REST in Flask

raising the service

I am: Yehor Nazarkin
Follow me: @nimnull
Email me: yehor@mediasapiens.co

REST in Pieces

- REpresentative State Transfer
- The Web is REST
- Hypertext transfer protocol:

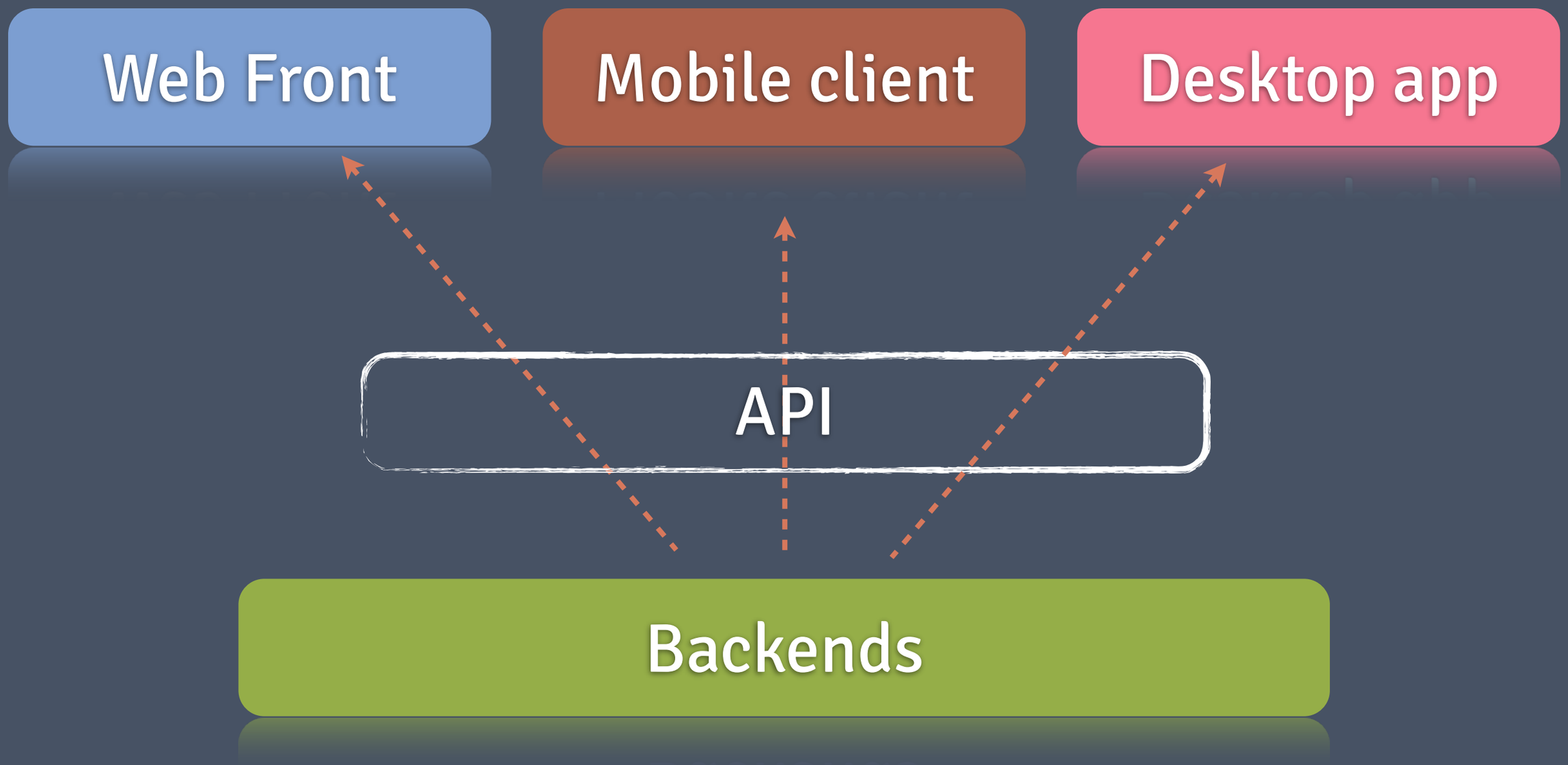
✓ URL endpoint as resource address

✓ Content-Type for data format

✓ Status codes

предыдущие докладчики уже достаточно сказали,
Ключевые моменты, полезные для построения сервисов
не RPC! — используйте HTTP методы по назначению

REST in Pieces



smart websolutions!

Why Flask? as we already have tones of stuff

- One drop at a time
- Clean codebase (20 python modules)
- I want to customize, not rewrite
- I’m not a single framework junkie

Why Flask? as we already have tones of stuff

	Flask	Django
WSGI	Werkzeug	own
Templates	Jinja2	own
Signals	Blinker	own
i18n	Babel	own
ORM	any	own

Does it called a NIH Syndrome?

что делать если не достаточно стандартного набора компонентов

smart websolutions!

Why Flask? as we already have tones of stuff

In general — because we can!

Achievements yes, we did

Based on Flask's MethodView

Extended with method-decorators

Simple to attach, easy to extend

Наша надстройка использует максимум существующих решений

Расширенный набор декраторов для обработки каждого типа HTTP запроса

Добавился MongoResource

Resource — for generic cases

ModelResource — SQLA mapped classes

DocumentResource — MongoDB mapped classes

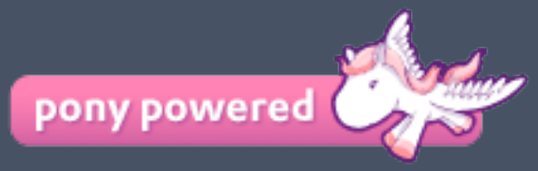
Derived classes — for special cases

Achievements yes, we did

Inspired by

django-tastypie

flask-restless



Achievements yes, we did

In Practice

```
@api_resource(account, 'sessions', {'id': None})
class SessionResource(Resource):
    validation = t.Dict({'email': t.Email}).allow_extra('*')
```

```
@api_resource(account, 'addresses', {'id': int})
class AddressResource(ModelResource):

    validation = t.Dict({'city': t.String,
                         'street': t.String,
                         'type': t.String(regex="(bill|delivery)")}
                        ).allow_extra('*')
    model = Address

    def get_objects(self):
        ...
    def get_object(self):
        ...
```

Achievements yes, we did

Differences

```
@api_resource(account, 'profiles', {'id': None})
class RoleResource(ModelResource):
    validation = ...

    decorators = [login_required, ]

    method_decorators = {'post': check_permission('is_superuser'),
                         'put': check_permission('is_superuser')}
```

Responses and fails

JSON Source

```
{‘email’: ‘test@examplecom’, ‘password’: ‘test’}
```

ERROR 400

```
{‘message’: ‘Bad data request’}
```

```
{‘email’: ‘not an email-type’,
 ‘password’: ‘should be more than 6 chars length’}
```

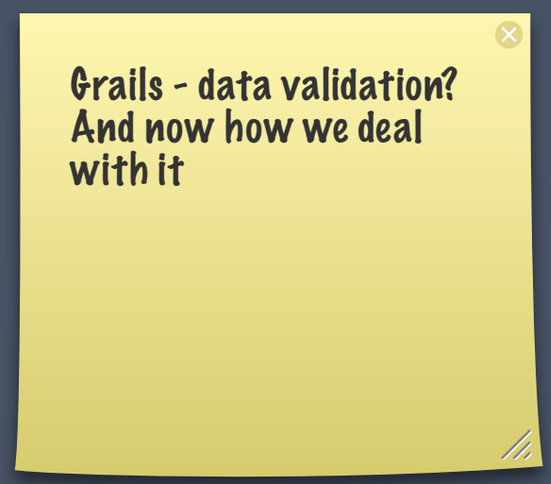
Validation resource tier, model tier

I don't trust those "data" arrived from anywhere outside

Solution

Trafaret http://pypi.python.org/pypi/trafaret

```
import trafaret as t

t.Dict({'first_name': t.String,
        'last_name': t.String,
        'phone': t.String,
        t.Key('role_id', optional=True): t.Int
       }).ignore_extra('*')
```



Validation resource tier, model tier

Key subset, data transformation:

```
t.Dict({'first_name': t.String,
        te.KeysSubset('pass', 'confirm'): cmp_pwds})
```

```
def cmp_pwds(value):
    if value['pass'] != value['confirm']:
        return {'confirm': t.DataError("Doesn't match")}
    return {'pass': value['pass']}
```

Validation resource tier, model tier

Type check:

- Null, Bool, String, Int, Float, List, Dict
- Atom, Email, URL, Enum, Callable
- Converters, ignorance, excludes, optional keys
- Useful case for MongoDB Documents also

Method arguments type check:

```
@guard(a=String, b=Int, c=String)
def guarded(a, b, c):
    return a, b, c
```

Roadmap no, we don't

To be done

- Filtering (DSL?)
- Caching
- Throttling?

Otherside from the client-side point

Almost forget

All client side is written with CoffeeScript

Win:

Extended dialect (inheritance, list comprehensions, context pass etc.)

Less visual garbage (indentation match, braceless, string interpolation)

Easy to read, cause you read much more often than write (I hope)

Fail:

You have to learn

A cup of coffee

In details

```
class PageView extends View
  template: require ‘templates/page’

  initialize: ->
    super
    model.on ‘change’, =>
      @render()
    model.fetch()
```

```
var PageView,
  __hasProp = {}.hasOwnProperty,
  __extends = function(child, parent) {
    for (var key in parent) {
      if (__hasProp.call(parent, key)) child[key] = parent[key];
    }
    function ctor() { this.constructor = child; }
    ctor.prototype = parent.prototype;
    child.prototype = new ctor();
    child.__super__ = parent.prototype;
    return child; };

PageView = (function(_super) {

  __extends(PageView, _super);

  function PageView() {
    return PageView.__super__.constructor.apply(this, arguments);
  }

  PageView.prototype.template = require(‘templates / page’);

  PageView.prototype.initialize = function() {
    var _this = this;
    PageView.__super__.initialize.apply(this, arguments);
    model.on(‘change’, function() {
      return _this.render();
    });
    return model.fetch();
  };

  return PageView;

})(View);
```

is it mess? :(

Spaghetti, sir?

Otherside from the client-side point

Chaplin.js

Based on top of the Backbone.js

Set of AMD modules

Extends MVC, event-driven

Still a lightweight tool

Brunch.io

To build-it-all

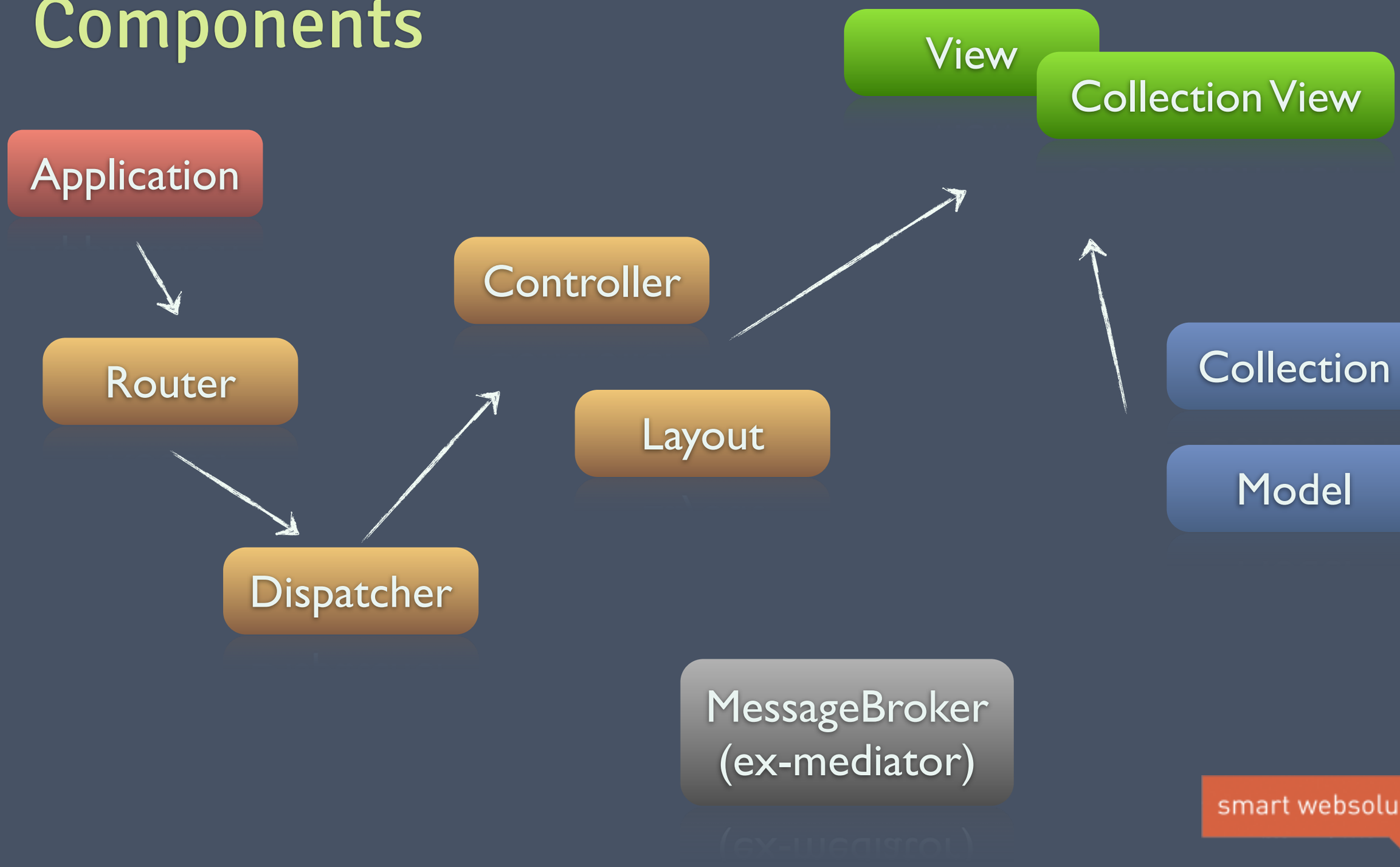
пример с angular js зафейлился, поэтому я могу предложить альтернативу в виде проверенных компонет

AMD for modular

```
define ‘nav_view’, [
    ‘view’
],(View) ->
  ‘use strict’

  class NavView extends View

    initialize: ->
      super
      @delegate ‘click’, ‘.popup’, @propagate

    propagate: ->
      @publishEvent ‘curtains:show’
```

Otherside from the client-side point

Components



Otherside from the client-side point

Flow

URL://page/about

```
module.exports = (match) ->
  match 'page/:slug', 'pages#show'
```

Router

```
controller: ‘pages_controller’
action: ‘show’
args: {slug: ‘about’}
```

Dispatcher

```
class PageController extends Controller

  show: ({slug}) ->
    model = new PageModel id: ‘slug’
    @view = new PageView {model}
```

Controller

View

```
class PageView extends View

  template: require ‘templates/page’

  initialize: ->
    super
    model.on ‘change’, =>
      @render()
    model.fetch()
```

smart websolutions!

Otherside from the client-side point

Event passthrough

```
class PageView extends View

  initialize: ->
    super
    @subscribeEvent ‘curtains:show’, @showCurtains

  showCurtains: ->
    @$(‘.curtains’).show()
```

```
class NavView extends View

  initialize: ->
    super
    @delegate ‘click’, ‘.popup’, @propagate

  propagate: ->
    @publishEvent ‘curtains:show’
```

MessageBroker
(ex-mediator)

smart websolutions!

Questions? write code, not hollywar

Coffee-script	http://coffeescript.org
Brunch	http://brunch.io
Chaplin	http://github.com/chaplinjs/chaplin
Flask	http://flask.pocoo.org