RESTful Services in Python on Google App Engine

Ryan Morlok

Docalytics

Overview

- What is REST?
- Why JSON?
- HTTP Considerations
- One example, three ways
  - Raw Webapp2
  - Google Cloud Endpoints
  - Protopy

http://documents.morlok.net/devfest-2015

What Make REST Different?

- **RE**presentational **S**tate **T**ransfer
- An approach, not a standard
- REST is **noun** oriented while RPC is **verb** oriented
- URIs
- HTTP Verbs
- A pragmatic spectrum exists

JSON

- **J**ava**S**cript **O**bject **N**otation
- Invented by Douglas Crockford in in the early-2000's
- Based on the object literal format in JavaScript

JSON versus XML

- Data interchange format
- Simple
- Maps naturally to native data types
- Human Readable/Writable
- Limited data types

- Document format
- Formal (schema definitions, namespaces, etc)
- Transforms
- XPath/Xquery
- Extensible

CRUD via HTTP

Create	→	POST
Read	→	GET
Update	→	PUT, PATCH
Delete	→	DELETE

PUT versus PATCH

- Idempotent
- Set the resource as desired
- Can also be used for create

- Idempotent
- Set sub-properties of the resource
- Only used for update
- Absent Property != null

HTTP Status Codes

Code	Meaning	Used in
200	OK	Successful GETs, PUTs, PATCHes
201	Created	Successful POSTs
204	No Content	Successful DELETEs
301	Moved Permanently	
302	Found	
400	Bad Request	Invalid query parameters or request body
403	Forbidden	Access denied to resource
404	Not Found	Cannot find resource
410	Gone	
418	I'm a teapot	This is the real meaning of this code
500	Internal Server Error	You screwed up
501	Not implemented	Valid resource, method not supported

Tools

- Fiddler
- Postman REST Client
- cUrl
- Python requests library

Entities

Team:

- Name
- Mascot
- Colors

Player:

- Team
- Name
- Position

URL Structure

GET	/v1/teams
POST	/v1/teams
GET	/v1/teams/<team_id>
PUT	/v1/teams/<team_id>
PATCH	/v1/teams/<team_id>
DELETE	/v1/teams/<team_id>
GET	/v1/teams/<team_id>/players
POST	/v1/teams/<team_id>/players
GET	/v1/teams/<team_id>/players/<player_id>
PUT	/v1/teams/<team_id>/players/<player_id>
PATCH	/v1/teams/<team_id>/players/<player_id>
DELETE	/v1/teams/<team_id>/players/<player_id>

NDB Entities

```
from google.appengine.ext import ndb


class Team(ndb.Model):
 name = ndb.StringProperty()
 colors = ndb.StringProperty(repeated=True)
 mascot = ndb.StringProperty()

class Player(ndb.Model):
 team = ndb.KeyProperty()
 name = ndb.StringProperty()
 position = ndb.StringProperty()
```

JSON Structure

Team

```
{
  "id": "...",
  "name": "Minnesota",
  "mascot": "Gopher",
  "colors": [
    "maroon",
    "gold"
  ]
}
```

Player

```
{
  "id": "...",
  "name": "Kyle Rau",
  "position": "Defense",
  "team_id": ”...",
}
```

JSON + WEBAPP2

http://documents.morlok.net/devfest-2015

Monkey Patch Webapp2 for PATCH

```
import webapp2

allowed_methods = webapp2.WSGIApplication.allowed_methods
new_allowed_methods = allowed_methods.union(('PATCH',))
webapp2.WSGIApplication.allowed_methods =
new_allowed_methods
```

Define Endpoints

```
class TeamHandler(webapp2.RequestHandler):
    …
    def get_team(self, team_id):
        self.response.content_type = 'application/json'
        self.response.write(json.dumps(team_to_dict(
               get_team_or_404(team_id))))

app = webapp2.WSGIApplication([
    …
    webapp2.Route(r'/v1/teams/<team_id>',
                  methods=['GET'],
                  handler='main.TeamHandler:get_team',
                  name='get_team')
    …
])
```

Raise Exceptions For Error Statuses

```
def get_team_or_404(team_id):
    t = ndb.Key(urlsafe=team_id).get()

    if not t:
        raise webapp2.exc.HTTPNotFound()

    if not isinstance(t, Team):
        raise webapp2.exc.HTTPNotFound()

    return t
```

Team: Entity ➔ Dict ➔ JSON

```
def team_to_dict(team):
    d = team.to_dict()
    d['id'] = team.key.id() if team.key else None
    return d


# json.dumps(team_to_dict(Team(name="Minnesota",
      mascot="Gopher", colors=["maroon", "gold"])))
# => {
  "id": null,
  "colors": ["maroon", "gold"],
  "name": "Minnesota",
  "mascot": "Gopher”
}
```

Player: Entity ➔ Dict ➔ JSON

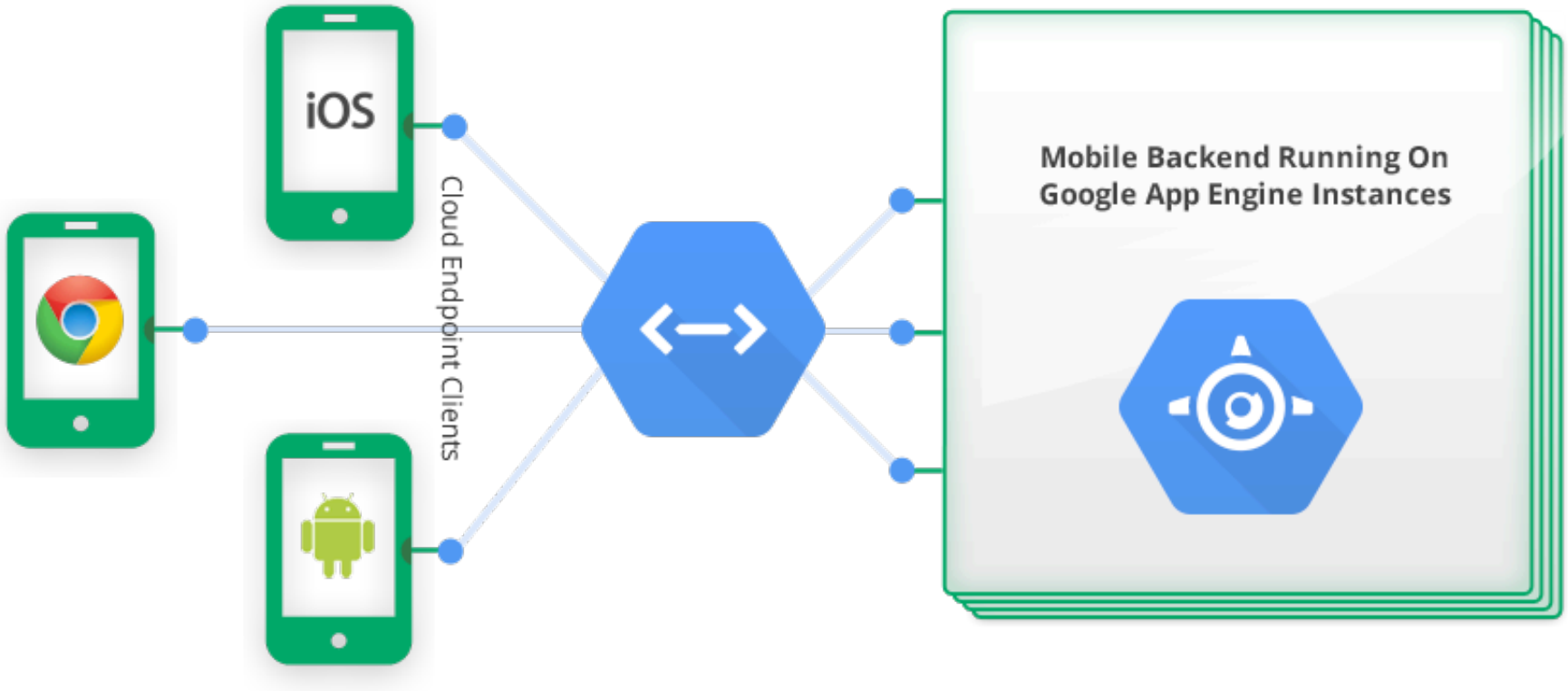
```
def player_to_dict(player):
  return {
    'id': player.key.id() if player.key else None,
    'name': player.name,
    'position': player.position,
    'team_id': player.team.id() if player.team
                                else None
    }

# json.dumps(main.player_to_dict(
    main.Player(name="Kyle Rau", position="Forward",
                team=t.key)))

# => {"position": "Forward", "team_id": 1,
      "id": null, "name": "Kyle Rau"}
```

Set Headers, Status, Write Response

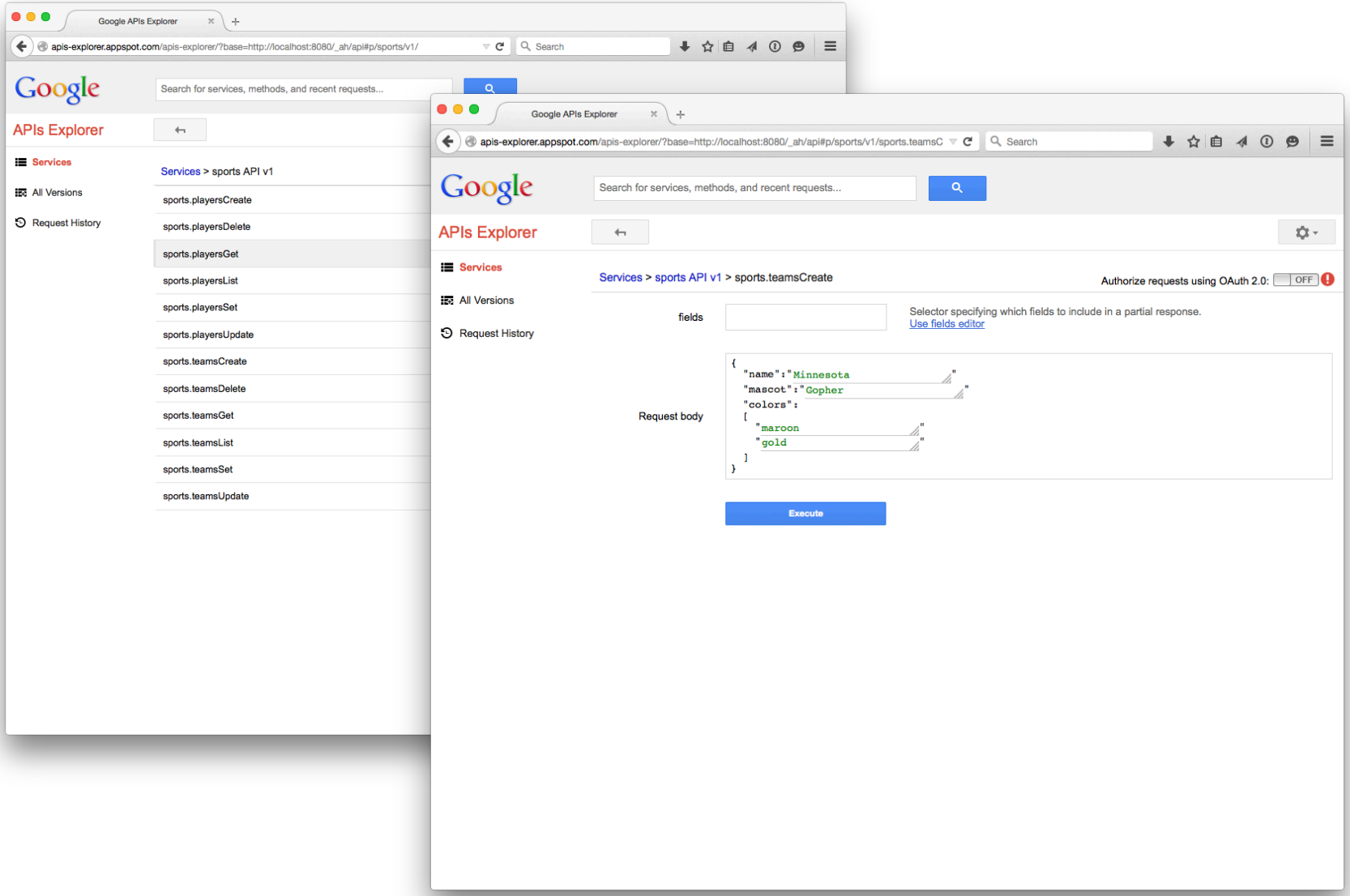
```
class TeamHandler(webapp2.RequestHandler):
  …
  def create_team(self):
    team = None
    try:
      team_dict = json.loads(self.request.body)
      team = dict_to_team_set(team_dict)
      team.put()
    except:
      raise webapp2.exc.HTTPBadRequest()

    self.response.headers['Location'] = webapp2.uri_for(
                    'get_team', team_id=team.key.urlsafe())
    self.response.status_int = 201
    self.response.content_type = 'application/json'
    self.response.write(json.dumps(team_to_dict(team)))
```

GOOGLE CLOUD ENDPOINTS

http://documents.morlok.net/devfest-2015


iOS
Cloud Endpoint Clients
Mobile Backend Running On
Google App Engine Instances

API Explorer



Define Messages

```
from protorpc import messages, message_types, remote

class TeamMessage(messages.Message):
    id = messages.StringField(1)
    name = messages.StringField(2)
    colors = messages.StringField(3, repeated=True)
    mascot = messages.StringField(4)

class PlayerMessage(messages.Message):
    id = messages.StringField(1)
    name = messages.StringField(2)
    position = messages.StringField(3)
    team_id = messages.StringField(4)
```

Annotate Endpoints

```
@endpoints.api(name='sports',
               version='v1',
               description='Sports API')
class SportsAPI(remote.Service):

    @endpoints.method(message_types.VoidMessage,
                      TeamsResponseMessage,
                      name='teamsList',
                      path='teams',
                      http_method='GET')
    def get_teams(self, request):
        response = TeamsResponseMessage()
        response.teams = []

        for team in Team.query().iter():
            response.teams.append(team_to_msg(team))

        return response
```

All Data Needs a Wrapper

```
TEAM_UPDATE = endpoints.ResourceContainer(
    TeamMessage,
    team_id=messages.StringField(2, required=True))

class SportsAPI(remote.Service):
    @endpoints.method(TEAM_UPDATE,
                      TeamMessage,
                      name='teamsSet',
                      path='teams/{team_id}',
                      http_method='PUT')
    def set_team(self, request):
        team = get_team_or_404(request.team_id)
        team = msg_to_team_set(request, team=team)
        team.put()

        return team_to_msg(team)
```

PATCH Not Possible

```
def msg_to_player_update(msg, player=None):
    player = player or Player()

    if msg.name is not None:
        player.name = msg.name

    if msg.position is not None:
        player.position = msg.position

    if msg.team_id is not None:
        team_id = msg.team_id
        player.team = ndb.Key(urlsafe=team_id)

    return player
```

PROTOPY

http://documents.morlok.net/devfest-2015

Why ProtoPy

- Messages are good, but need to be simplified
- Fully support PATCH
- Maintain control of HTTP
- Control URL structure
- Support for unstructured data

Messages

```
class TeamMessage(messages.Message):
    id = messages.StringField()
    name = messages.StringField()
    colors = messages.StringField(repeated=True)
    mascot = messages.StringField()


class PlayerMessage(messages.Message):
    id = messages.StringField()
    name = messages.StringField()
    position = messages.StringField()
    team_id = messages.StringField()
```

Annotate Endpoints

```
class PlayerHandler(webapp2.RequestHandler):
    @protopy.endpoint
    @protopy.query.string('position')
    def get_players(self, team_id, position):
        _ = get_team_or_404(team_id)
        response = PlayersResponseMessage()
        response.players = []

        q = Player.query(Player.team ==
                         ndb.Key(urlsafe=team_id))

        if position:
            q = q.filter(Player.position == position)

        for player in q.iter():
            response.players.append(player_to_msg(player))

        return response
```

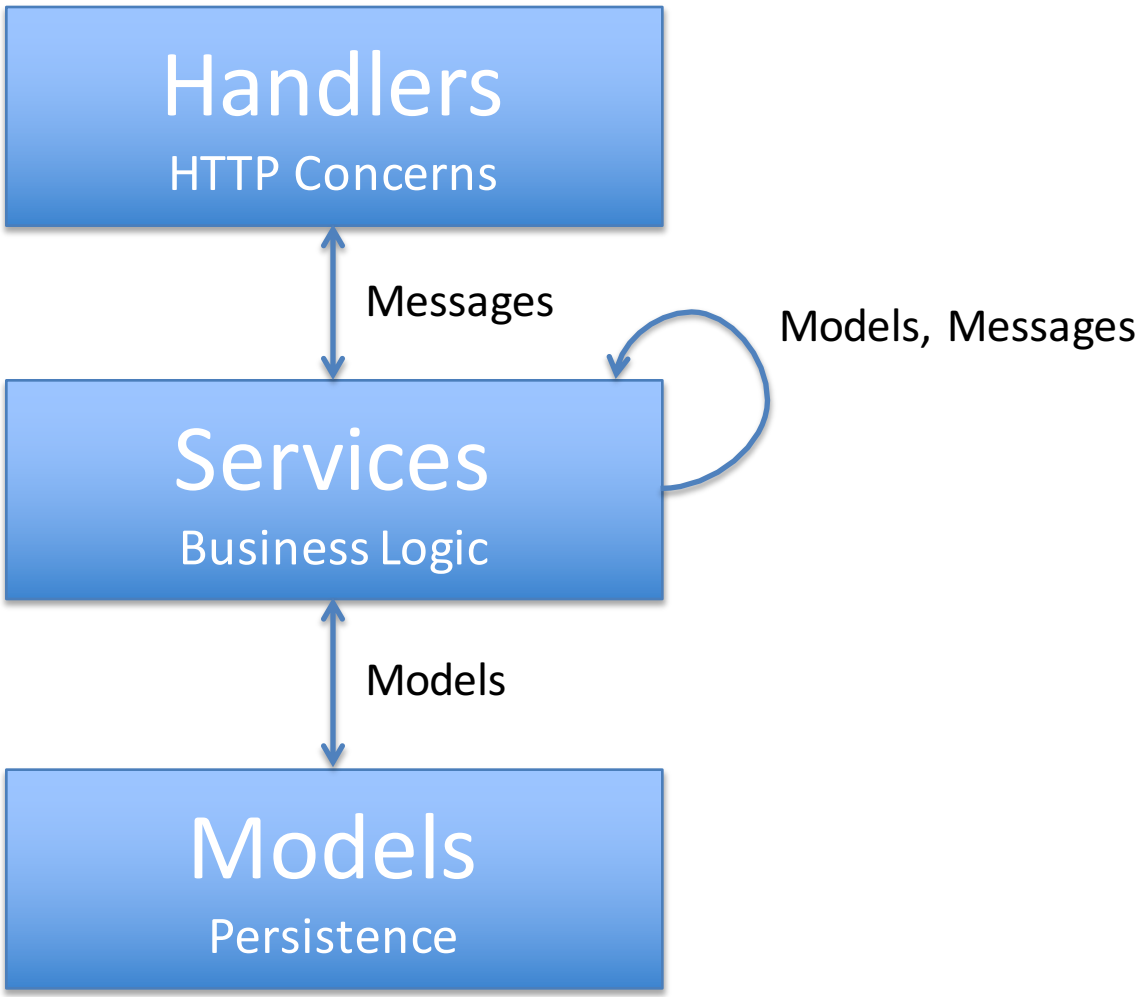
Control HTTP

```
class PlayerHandler(webapp2.RequestHandler):
    @protopy.endpoint(player_details=PlayerMessage)
    def create_player(self, team_id, player_details):
        _ = get_team_or_404(team_id)
        player_details.team_id = team_id
        player = msg_to_player_set(player_details)
        player.put()

        return 201, player_to_msg(player),
              {'Location': webapp2.uri_for('get_player',
         team_id=team_id, player_id=player.key.urlsafe())}
```

CLOSING THOUGHTS

http://documents.morlok.net/devfest-2015

Code Structure



Tips & Tricks

- Special IDs
- RPC methods off of resources when it makes more sense
- Think hard before using nested resources
- Prefer PATCH to PUT

Thank You

@rmorlok

ryan.morlok@morlok.com

https://github.com/rmorlok