AN IPHONE-PYTHON LOVE AFFAIR

Building APIs for Mobile

Music Hack Day, February 2011

Samantha & Matt

ACCOMPLICE #1

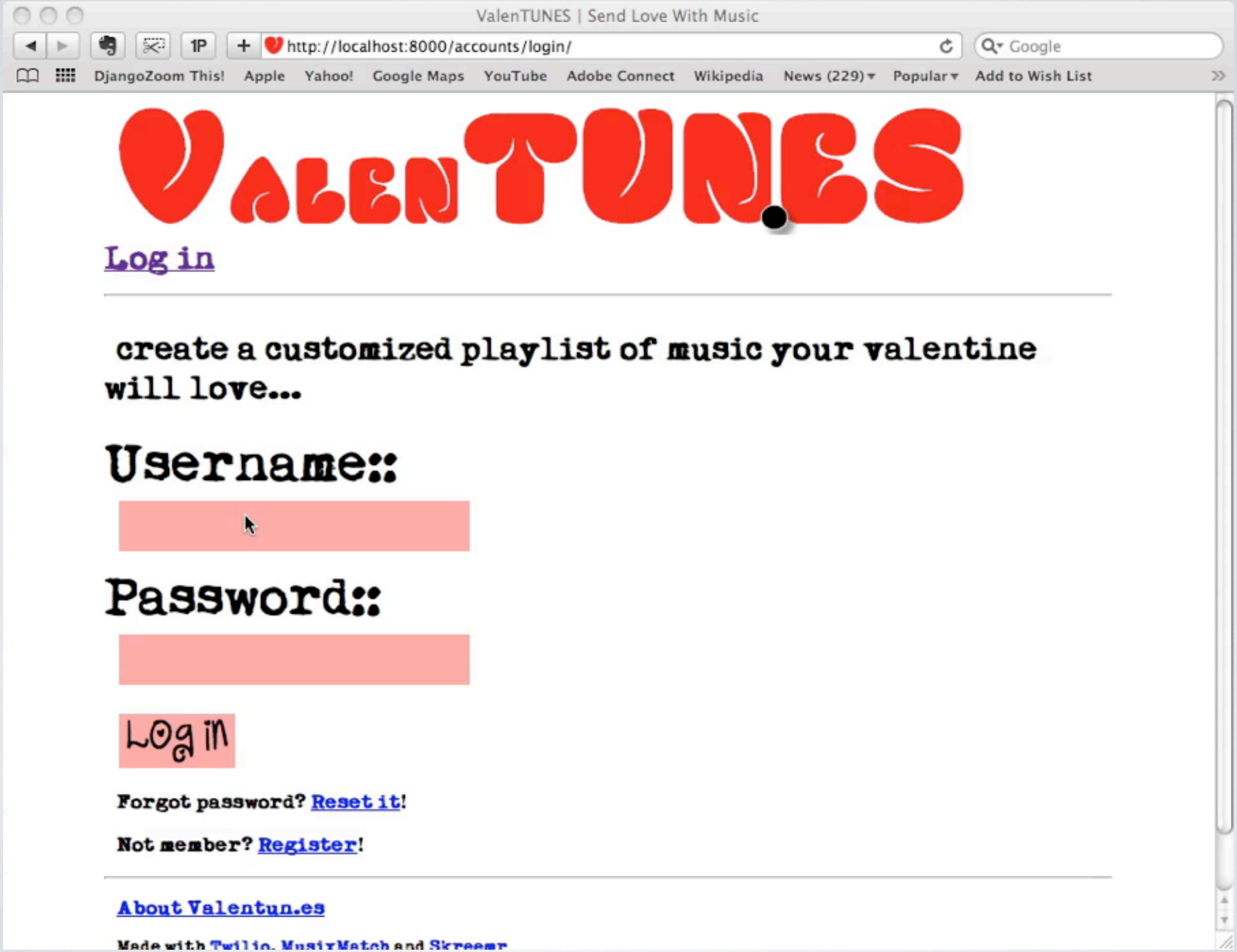
Anna Callahan: iOS developer, jazz trumpet player
@jazztpt

ACCOMPLICE #2

Nate Aune: Django/Python/Plone developer, saxophonist
@natea

VALENTUN.ES!

http://www.youtube.com/watch?v=0C02wev40K0



ValenTUNES | Send Love With Music

http://localhost:8000/accounts/login/

ValenTUNES

Log in

create a customized playlist of music your valentine will love...

Username::

Password::

Log in

Forgot password? Reset it!

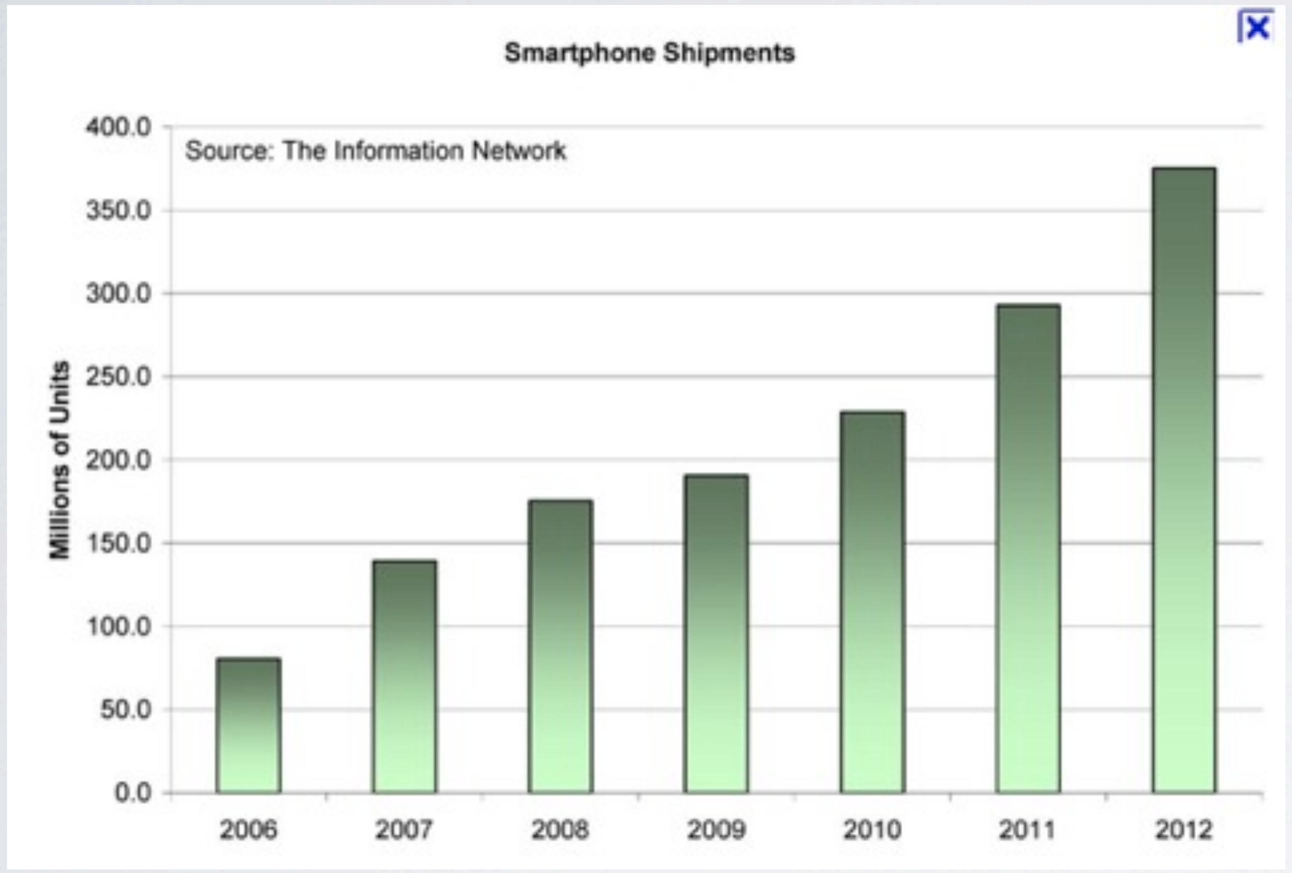
Not member? Register!

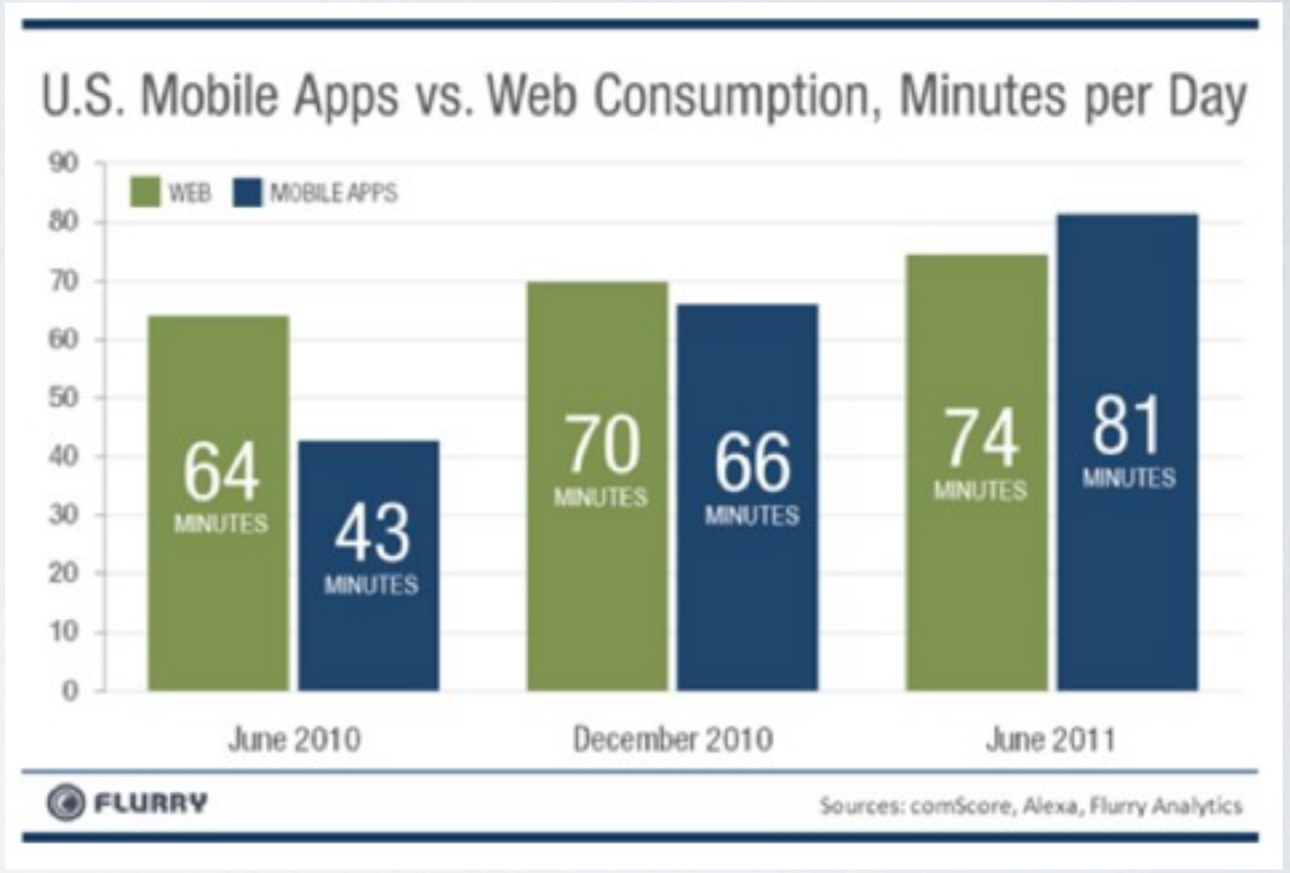
About Valentun.es

Made with Twilio, MusixMatch and Skreemr

Friday, June 24, 2011

MOBILE TAKEOVER





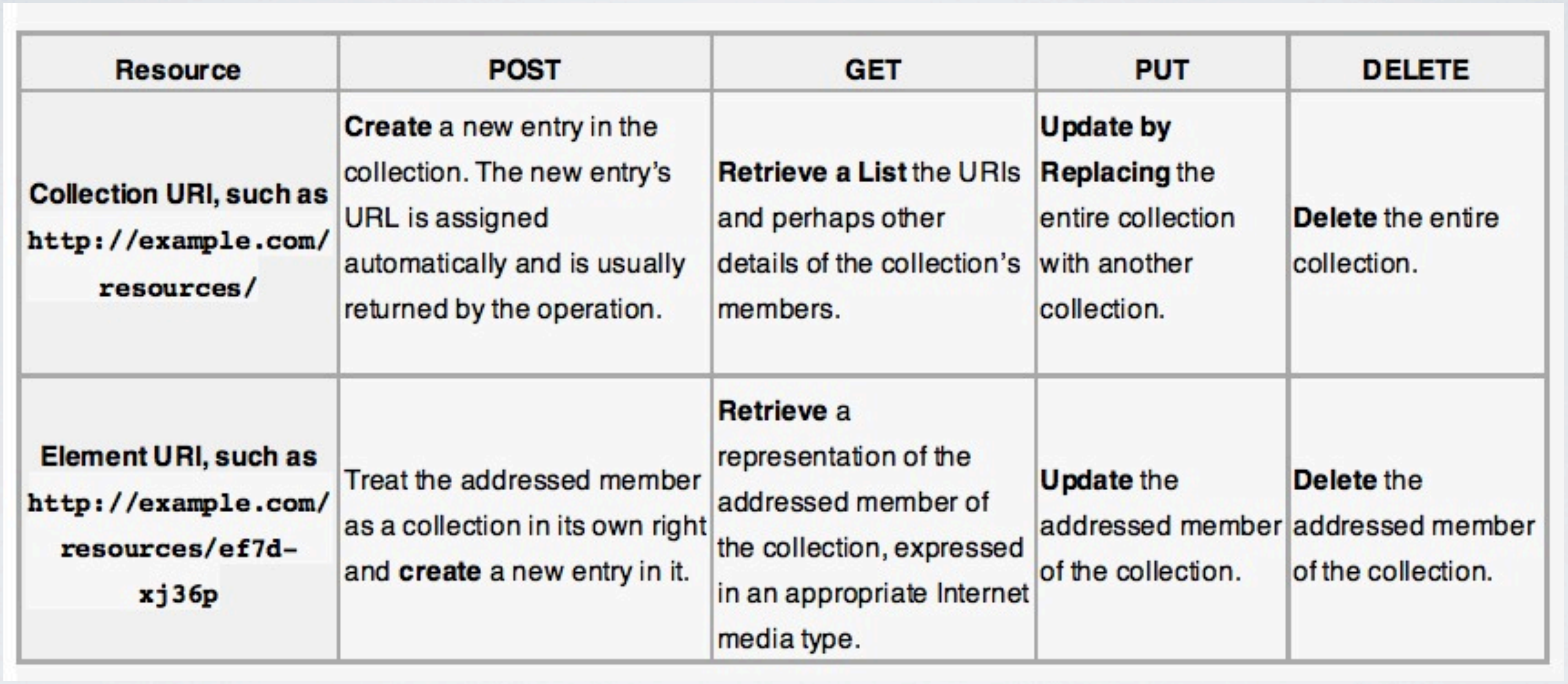
If you're not building for mobile now, you will be soon.

WHAT'S DIFFERENT ABOUT MOBILE?

- Isn't REST appropriate for everything?
- Don't I want a single API for all clients?

* You or your customer controls the mobile app.

WHAT IS REST?



Resource	POST	GET	PUT	DELETE
Collection URI, such as **`http://example.com/resources/`**	**Create** a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	**Retrieve a List** the URIs and perhaps other details of the collection's members.	**Update by Replacing** the entire collection with another collection.	**Delete** the entire collection.
Element URI, such as **`http://example.com/resources/ef7d-xj36p`**	Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Update** the addressed member of the collection.	**Delete** the addressed member of the collection.

Typical REST API implementation.

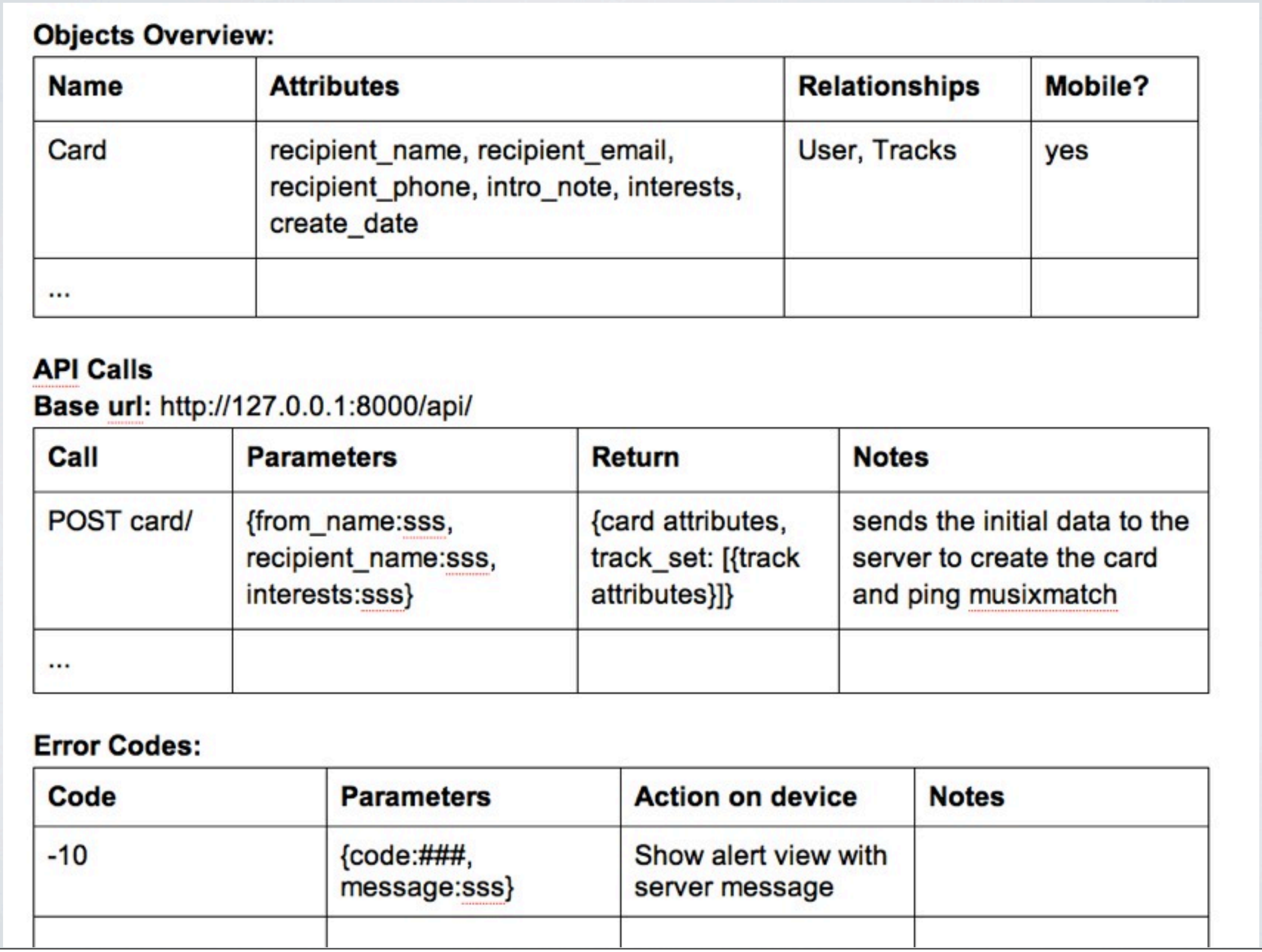
WHAT'S DIFFERENT ABOUT MOBILE?

Mobile users are unwilling to wait.

1. Connection = slow, spotty, or non-existent

2. Mobile is not as powerful at fetching & saving data or calculations

CREATE AN API DOC



Objects Overview:

Name	Attributes	Relationships	Mobile?
Card	recipient_name, recipient_email, recipient_phone, intro_note, interests, create_date	User, Tracks	yes
...			

API Calls
Base url: http://127.0.0.1:8000/api/

Call	Parameters	Return	Notes
POST card/	{from_name:sss, recipient_name:sss, interests:sss}	{card attributes, track_set: [{track attributes}]}	sends the initial data to the server to create the card and ping musixmatch
...			

Error Codes:

Code	Parameters	Action on device	Notes
-10	{code:###, message:sss}	Show alert view with server message	

SO YOUR API SHOULD

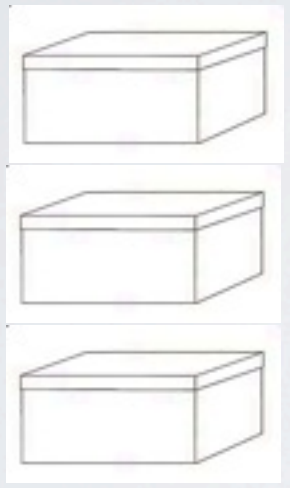
- Return hierarchies of related data
- Authentication and Authorization
- Have mobile-specific error codes & messages
- Accept arrays of related or unrelated data
- Return pre-calculated data or data that doesn’t exist on device

HIERARCHIES OF DATA

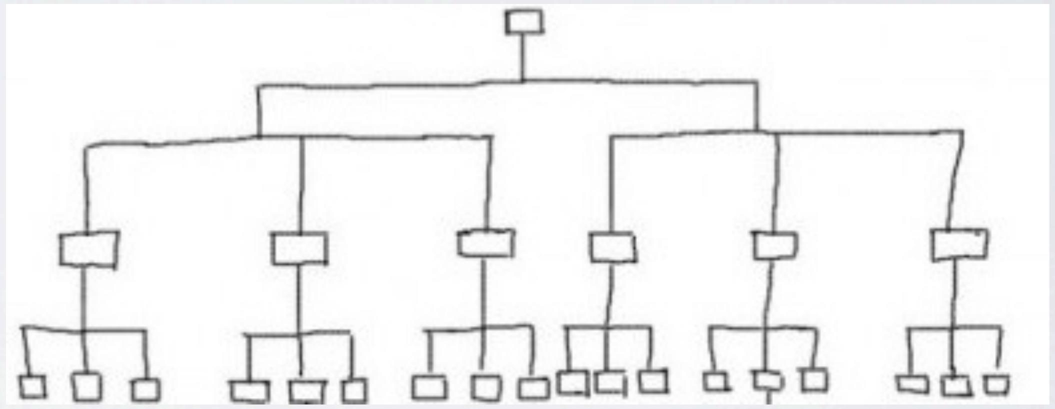
Bad :(

Good!

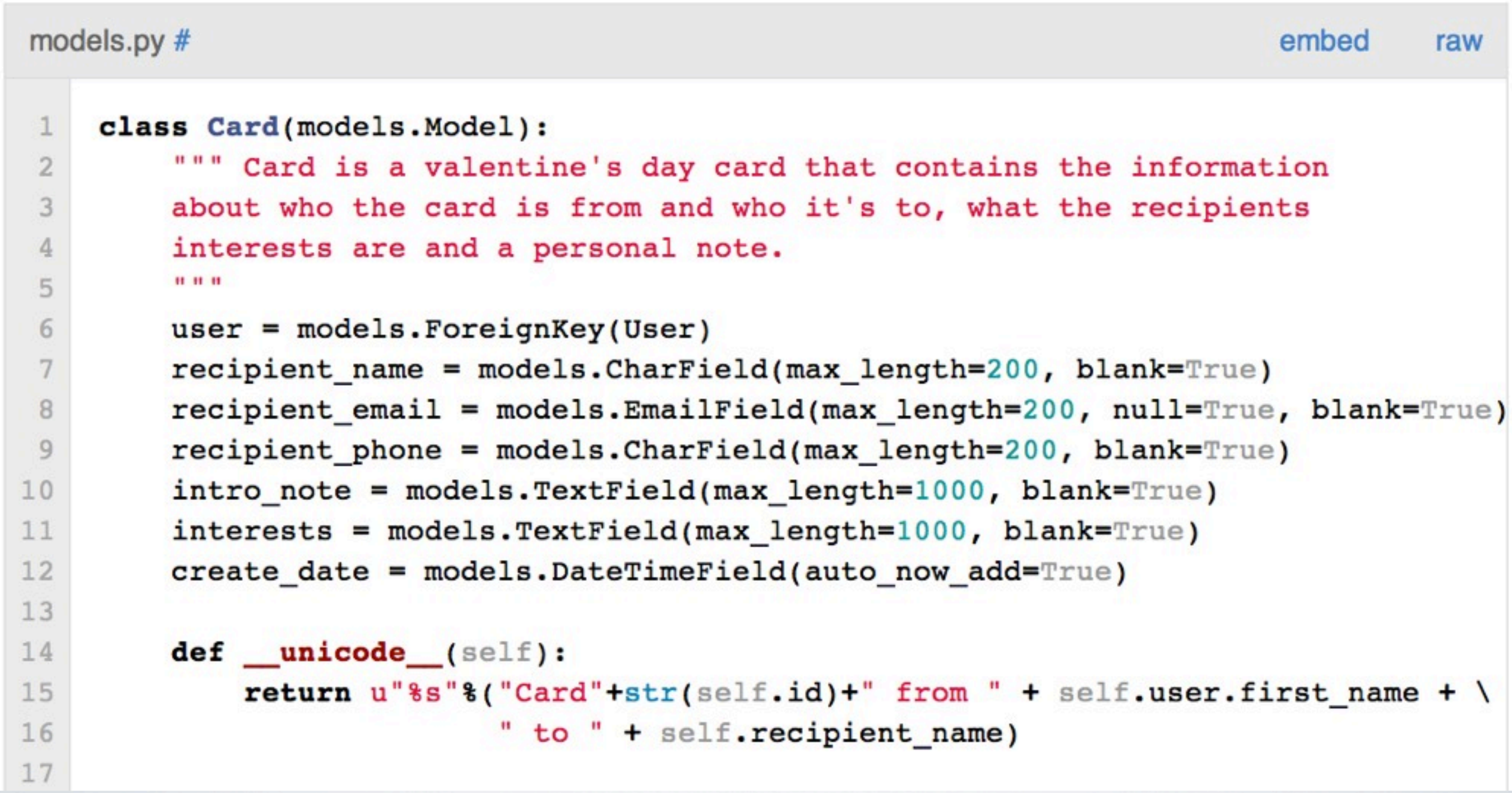
Cards



Tracks



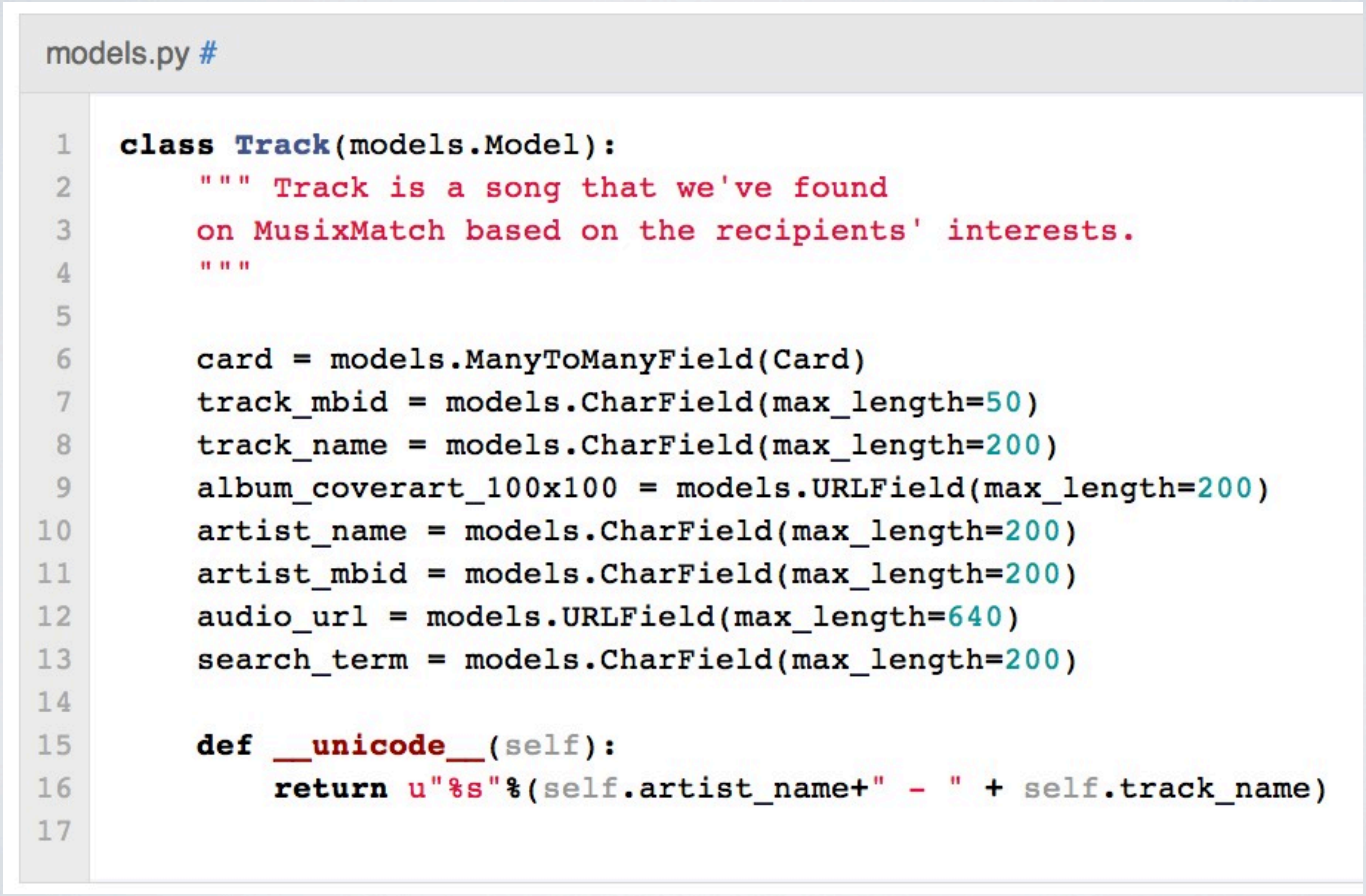
CARD CLASS



models.py # embed raw

```python
class Card(models.Model):
    """ Card is a valentine's day card that contains the information
    about who the card is from and who it's to, what the recipients
    interests are and a personal note.
    """
    user = models.ForeignKey(User)
    recipient_name = models.CharField(max_length=200, blank=True)
    recipient_email = models.EmailField(max_length=200, null=True, blank=True)
    recipient_phone = models.CharField(max_length=200, blank=True)
    intro_note = models.TextField(max_length=1000, blank=True)
    interests = models.TextField(max_length=1000, blank=True)
    create_date = models.DateTimeField(auto_now_add=True)

    def __unicode__(self):
        return u"%s"%("Card"+str(self.id)+" from " + self.user.first_name + \
                      " to " + self.recipient_name)

```

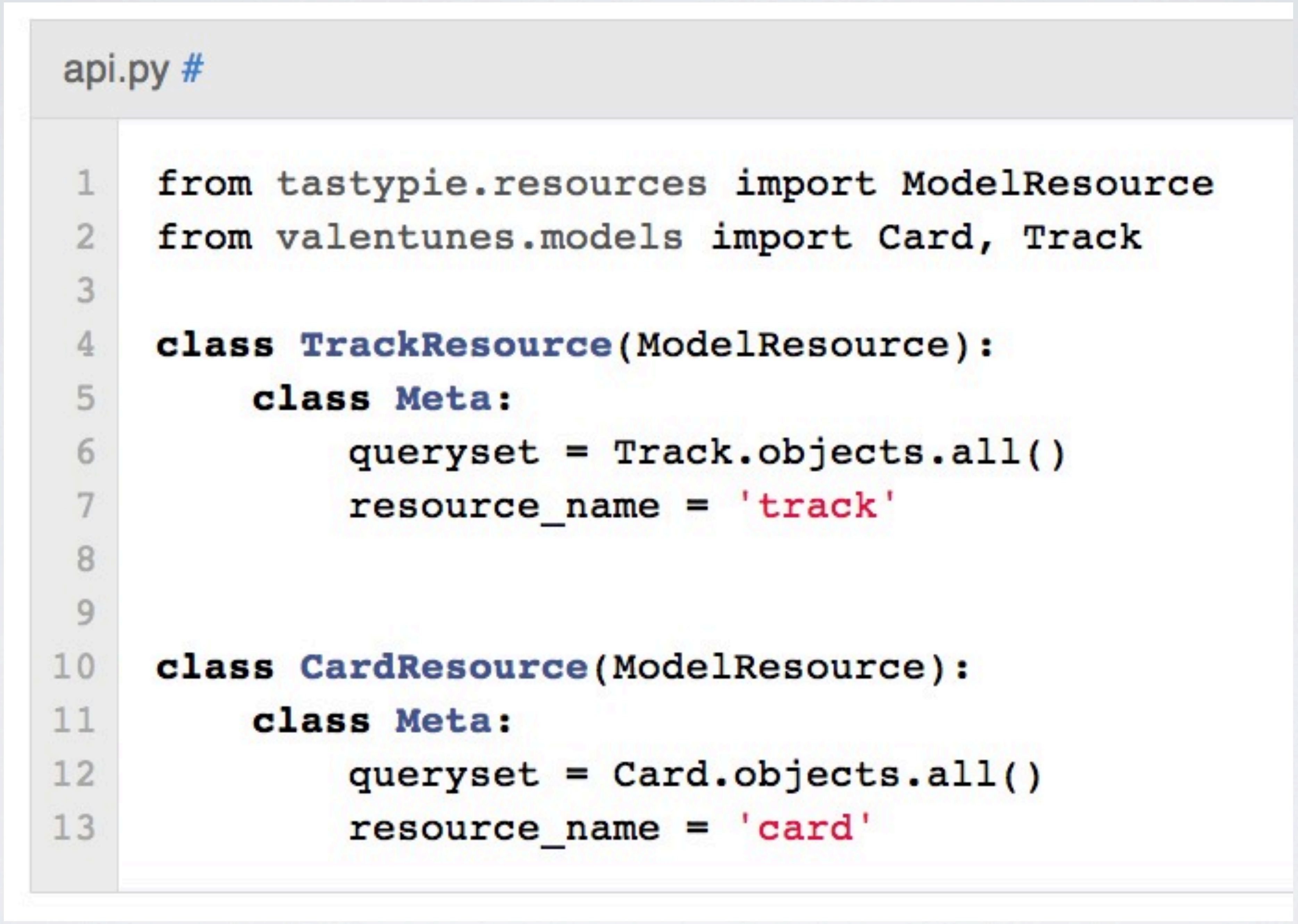
TRACK CLASS



models.py #

```
class Track(models.Model):
    """ Track is a song that we've found
    on MusixMatch based on the recipients' interests.
    """

    card = models.ManyToManyField(Card)
    track_mbid = models.CharField(max_length=50)
    track_name = models.CharField(max_length=200)
    album_coverart_100x100 = models.URLField(max_length=200)
    artist_name = models.CharField(max_length=200)
    artist_mbid = models.CharField(max_length=200)
    audio_url = models.URLField(max_length=640)
    search_term = models.CharField(max_length=200)

    def __unicode__(self):
        return u"%s"%(self.artist_name+" - " + self.track_name)

```

SIMPLE REST API

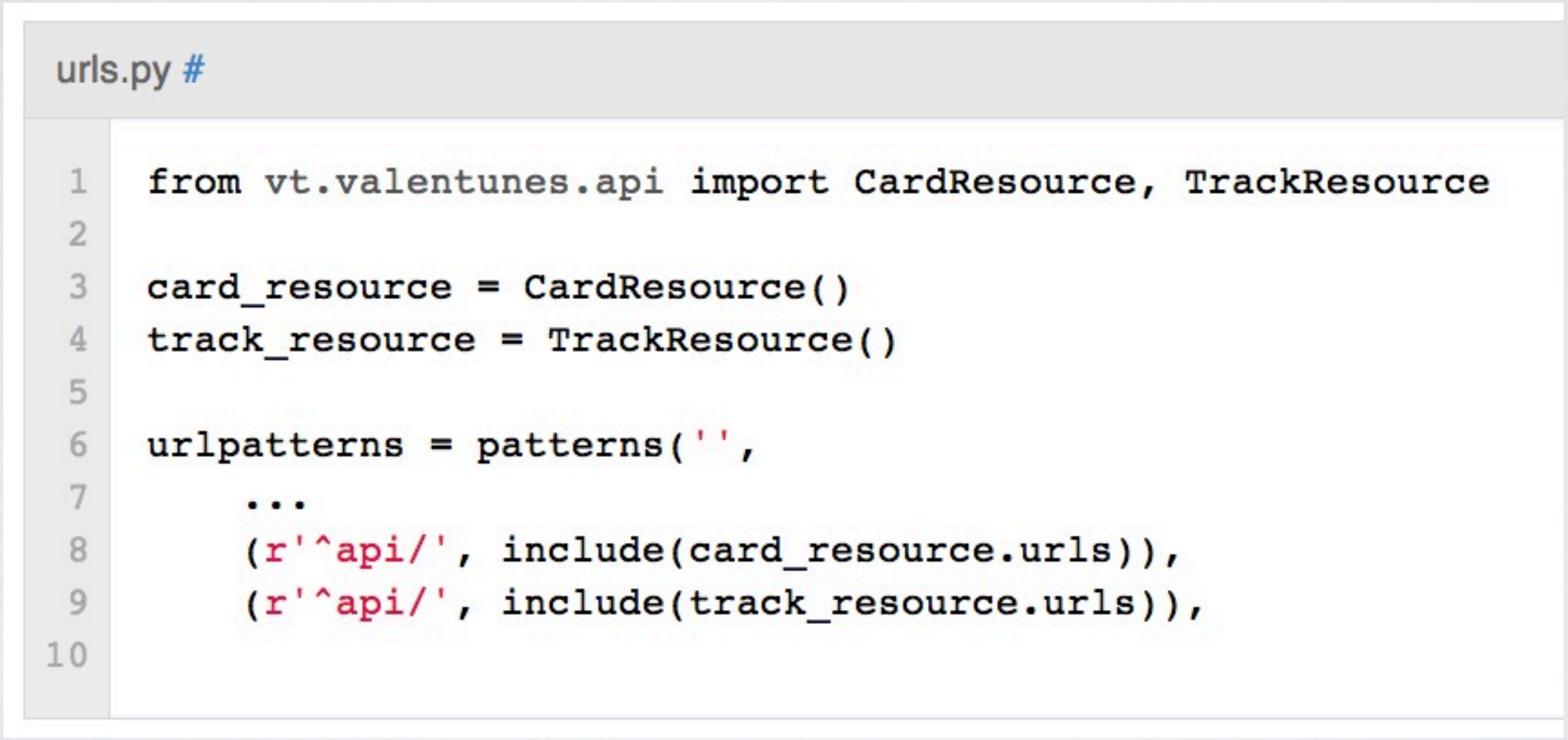


api.py #

```python
from tastypie.resources import ModelResource
from valentunes.models import Card, Track

class TrackResource(ModelResource):
    class Meta:
        queryset = Track.objects.all()
        resource_name = 'track'


class CardResource(ModelResource):
    class Meta:
        queryset = Card.objects.all()
        resource_name = 'card'
```

Basic CRUD operations via API.

URLS.PY



urls.py #

```python
from vt.valentunes.api import CardResource, TrackResource

card_resource = CardResource()
track_resource = TrackResource()

urlpatterns = patterns('',
    ...
    (r'^api/', include(card_resource.urls)),
    (r'^api/', include(track_resource.urls)),
```

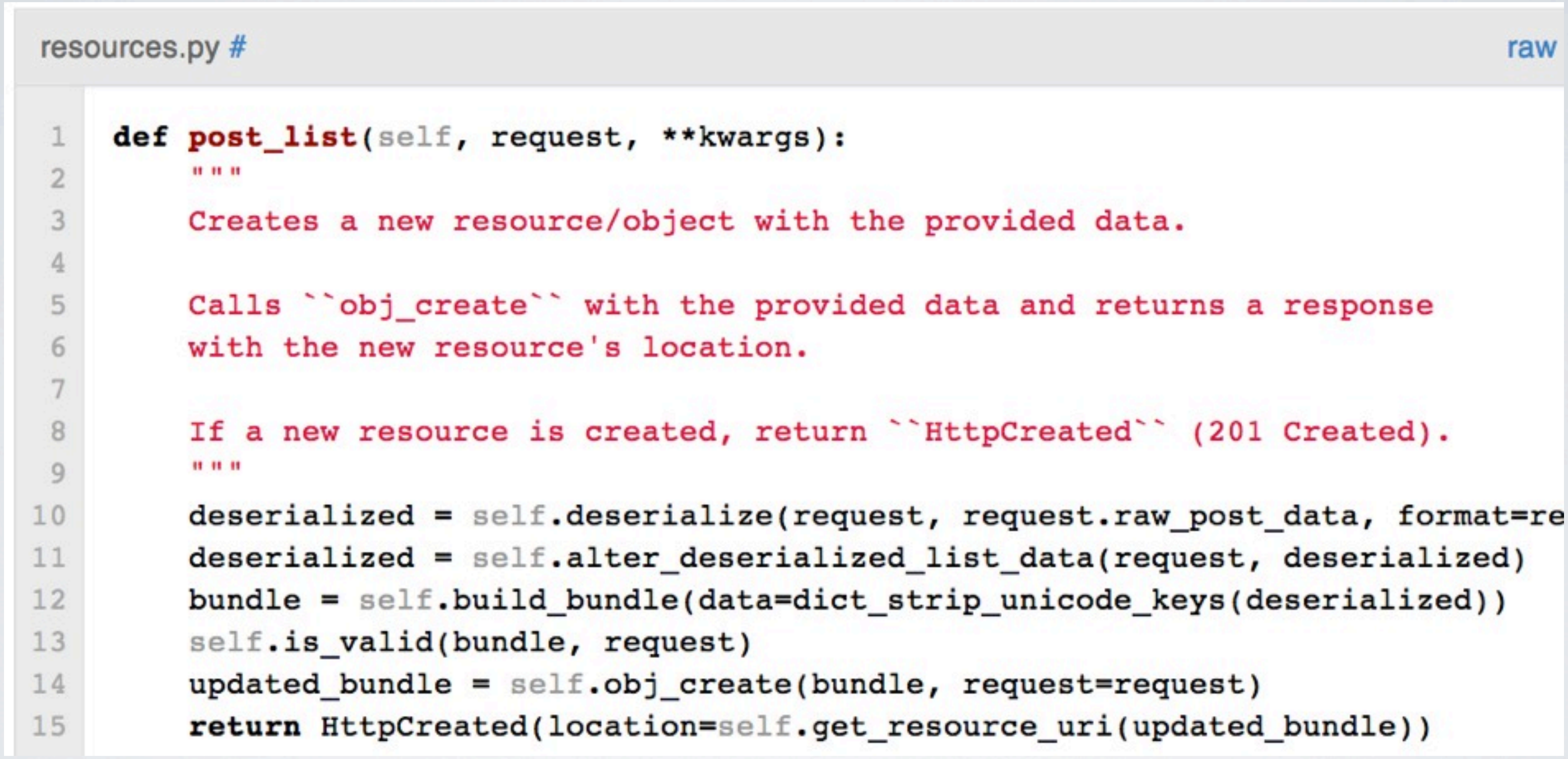
Now access cards and tracks with /api/card/ and /api/track/

CREATE A CARD

```
$ curl -X POST -H 'Content-Type: application/json' -u nate:nate
       --data '{"recipient_name" : "Anna",
                "interests" : "dancing, coffee"}'
       http://localhost:8000/api/card/
```

```
{
    "create_date": "2011-06-06T06:41:31.454924",
     "id": "1",
     "interests": "dancing, coffee",
     "intro_note": "",
     "recipient_email": "",
     "recipient_name": "Anna",
     "recipient_phone": "",
     "resource_uri": "/api/card/1/"
}
```

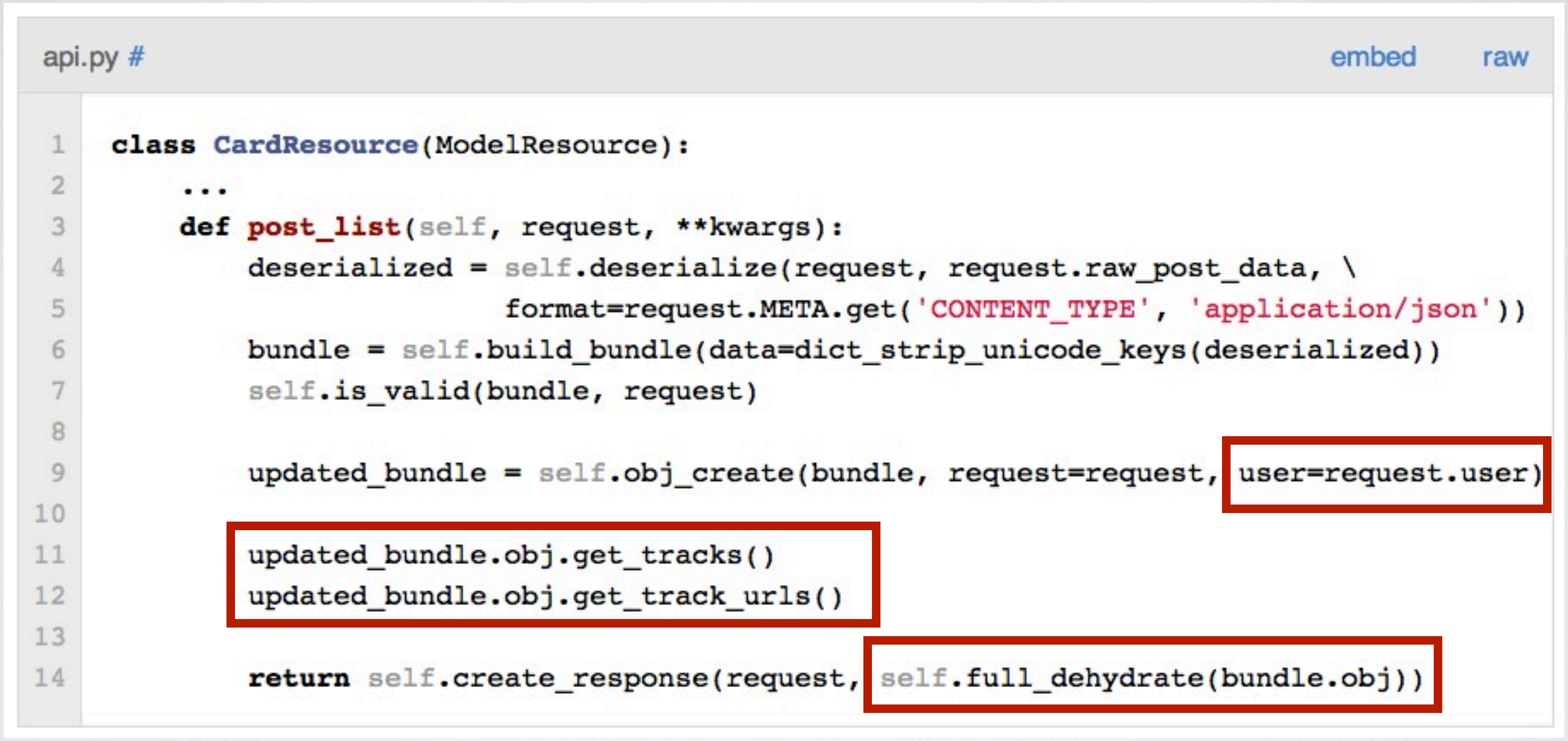
WHAT ABOUT TRACKS?



```
def post_list(self, request, **kwargs):
    """
    Creates a new resource/object with the provided data.

    Calls ``obj_create`` with the provided data and returns a response
    with the new resource's location.

    If a new resource is created, return ``HttpCreated`` (201 Created).
    """
    deserialized = self.deserialize(request, request.raw_post_data, format=re
    deserialized = self.alter_deserialized_list_data(request, deserialized)
    bundle = self.build_bundle(data=dict_strip_unicode_keys(deserialized))
    self.is_valid(bundle, request)
    updated_bundle = self.obj_create(bundle, request=request)
    return HttpCreated(location=self.get_resource_uri(updated_bundle))
```

Default post_list from resources.py (create object via POST)

WHAT ABOUT TRACKS?

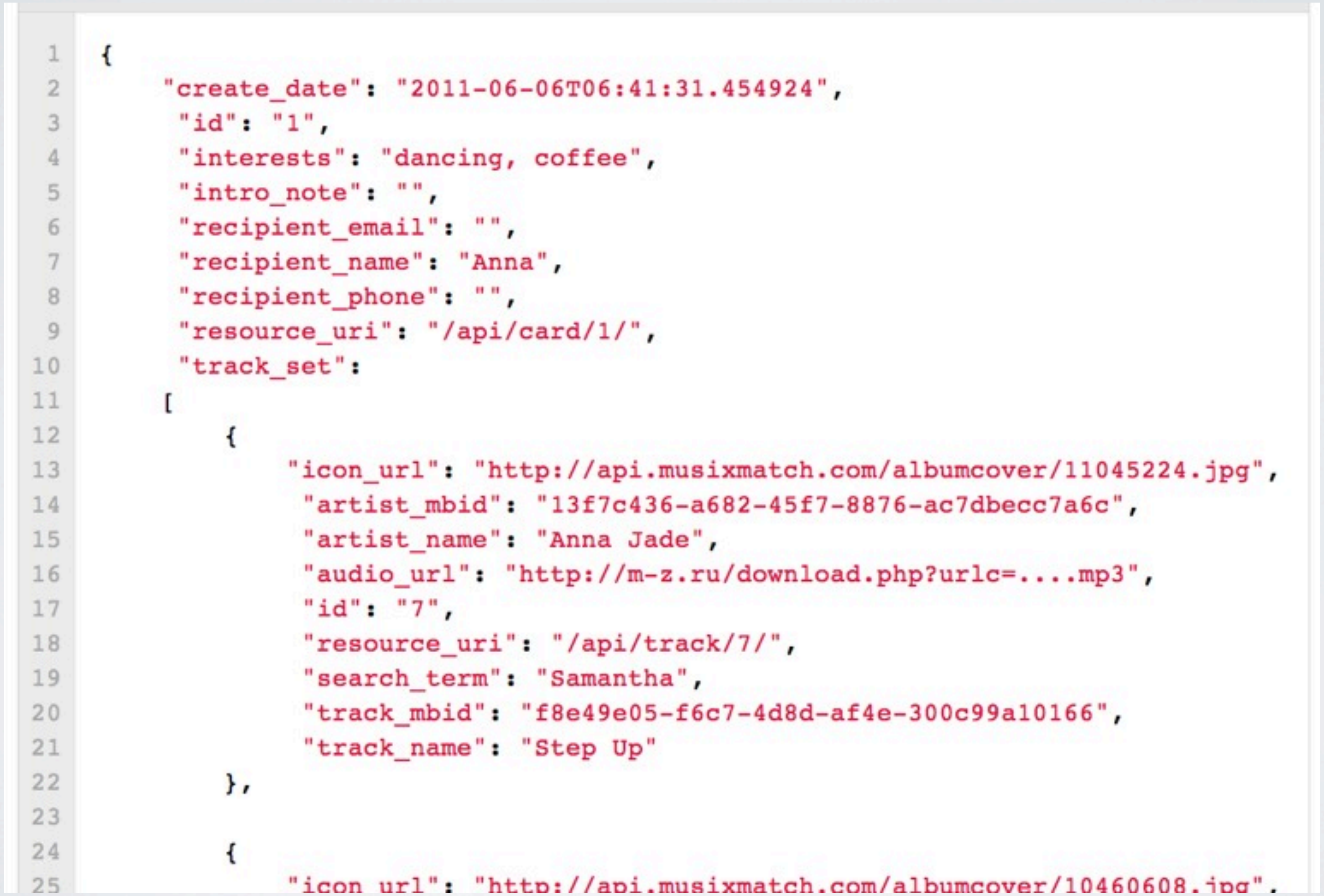


api.py # embed raw

```python
class CardResource(ModelResource):
    ...
    def post_list(self, request, **kwargs):
        deserialized = self.deserialize(request, request.raw_post_data, \
                            format=request.META.get('CONTENT_TYPE', 'application/json'))
        bundle = self.build_bundle(data=dict_strip_unicode_keys(deserialized))
        self.is_valid(bundle, request)

        updated_bundle = self.obj_create(bundle, request=request, user=request.user)

        updated_bundle.obj.get_tracks()
        updated_bundle.obj.get_track_urls()

        return self.create_response(request, self.full_dehydrate(bundle.obj))
```

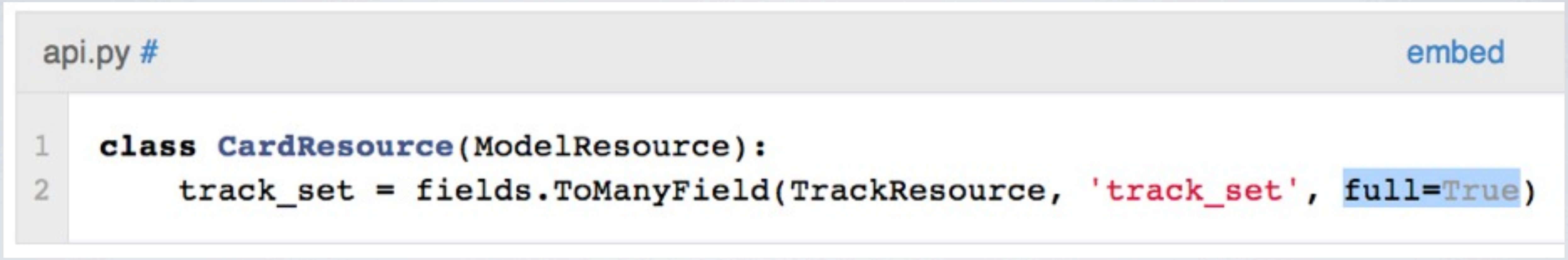
api.py post_list (override method)

MOBILE WANTS HIERARCHICAL DATA



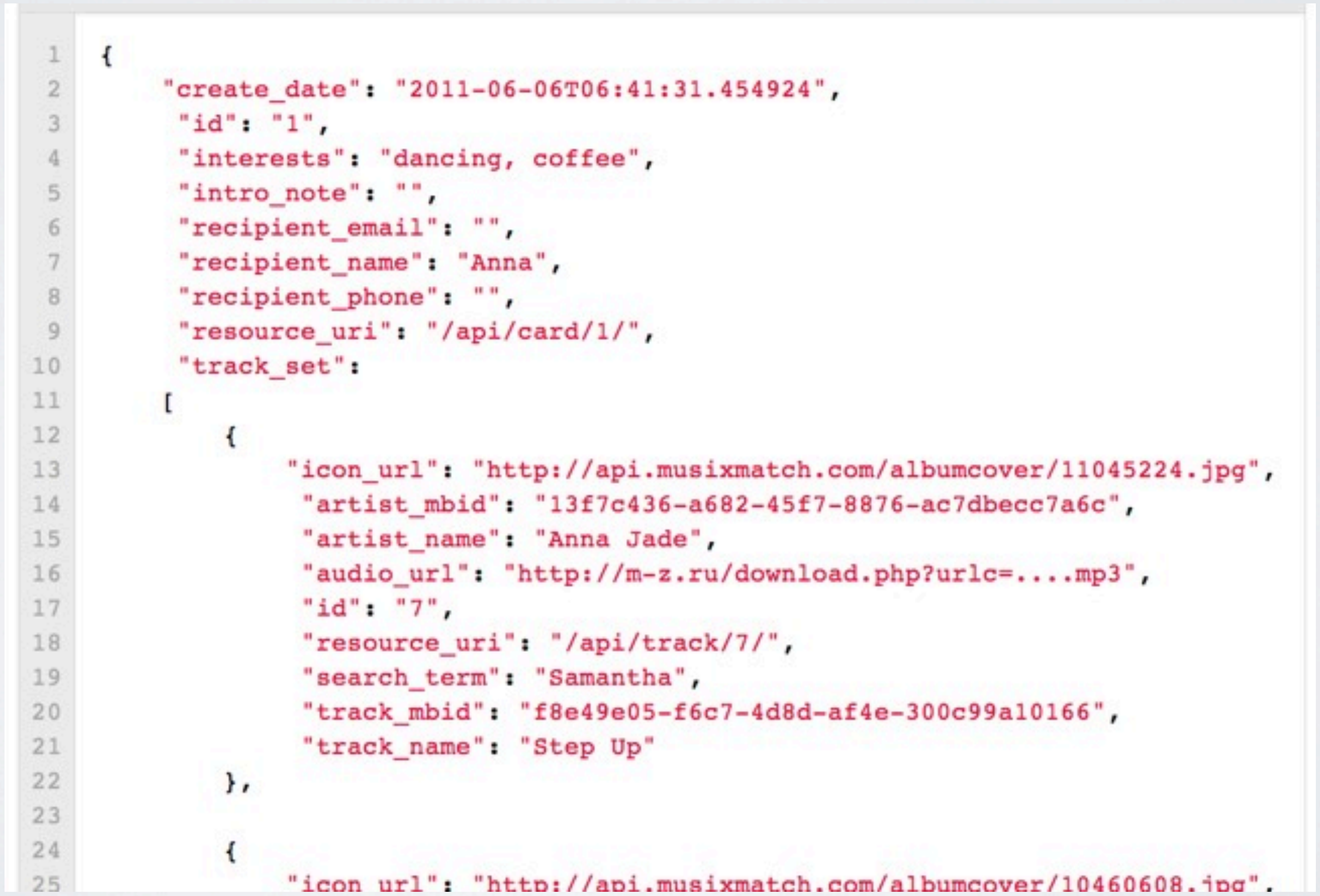
```
{
    "create_date": "2011-06-06T06:41:31.454924",
     "id": "1",
     "interests": "dancing, coffee",
     "intro_note": "",
     "recipient_email": "",
     "recipient_name": "Anna",
     "recipient_phone": "",
     "resource_uri": "/api/card/1/",
     "track_set":
    [
        {
            "icon_url": "http://api.musixmatch.com/albumcover/11045224.jpg",
             "artist_mbid": "13f7c436-a682-45f7-8876-ac7dbecc7a6c",
             "artist_name": "Anna Jade",
             "audio_url": "http://m-z.ru/download.php?urlc=....mp3",
             "id": "7",
             "resource_uri": "/api/track/7/",
             "search_term": "Samantha",
             "track_mbid": "f8e49e05-f6c7-4d8d-af4e-300c99a10166",
             "track_name": "Step Up"
        },

        {
            "icon_url": "http://api.musixmatch.com/albumcover/10460608.jpg",
```

TASTYPIE MAKES IT EASY



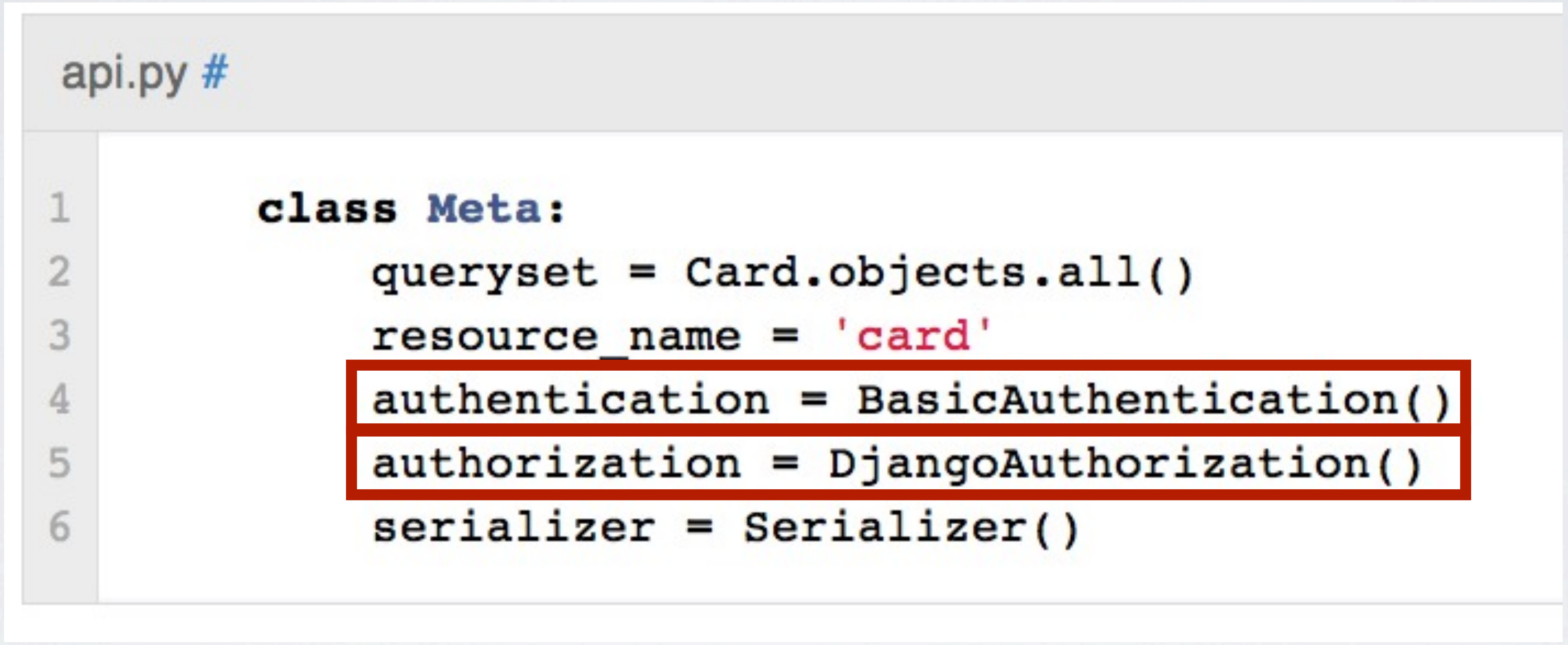
api.py # embed

```python
class CardResource(ModelResource):
    track_set = fields.ToManyField(TrackResource, 'track_set', full=True)
```



```json
{
    "create_date": "2011-06-06T06:41:31.454924",
     "id": "1",
     "interests": "dancing, coffee",
     "intro_note": "",
     "recipient_email": "",
     "recipient_name": "Anna",
     "recipient_phone": "",
     "resource_uri": "/api/card/1/",
     "track_set":
    [
        {
            "icon_url": "http://api.musixmatch.com/albumcover/11045224.jpg",
             "artist_mbid": "13f7c436-a682-45f7-8876-ac7dbecc7a6c",
             "artist_name": "Anna Jade",
             "audio_url": "http://m-z.ru/download.php?urlc=....mp3",
             "id": "7",
             "resource_uri": "/api/track/7/",
             "search_term": "Samantha",
             "track_mbid": "f8e49e05-f6c7-4d8d-af4e-300c99a10166",
             "track_name": "Step Up"
        },

        {
            "icon_url": "http://api.musixmatch.com/albumcover/10460608.jpg",
```

AUTHENTICATION & AUTHORIZATION

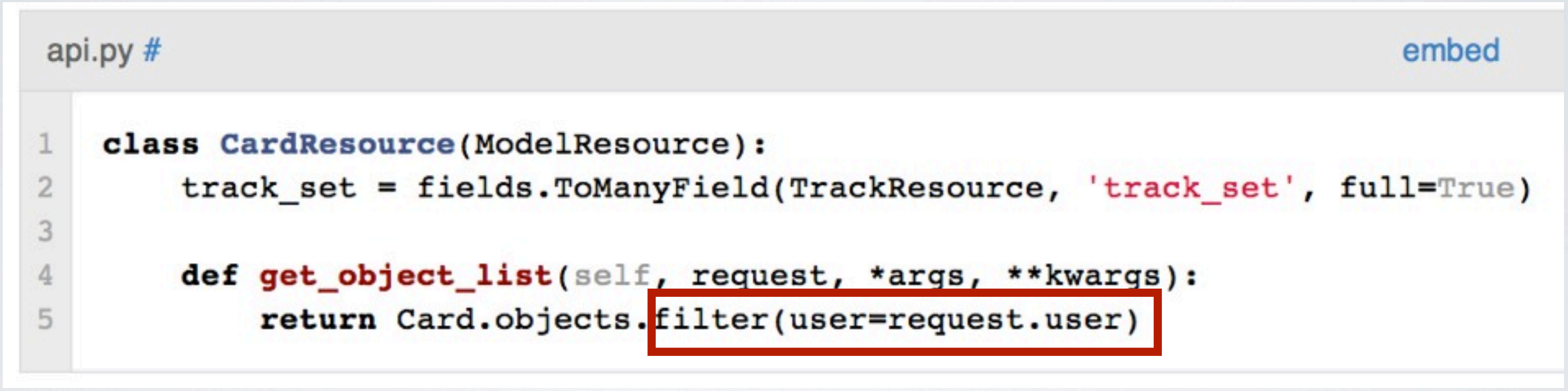


api.py #

```
class Meta:
    queryset = Card.objects.all()
    resource_name = 'card'
    authentication = BasicAuthentication()
    authorization = DjangoAuthorization()
    serializer = Serializer()
```

Authentication - let the user in the door
Authorization - what the user can see

LIMITING BY USER


api.py # embed

```python
class CardResource(ModelResource):
    track_set = fields.ToManyField(TrackResource, 'track_set', full=True)

    def get_object_list(self, request, *args, **kwargs):
        return Card.objects.filter(user=request.user)
```

Returns only the objects owned by the current user.

ERROR CODES

Ideally your API should:

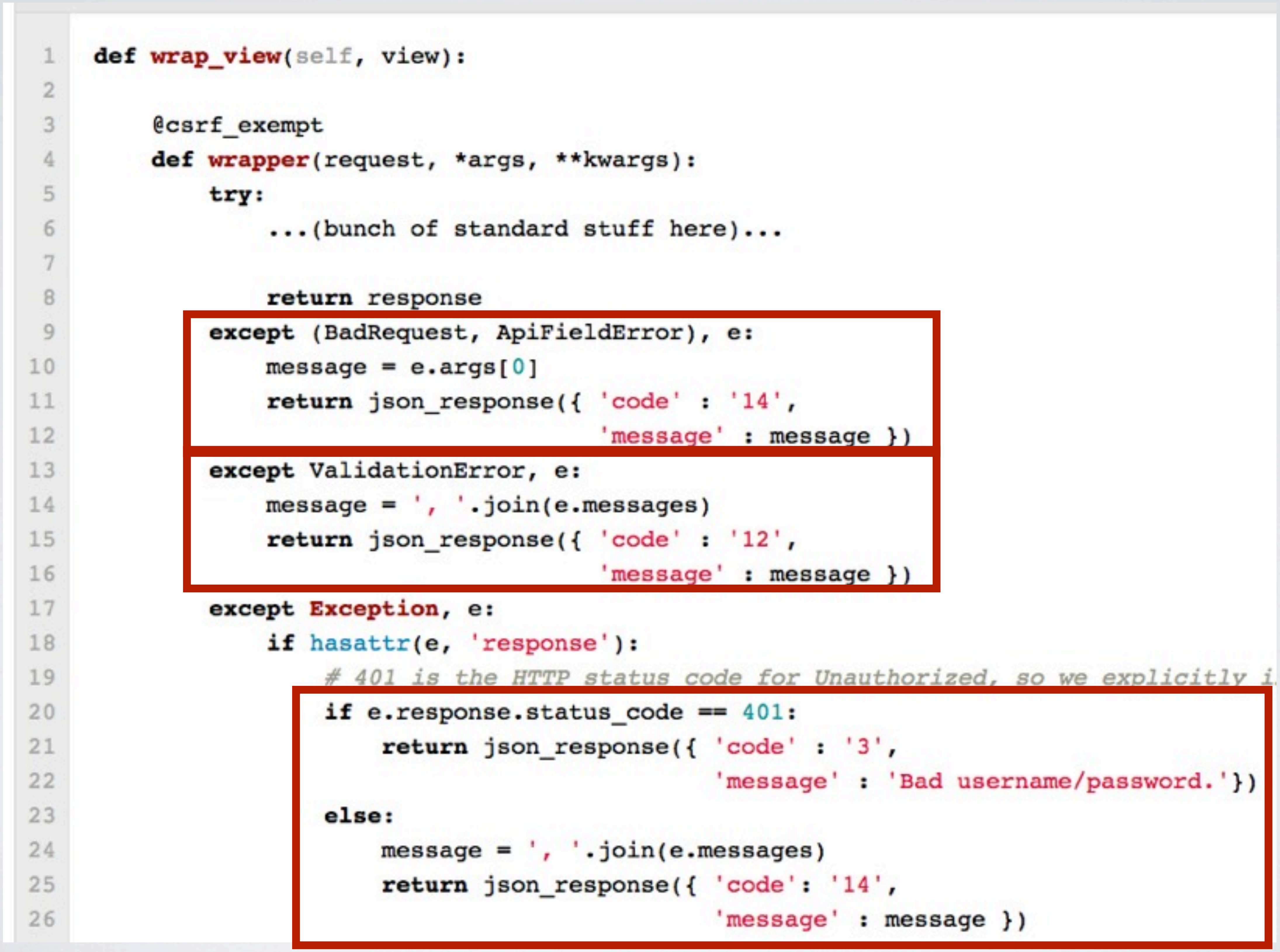
- Never return HTML
- Tailor response codes to actions on device
- Return messages designed for the end user
    - Don't forget the App Store
- Never, ever return HTML

TAILOR ERROR CODES TO ACTIONS ON DEVICE

```
code 200 or 201 = success

code -10 = show alert; include user message

code -20 = show type x alert; log message

code -30 = don't alert user, but send certain info to the server

code -40 = try again

code -50 = push a web view and point it to this url
```

(a very simple example)

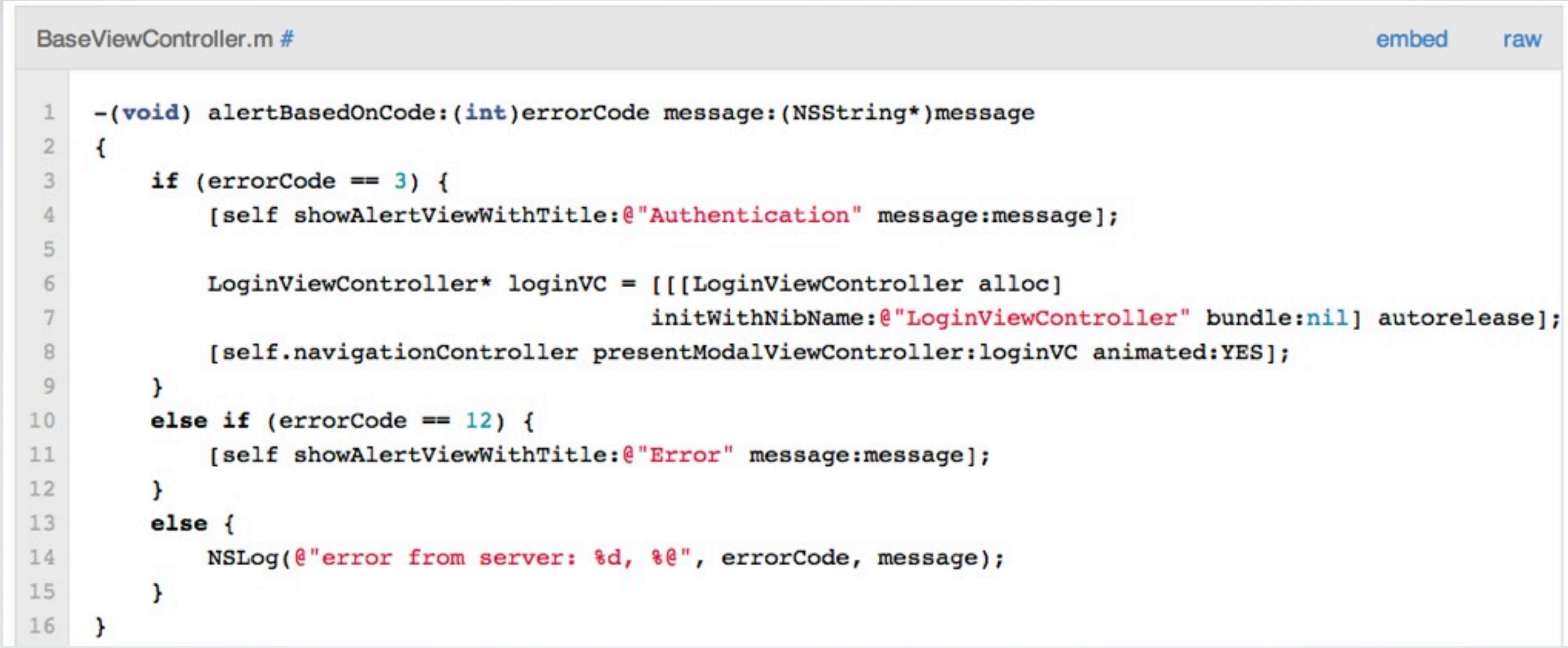
JSON ERROR RESPONSES



```python
def wrap_view(self, view):

    @csrf_exempt
    def wrapper(request, *args, **kwargs):
        try:
            ...(bunch of standard stuff here)...

            return response
        except (BadRequest, ApiFieldError), e:
            message = e.args[0]
            return json_response({ 'code' : '14',
                                   'message' : message })
        except ValidationError, e:
            message = ', '.join(e.messages)
            return json_response({ 'code' : '12',
                                   'message' : message })
        except Exception, e:
            if hasattr(e, 'response'):
                # 401 is the HTTP status code for Unauthorized, so we explicitly i
                if e.response.status_code == 401:
                    return json_response({ 'code' : '3',
                                           'message' : 'Bad username/password.'})
                else:
                    message = ', '.join(e.messages)
                    return json_response({ 'code': '14',
                                           'message' : message })
```

override wrap_view from resources.py

JSON ERROR HANDLING

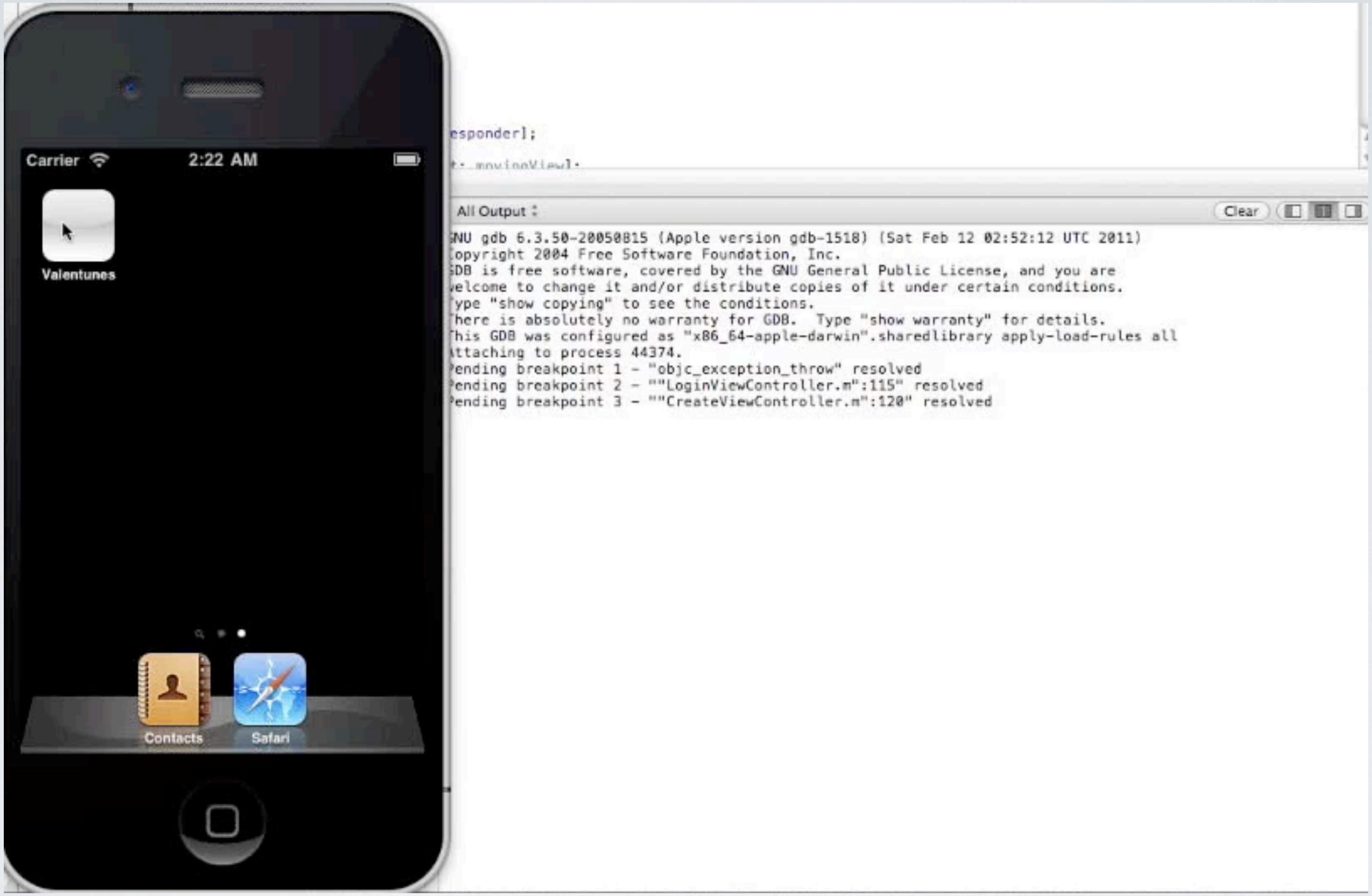


BaseViewController.m # embed raw

```
-(void) alertBasedOnCode:(int)errorCode message:(NSString*)message
{
    if (errorCode == 3) {
        [self showAlertViewWithTitle:@"Authentication" message:message];

        LoginViewController* loginVC = [[[LoginViewController alloc]
                                        initWithNibName:@"LoginViewController" bundle:nil] autorelease];
        [self.navigationController presentModalViewController:loginVC animated:YES];
    }
    else if (errorCode == 12) {
        [self showAlertViewWithTitle:@"Error" message:message];
    }
    else {
        NSLog(@"error from server: %d, %@", errorCode, message);
    }
}
```

Objective-C in Xcode

http://www.youtube.com/watch?v=maZxd8K7Tjc



```
esponder];
t: movingView];
```

```
GNU gdb 6.3.50-20050815 (Apple version gdb-1518) (Sat Feb 12 02:52:12 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".sharedlibrary apply-load-rules all
Attaching to process 44374.
Pending breakpoint 1 - "objc_exception_throw" resolved
Pending breakpoint 2 - ""LoginViewController.m":115" resolved
Pending breakpoint 3 - ""CreateViewController.m":120" resolved
```

ACCEPT ARRAYS OF DATA

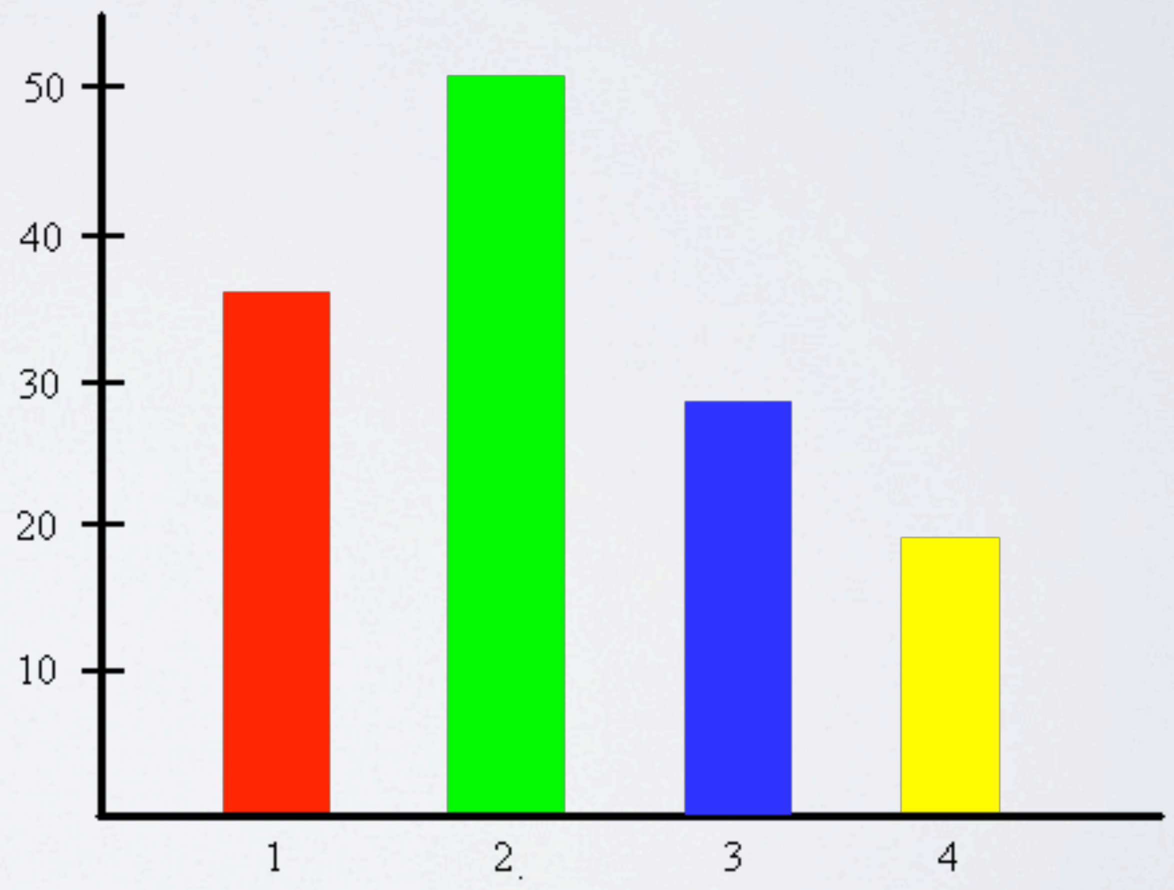
- User enters tunnel
- User uses your app
- User closes your app
- User exits tunnel

ACCEPT ARRAYS OF DATA

```
{
     "pointevents":
     [
       {
          "datetime":"2011-06-19 14:33:02",
          "level":"1",
          "points":"92"
       }
     ],
     "awards":
     [
       {
          "datetime":"2011-06-19 14:29:44",
          "award":"12",
        }
     ]
}
```

PRE-CALCULATED DATA OR DATA NOT STORED ON DEVICE

- No other users are stored on device
- Leaderboards or other calculated user data must come from the server

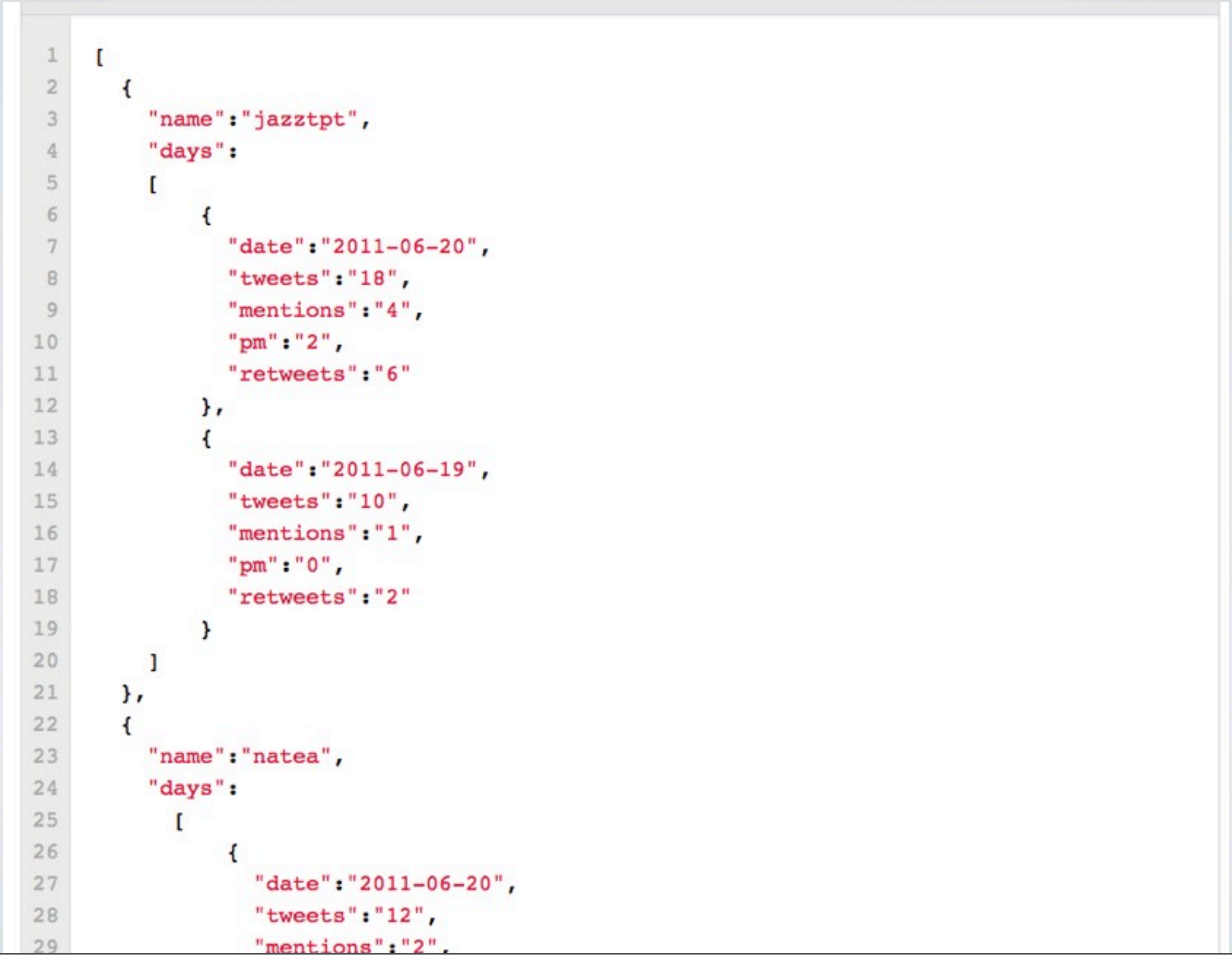


TWITTER LEADERBOARD

- Compete with your friends
- Leaderboard shows daily statistics
- Best returned json for mobile:

```
[array of users containing username and
    [array of days containing num tweets, mentions, etc]
]
```

LEADERBOARD JSON



```
[
  {
    "name":"jazztpt",
    "days":
    [
        {
          "date":"2011-06-20",
          "tweets":"18",
          "mentions":"4",
          "pm":"2",
          "retweets":"6"
        },
        {
          "date":"2011-06-19",
          "tweets":"10",
          "mentions":"1",
          "pm":"0",
          "retweets":"2"
        }
    ]
  },
  {
    "name":"natea",
    "days":
      [
          {
            "date":"2011-06-20",
            "tweets":"12",
            "mentions":"2",
```

WHEN THIS DOESN'T APPLY

- Large data sets -- only expose what client needs
- Multiple third-party clients
    - Allow client to set depth level
    - Create a few special expected api calls

```
api/card/?depth=1
```

or send in json package, or send in the accept header

THANK YOU!

- Music Hack Day Accomplices: Matt Katz, Alexandre Passant, Jeff Novich, Twom Deryckere
- Danielzilla (Daniel Lindsley) - TastyPie
- IsaacKelly
- DjangoCon

VALENTUNES

- Valentunes (Django code)
  https://github.com/natea/valentunes
- Valentunes (iPhone code)
  https://github.com/jazztpt/Valentunes_iPhone
- Valentunes (Twilio integration)
  https://github.com/terraces/valentunes-twilio

DJANGO API FRAMEWORKS

- TastyPie documentation (the one we used)
  http://readthedocs.org/docs/django-tastypie/en/latest/
- django-piston
  https://bitbucket.org/jespern/django-piston/
- Django REST framework
  http://django-rest-framework.org

QUESTIONS?

- Blog post with more detail on mobile api design: http://www.annacallahan.com/blog/2011/06/24/mobile-api-design/
- Anna Callahan:
  annacallahan.com
  @jazztpt
- Nate Aune:
  djangozoom.com
  @natea