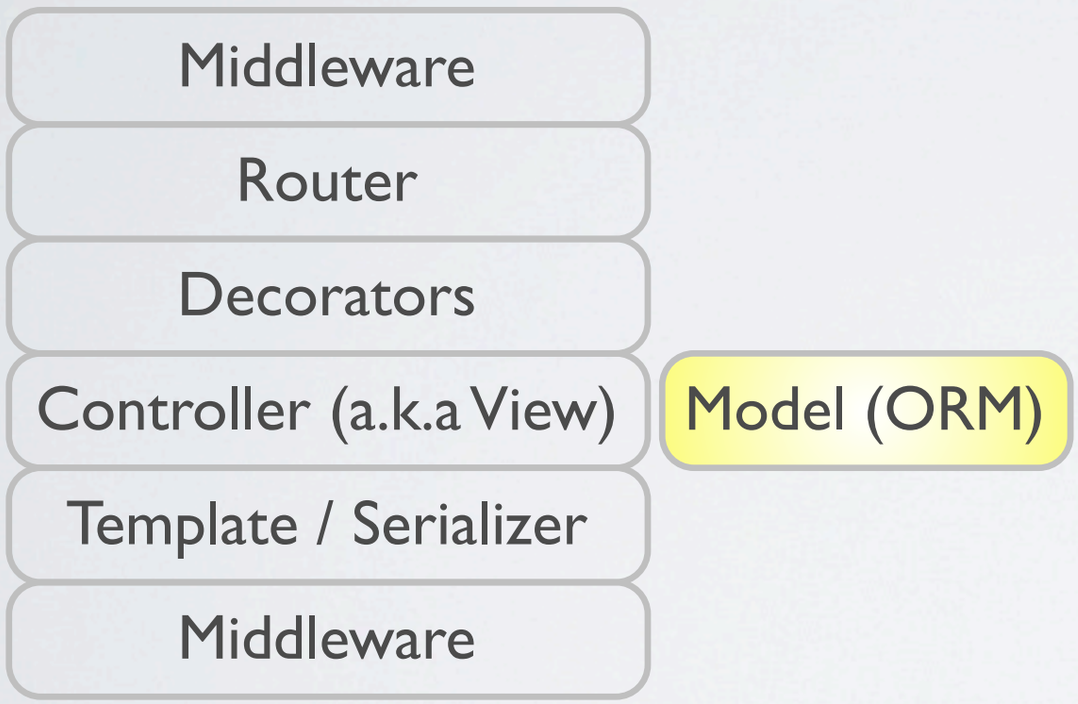
EXAMPLE

```
def display_institute(request, id):
    institute = get_object_or_404(Institute, pk=safeint(id))

    return render_to_response('institute_display.html',
        { 'institute' : institute })
```

CONTROLLER (VIEW)
EXAMPLE

```
def save_offer(self, request):
    form_data = json_deserialize(request.read())

    # Validate incoming offer data
    offer_form = OfferForm(form_data)
    validation_errors = offer_form.errors

    if not validation_errors:
        offer = offer_form.save()
        response = { 'success' : True, 'offer_url' : offer.get_absolute_url() }

    if validation_errors:
        for key in validation_errors.keys():
            validation_errors[key] = " ".join(validation_errors[key])
        response = { 'success' : False, 'errors' : validation_errors }

    return HttpResponse(json_serialize(response),  mimetype='application/json')
```

MODEL (ORM)



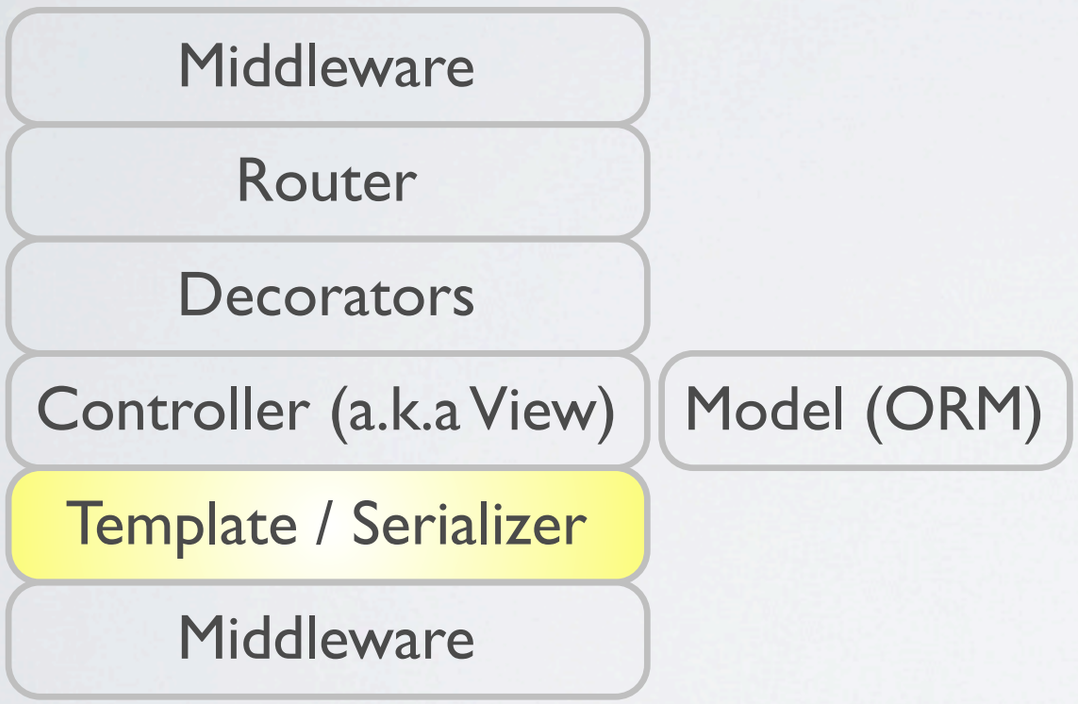
- Database abstraction layer
- Converts DB rows into Python objects
- Generates DB structure and lazy-loading SQL queries

MODEL (ORM)
EXAMPLE

```
class Person(models.Model):
    first_name = models.CharField(_('first name'), max_length=30, blank=True)
    last_name = models.CharField(_('last name'), max_length=30, blank=True)
    email = models.EmailField(_('email address'), blank=True)

    def get_full_name(self):
        """
        Returns the first_name plus the last_name, with a space in between.
        """
        full_name = '%s %s' % (self.first_name, self.last_name)
        return full_name.strip()
```

```
Offer.objects.filter(deadline__date__gte=datetime.date.today())\
    .distinct().annotate(closest_deadline=Min('deadline__date'))\
    .order_by('closest_deadline').select_related()
```

TEMPLATE

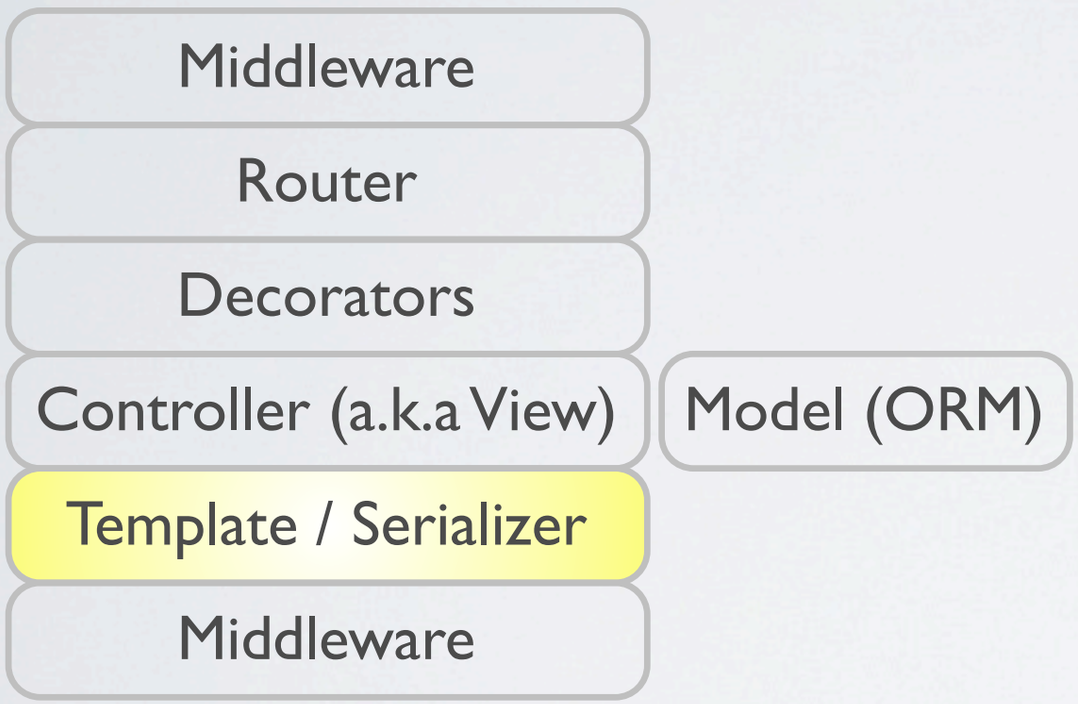


- Generate HTML
- Use Django's templating language

TEMPLATE
EXAMPLE

```
{% extends "base.html" %}
{% load markup_textile %}

{% block title %}{{ page.title }} {% endblock %}

{% block body %}
    <div class="page-header">
        <h3>{{ page.title }}</h3>
    </div>

    <div class="span7">
        <p>
            {{ page.body|textile }}
        </p>
    </div>

    {% if page.comments %}
        <div id="disqus_thread" class="span7"></div>
    {% endif %}
{% endblock %}
```

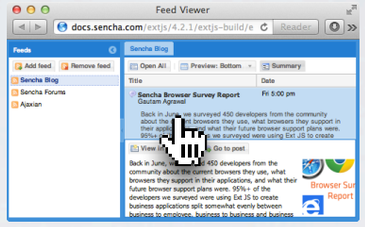
SERIALIZER



- Convert objects into strings for easy transport and data exchange

```
return HttpResponse(json_serialize(data),  mimetype='application/json')
```

APPLICATION ARCHITECTURE



Browser event

click

view update

Middleware

Router

Decorators

Controller (a.k.a View)

Template / Serializer

Middleware

Model (ORM)

Django

response

API

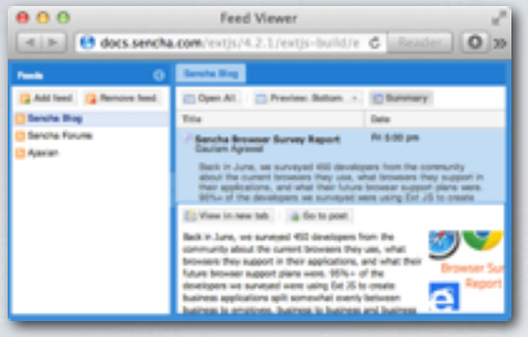
request

ExtJS

Controller

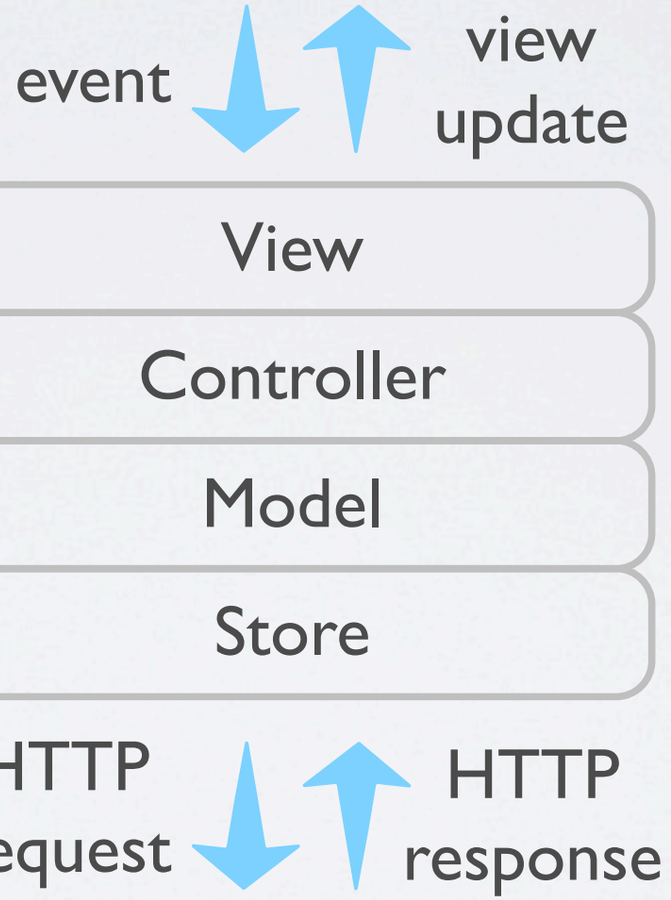
Model

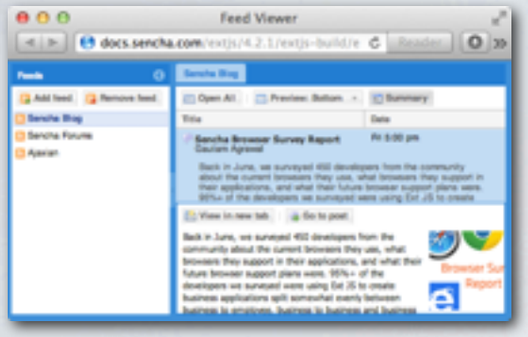
Store



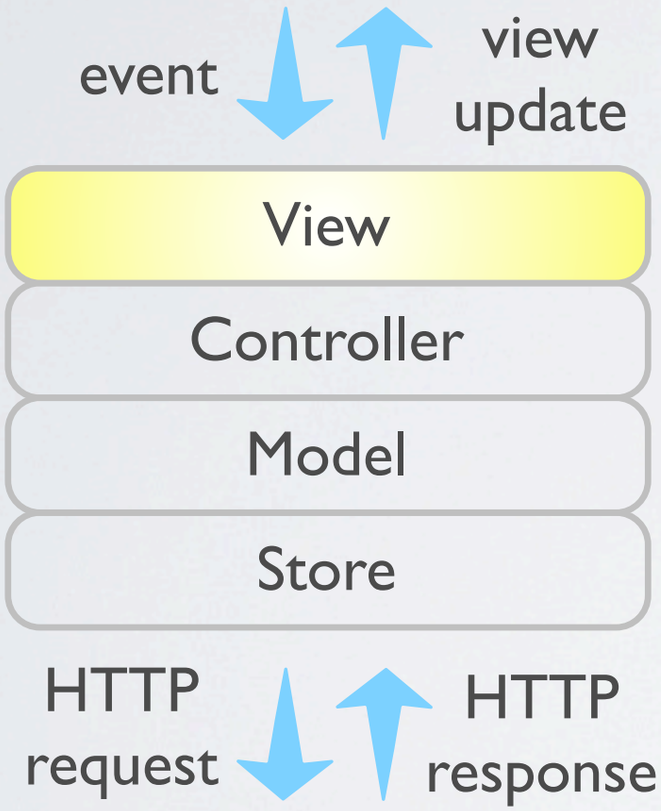
FRONTEND (EXTJS)

CODE STRUCTURE

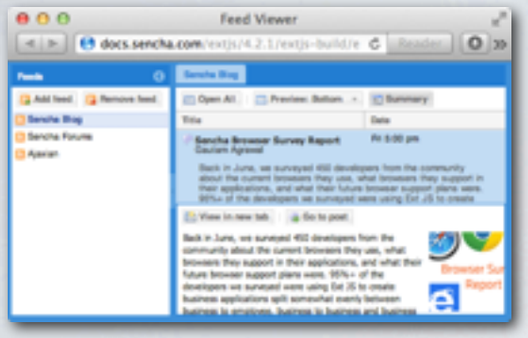




VIEW

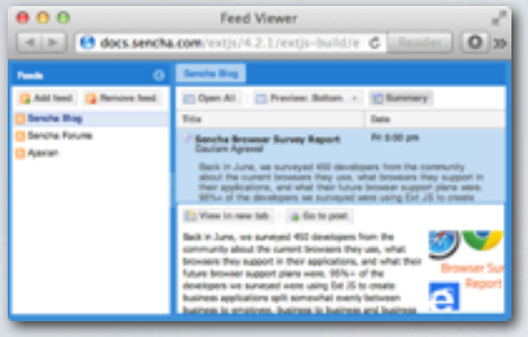


- Sets up visible GUI widgets
- Binds collection views to data stores



VIEW
EXAMPLE

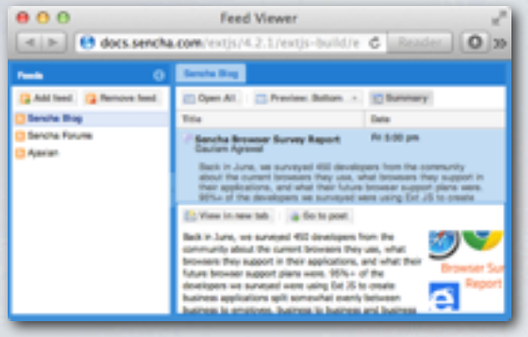
```
Ext.create('Ext.grid.Panel', {
    title: 'People',
    alias  : 'widget.peoplepanel',
    store: Ext.data.StoreManager.lookup('peopleStore'),

    columns: [
        { text: 'First name', dataIndex: 'first_name' },
        { text: 'Last name',  dataIndex: 'last_name' },
        { text: 'Email',      dataIndex: 'email', flex: 1 }
    ],
    height: 200,
    width: 400,
    renderTo: Ext.getBody()
});
```



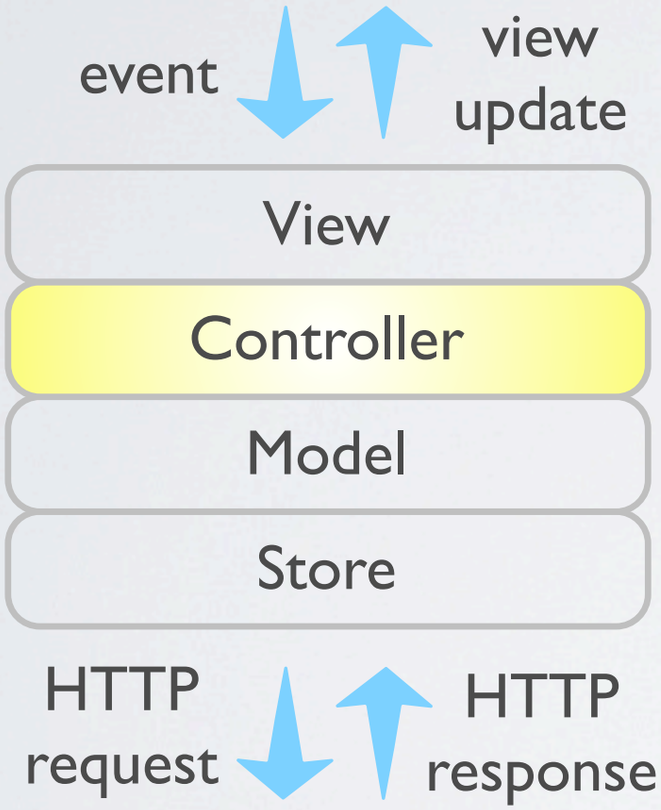
VIEW
EXAMPLE

People

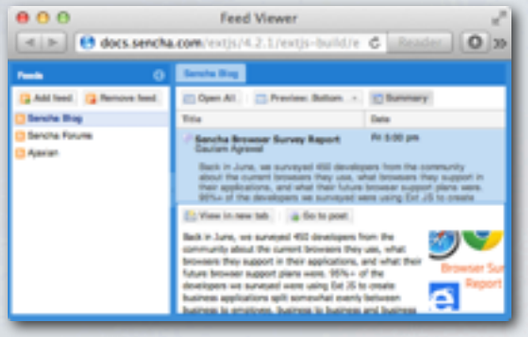
First Name	Last Name ▲	Email
Sterling	Archer	sterling@isis.com
Cyril	Figgis	cyrli@isis.com
Lana	Kane	lana@isis.com
Algernop	Krieger	algernop@isis.com



CONTROLLER



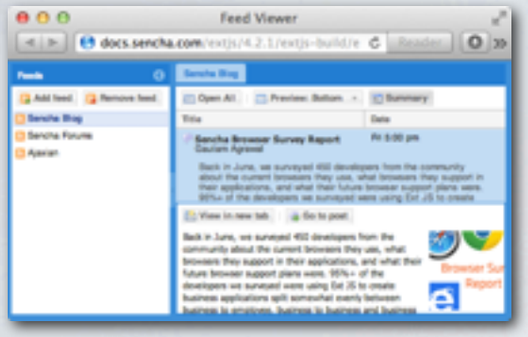
- Main application logic
- Sets up event listeners
- Interacts with models
- ...or sends custom requests to the API, processes responses, updates the GUI



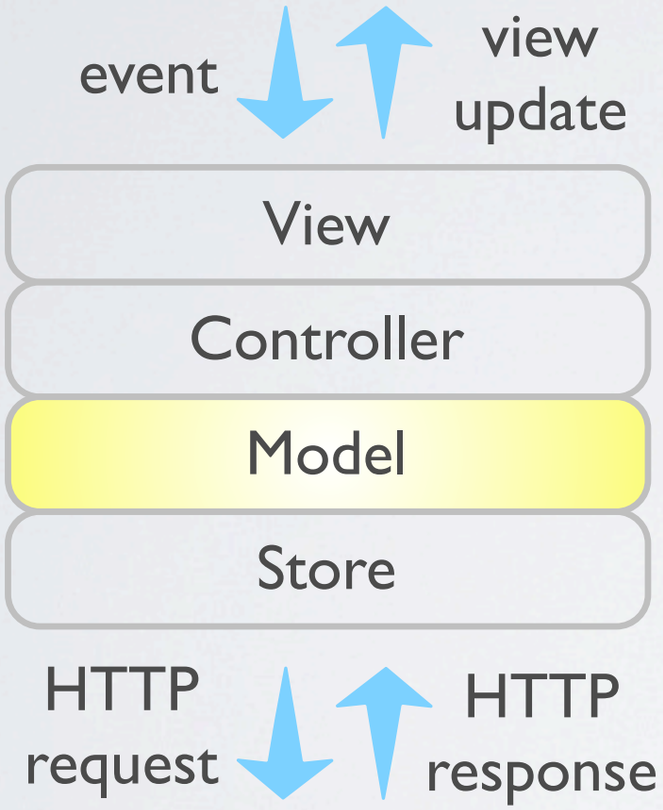
CONTROLLER

EXAMPLE

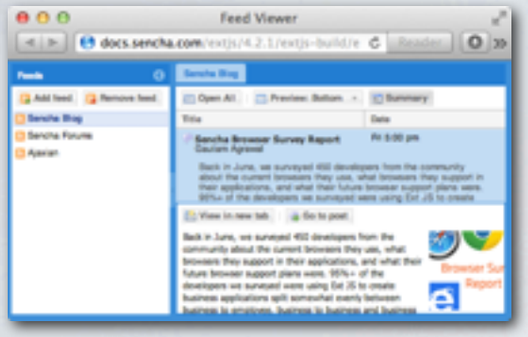
```
Ext.define('MyApp.controller.People', {
    extend:'Ext.app.Controller',

    init: function() {
        this.control({
            'peoplepanel':{
                itemclick : this.onPersonClicked
            }
        });
    },

    onPersonClicked: function(view, record, item, index, e, eOpts){
        console.log('Person item clicked:' record.get('firstName'));

        record.set('was_clicked', true);
        record.save();
    }
});
```



MODEL

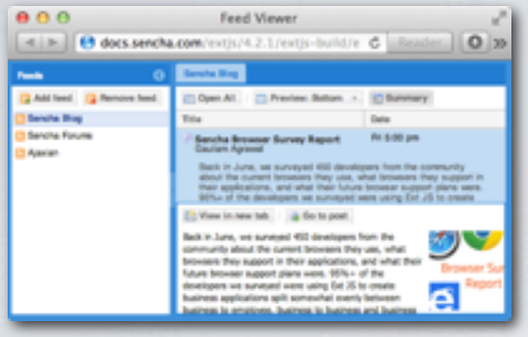


- Represents API data as JavaScript objects
- Client-side representation of server-side models
- Performs data validation

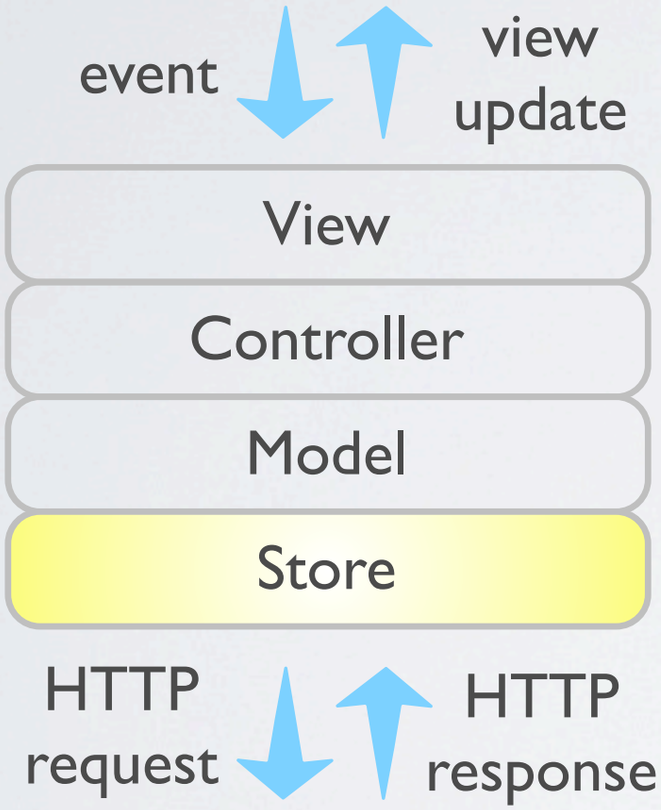


MODEL
EXAMPLE

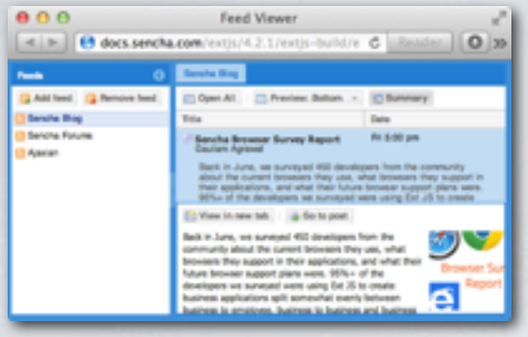
```
Ext.define('Person', {
      extend: 'Ext.data.Model',

      idProperty: 'id',

      fields: [
          {name: 'id',          type: 'int'},
          {name: 'first_name', type: 'string'},
          {name: 'last_name',  type: 'string'},
          {name: 'age',         type: 'int'},
          {name: 'was_clicked',type: 'bool'}
      ],

      get_full_name: function() {
         return this.get('first_name') + ' ' + this.get('last_name');
      }
});
```



STORE



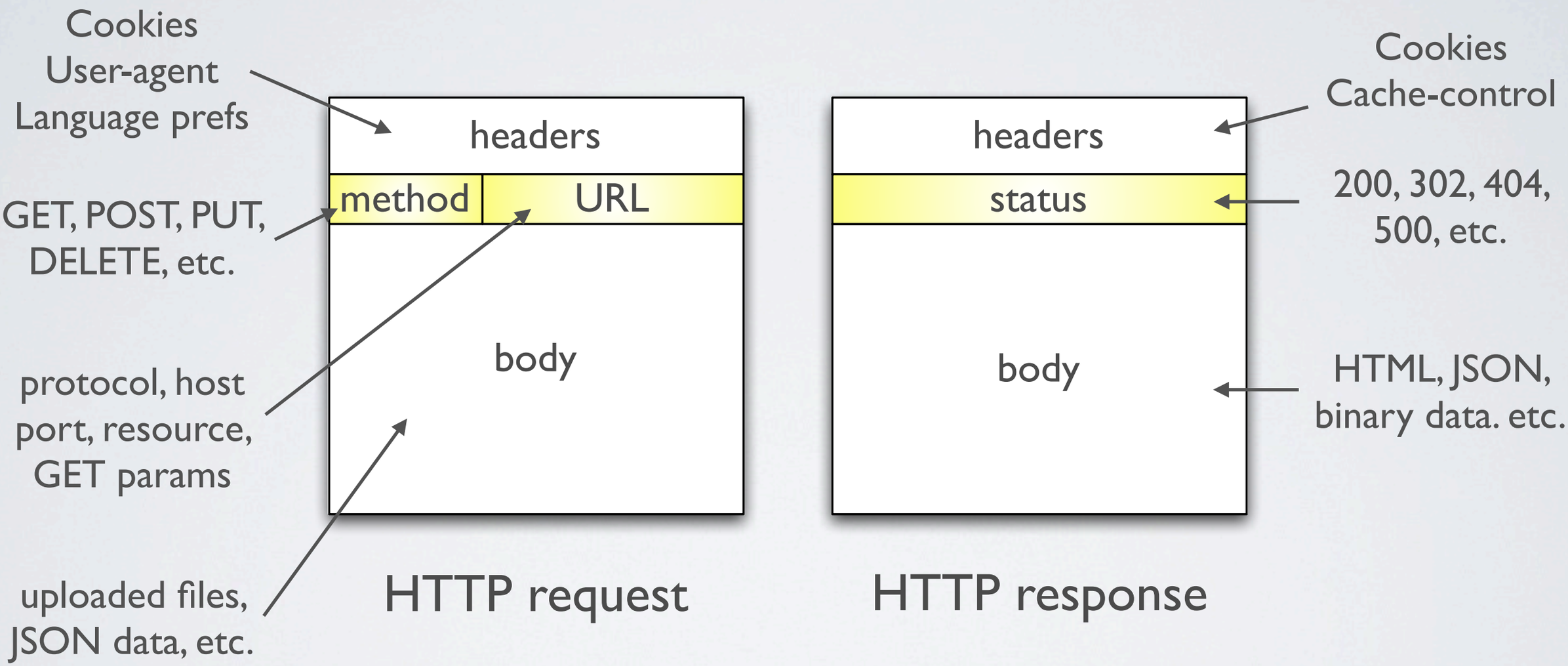
- Communicates with the API
- Enables UI data binding
- Converts API responses to instances of models



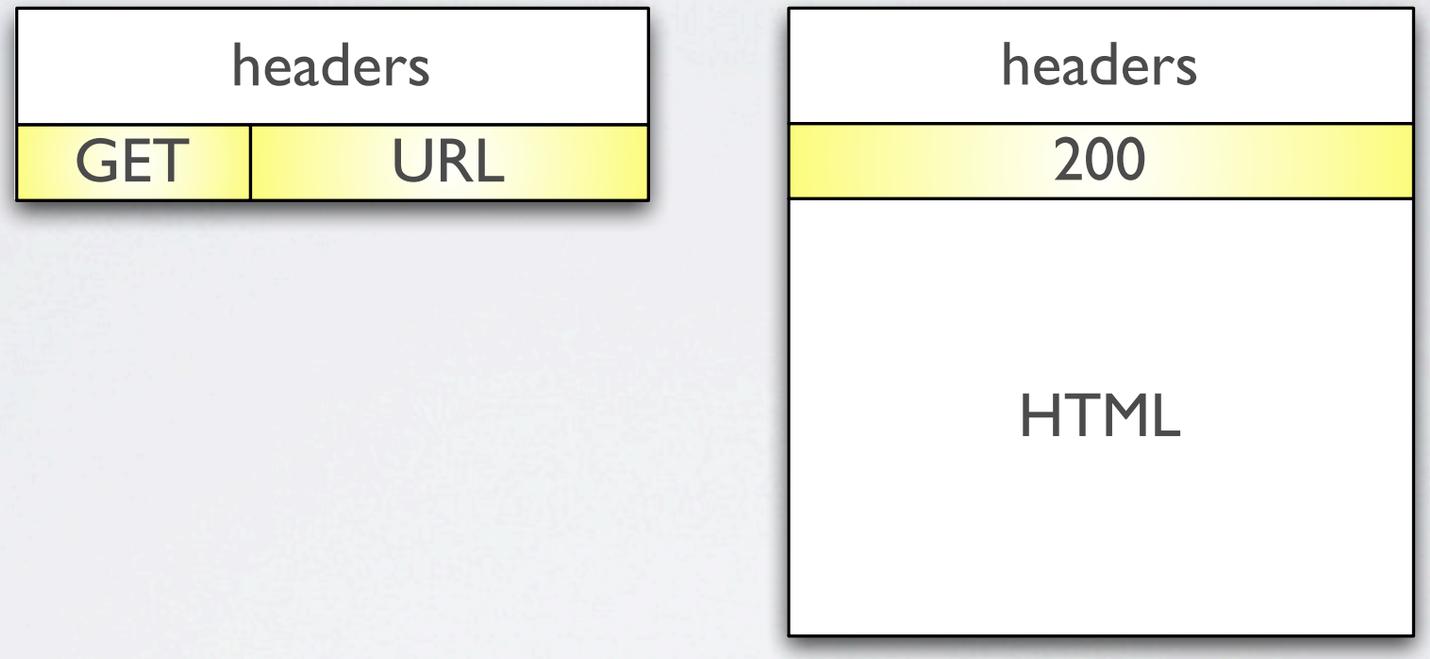
STORE
EXAMPLE

```
Ext.define('MyApp.store.Persons',{
    extend  : 'Ext.data.Store',
    storeId:'peopleStore',
    model: 'MyApp.model.Person'

    proxy: {
        type: 'ajax',
        api: {
            create  : '/api/person/new',
            read    : '/api/person/load',
            update  : '/api/person/update',
            destroy : '/api/person/destroy'
        },
        reader: {
            type: 'json',
            root: 'person',
            successProperty: 'success'
        }
    }
});
```

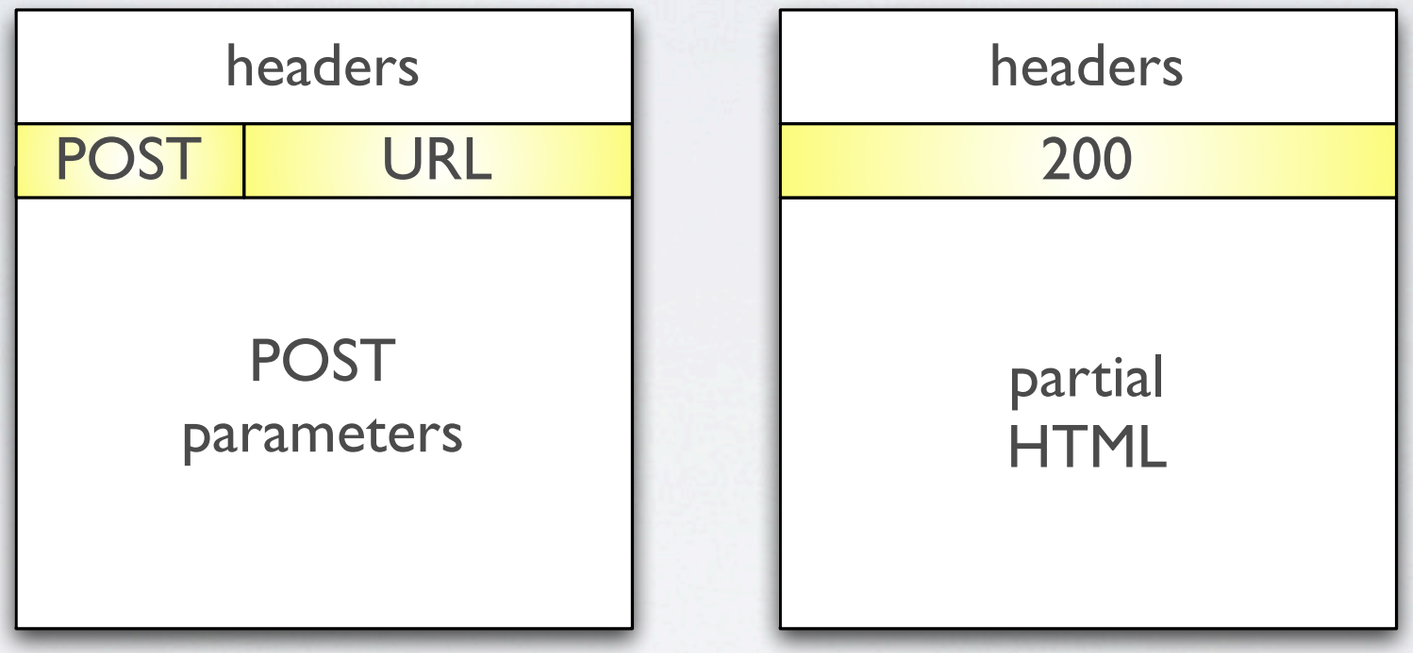
HTTP **API** DESIGN



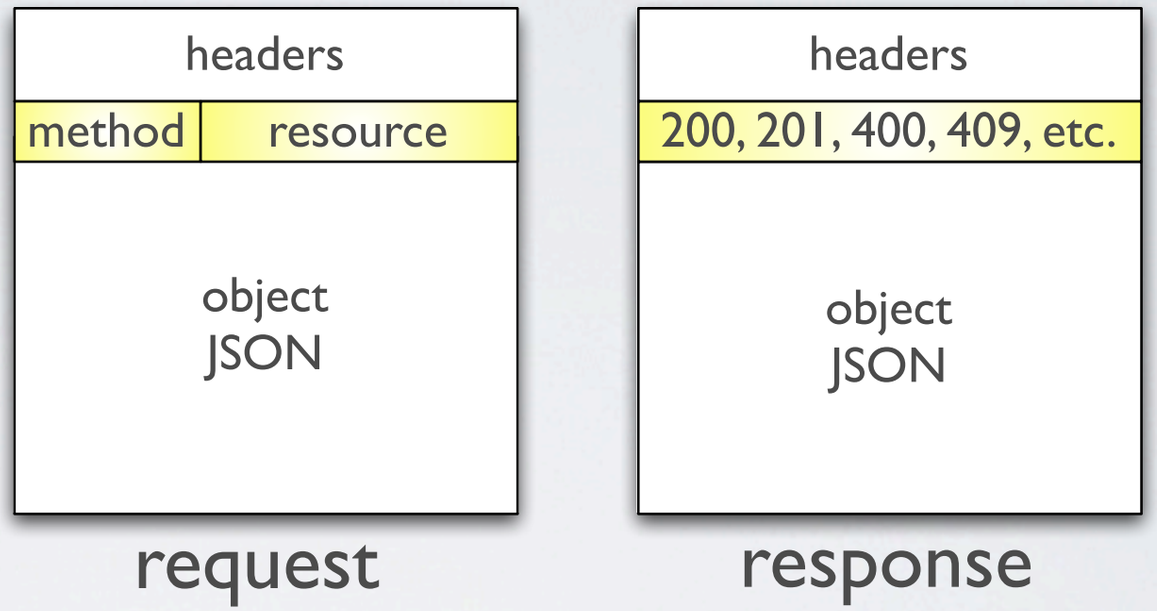
TRADITIONAL HTTP



AJAXIFIED WEBSITE

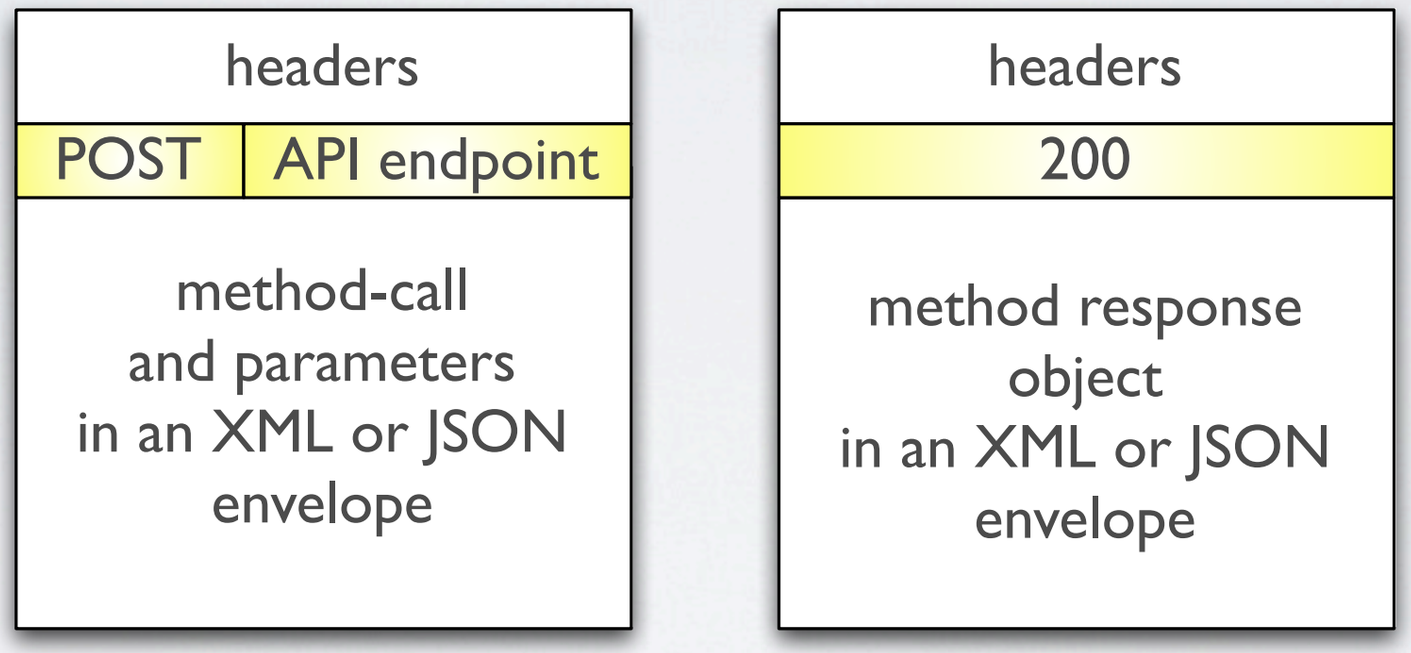


REST



Resource	GET	PUT	POST	DELETE
Collection URI	List elements	Replace collection	Create element	Delete collection
Element URI	Retrieve element	Replace or create		Delete element

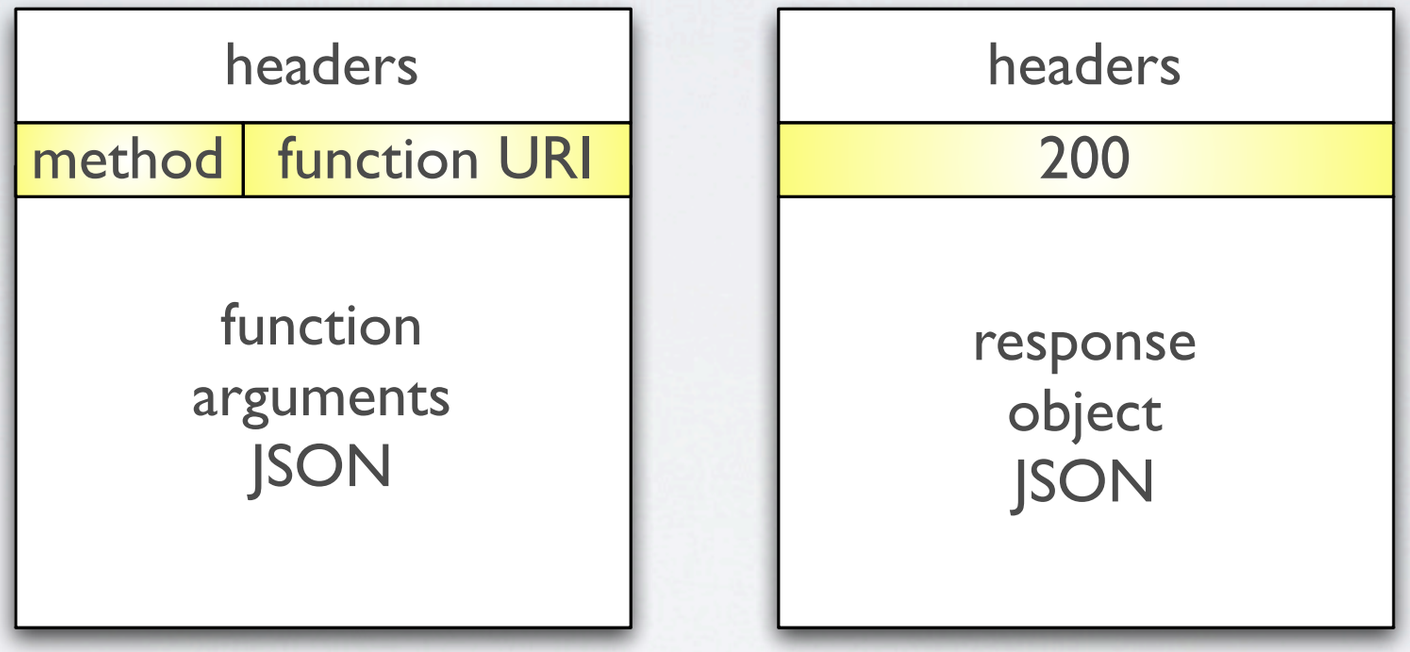
SOAP / JSON-RPC
REMOTE PROCEDURE CALL



request

response

EXTJS-STYLE API



Demo

READY... SET... DEPLOY!!!

SCALING

Step 1: Optimize your code

- optimize number of DB queries
- rewrite slower ORM queries as native SQL (stored procedures)
- use cacheing

SCALING

Step 2: Add **Redis**

- in memory key-value store
- as session store
- as cache backend

SCALING

Step 3: Add **Varnish**

- in-memory cache
- speed up serving static files

SCALING

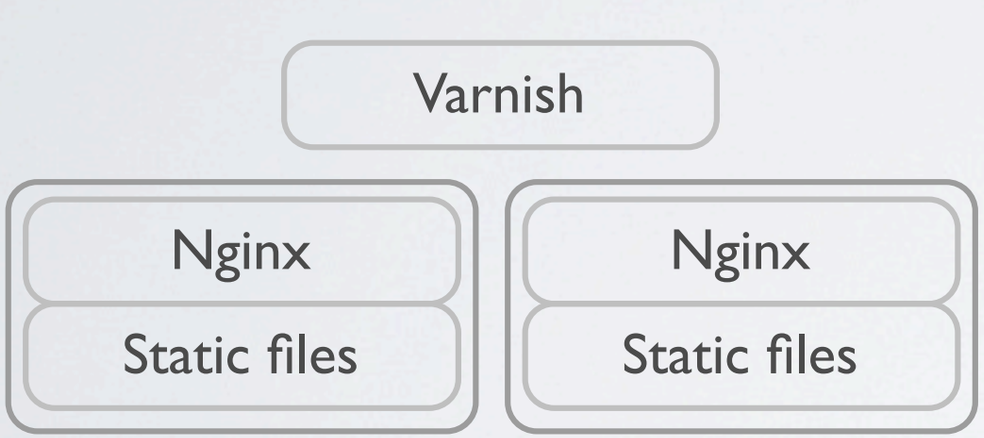
Step 4:

Split up your servers

SCALING

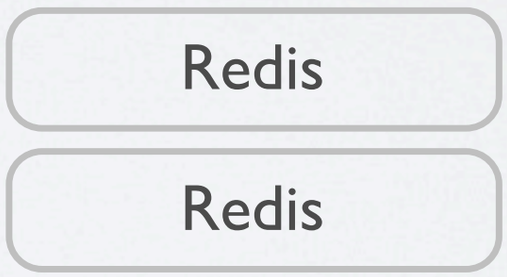
Step 5:

Clusters



Static files cluster
or use a CDN

App-node cluster



Redis cluster
(coming soon)

DB Cluster

Thank you

http://michal.karzynski.pl