Moxie API Documentation

Release 0.1

Mobile Oxford team, IT Services, University of Oxford

February 03, 2017

Contents

CHAPTER 1

HTTP API

1.1 API specification

1.1.1 Formats

The API returns HAL+JSON at the moment (see the HAL specification).

1.1.2 HAL+JSON

Responses have a *_links* attribute containing links to help in the navigation (e.g. when results need pagination).

Each individual entity has a *self* attribute in *_links* that represents the path to itself.

It is highly recommended for clients to use these links (Relations) to navigate between resources.

1.1.3 CORS

We use CORS (Cross-origin resource sharing). JSONP is not available (causes problem e.g. no custom header).

1.1.4 Pagination

Standard parameters are available for pagination: start and count. It is advised to use Relations (first, last, prev and next) to browse results.

1.1.5 Errors

Error messages are provided as JSON, the key *description* gives a technical error message. In addition to the message, you should check the HTTP response code which will give you an idea of the problem (4xx vs. 5xx...).

1.2 Authentication

Authentication within Moxie API's is done with HMAC. This means HTTP requests on these API's must be made with specific headers. Before going into detail on how these requests can be build here is a brief glossary of terms used when discussing Authentication.

HMAC keyed-Hash Message Authentication Code: a code generated by calculating a hashing function in combination with a secret key.

Canonical representation What is being hashed. String representation of the request being made.

Shared secret Generally a string issued by the service owner to grant access to an API. This is used as the hashing key by both parties to successfully authenticate the user. **NEVER** sent visibly over the wire.

API Key Sent over the wire to identify a request with a particular user.

1.2.1 Step 1. Building the "Canonical Representation"

As described above the canonical representation is what is being hashed by our hashing function to generate the HMAC. Within Moxie the canonical representation of a HTTP request looks like this:

```
{method}
{url}
{headers}
```

All of the canonical representation is in **lowercase**.

Where `method` is the HTTP method being invoked for example get, post, put, delete. `url` is the absolute URL being requested. `headers` is the only tricky part. Some of the HTTP request headers are included in the canonical representation. For Moxie we require the `Date` header and a special header, the `X-HMAC-Nonce` (in the canonical representation both of these will be lowercase of course). They are included in the following format:

```
date:{date}
x-hmac-nonce:{nonce}
```

The values of these headers is the responsibility of the client making the request. However we recommend following the HTTP spec and include a `Date` in the standard format, such as "Wed, 15 Nov 2013 06:25:24 GMT". Perhaps more important is the usage of a good Cryptographic Nonce. Good options for a nonce are pseudo-random numbers.

Here is a complete example of a "canonical representation":

```
POST
http://localhost:5000/notifications/alert
date:Wed, 15 Nov 2013 06:25:24 GMT
x-hmac-nonce:29582
```

Attention: There is not a new line character "\n" after the headers.

1.2.2 Step 2. Generating the HMAC

As mentioned before HMAC requires the use of a hashing function. We use the `SHA-1` hashing function.

This step depends on your toolset, find your preferable hmac function and call it with the `SHA-1` algorithm and your shared secret as the hash key and the canonical representation built above as the message to be hashed. Here are a few possible ways to run this

- Python - hmac
- PHP - hash_hmac
- OpenSSL

Using whatever tools appropriate you need just the hexidemical representation of the digest.

1.2.3 Step 3. Making the request

Set the `Authorization` HTTP header on your request to the hexidecimal digest value from the previous step. Also set a custom `X-Moxie-Key` header to the value of your API key, this is used to identify your request. Your complete HTTP request should look similar to this:

```
POST /alert HTTP/1.1
Host: api.m.ox.ac.uk
X-Moxie-Key: d51459b5-d634-48f7-a77c-d87c77af37f1
X-HMAC-Nonce: 12642
Date: Fri, 10 Jan 2014 11:49:55 GMT
Authorization: ccd61eddf0c6be849b13e524c171e4c14a0d571f
Content-Type: application/json
```

Should your request fail and you receive a 401. There should be a `WWW-Authenticate` header which will provide a "reason" why your request failed. This also describes how to make requests on the API:

```
WWW-Authenticate: HMACDigest realm="HMACDigest Moxie", reason="missing header: HTTP_AUTHORIZATION", a
```

So this request failed because they missed the `Authorization` header.

Endpoints

1.3 Places endpoint

Endpoint to search and retrieve information about places.

All the responses are conform to the HAL specification.

GET /places/(**string:** *id*) **[,**
string: *id* **...]** Get details of one or multiple places by their ID

Example request:

```
GET /places/oxpoints:23232339 HTTP/1.1
Host: api.m.ox.ac.uk
Accept: application/json
```

Example request for multiple POIs:

The response representation for multiple POIs will be equivalent to the search method.

```
GET /places/osm:3646652,oxpoints:23232392 HTTP/1.1
Host: api.m.ox.ac.uk
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "_embedded": {
    "files": [
      {
        "location": "oxpoints/54559254/depiction/original/primary.jpg",
        "primary": true,
        "type": "depiction",
        "url": "//mox-static-files.oucs.ox.ac.uk/oxpoints/54559254/depiction/original/primary.jp
```

```
        },
      ]
    },
    "_links": {
      "child": [
        {
          "href": "/places/oxpoints:32320005",
          "title": "Balliol College Library",
          "type": [
            "/university/library"
          ],
          "type_name": [
            "Library"
          ]
        }
      ],
      "primary_place": {
        "href": "/places/oxpoints:23232339",
        "title": "Balliol College",
        "type": [
          "/university/college"
        ],
        "type_name": [
          "College"
        ]
      },
      "self": {
        "href": "/places/oxpoints:23232339"
      }
    },
    "address": "Broad Street OX1 3BJ",
    "distance": 0,
    "id": "oxpoints:23232339",
    "identifiers": [
      "oxpoints:54559254",
      "oxpoints:23232339",
      "obn:851",
      "osm:187925177",
      "oucs:ball",
      "finance:RB"
    ],
    "lat": "51.754425",
    "lon": "-1.257216",
    "name": "Balliol College",
    "name_sort": "Balliol College",
    "shape": "POLYGON ((... 51.755528699999999 0,-1.2584285 51.755520099999998 0))",
    "social_facebook": [
      "https://www.facebook.com/balliolcollege"
    ],
    "social_twitter": [
      "https://www.twitter.com/BalliolOxford"
    ],
    "type": [
      "/university/college"
    ],
    "type_name": [
      "College"
    ],
```

```
    "website": "http://www.balliol.ox.ac.uk/"
}
```

Parameters

- **id** (*string*) – ID of the resource, if multiple resources, separated by a comma

Status Codes

- 200 OK – resource found
- 301 Moved Permanently – redirection to the resource by its main ID
- 404 Not Found – no resource found
- 503 Service Unavailable – Service not available

If multiple resources are requested, as much documents as possible will be returned (i.e. if one of the identifier requested is not found, all other documents will be returned).

GET /places/search

Search for places using full-text search on name, tags and type of place. Also searches in identifiers (e.g. searching "69326473" will return the bus stop corresponding to this Naptan ID). Results can be filtered by a type and its subtypes or can be filtered by specific types (both options cannot be used at the same time). Note that the result might be using a different search as spellchecking is done (e.g. searching for "Wolverkote" will return results with "Wolvercote").

Example request:

```
GET /places/search?q=aldates&type=/transport HTTP/1.1
Host: api.m.ox.ac.uk
Accept: application/json
Geo-Position: 0.232, 51.347
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "_embedded": {
    "pois": [
      {
        "_links": {
          "child": [
            {
              "href": "/places/atco:340000004H5",
              "title": "Stop H5 St Aldates",
              "type": [
                "/transport/bus-stop"
              ],
              "type_name": [
                "Bus stop"
              ]
            },
          ],
          "parent": {
            "href": "/places/stoparea:340G00004000",
            "title": "Oxford City Centre",
            "type": [
```

```
                "/transport/stop-area"
              ],
              "type_name": [
                "Bus stop area"
              ]
            },
            "self": {
              "href": "/places/stoparea:340G00003140"
            }
          },
          "distance": 0,
          "id": "stoparea:340G00003140",
          "identifiers": [
            "stoparea:340G00003140"
          ],
          "lat": "51.7508834555",
          "lon": "-1.2571120376",
          "name": "St Aldates",
          "name_sort": "St Aldates",
          "type": [
            "/transport/stop-area"
          ],
          "type_name": [
            "Bus stop area"
          ]
        },
        {
          "_links": {
            "curie": {
              "href": "http://moxie.readthedocs.org/en/latest/http_api/rti.html#{type}",
              "name": "rti",
              "templated": true
            },
            "parent": {
              "href": "/places/stoparea:340G00003140",
              "title": "St Aldates",
              "type": [
                "/transport/stop-area"
              ],
              "type_name": [
                "Bus stop area"
              ]
            },
            "rti:bus": {
              "href": "/places/atco:340000004H5/rti/bus",
              "title": "Live bus departure times"
            },
            "self": {
              "href": "/places/atco:340000004H5"
            }
          },
          "distance": 0,
          "id": "atco:340000004H5",
          "identifiers": [
            "atco:340000004H5",
            "naptan:69326543"
          ],
          "lat": "51.7502787977",
```

```
            "lon": "-1.2567597994",
            "name": "Stop H5 St Aldates",
            "name_sort": "Stop H5 St Aldates",
            "type": [
              "/transport/bus-stop"
            ],
            "type_name": [
              "Bus stop"
            ]
          },
        ]
      },
      "_links": {
        "curies": [
          {
            "href": "http://moxie.readthedocs.org/en/latest/http_api/relations/{rel}.html",
            "name": "hl",
            "templated": true
          },
          {
            "href": "http://moxie.readthedocs.org/en/latest/http_api/relations/facet.html",
            "name": "facet"
          }
        ],
        "hl:first": {
          "href": "/places/search?q=aldates&facet=type&type=%2Ftransport&count=35"
        },
        "hl:last": {
          "href": "/places/search?q=aldates&facet=type&type=%2Ftransport&count=35"
        },
        "hl:types": [
          {
            "count": 10,
            "href": "/places/search?q=aldates&facet=type&type=%2Ftransport%2Fbus-stop",
            "name": "/transport/bus-stop",
            "title": [
              "Bus stop"
            ],
            "value": "/transport/bus-stop"
          },
          {
            "count": 1,
            "href": "/places/search?q=aldates&facet=type&type=%2Ftransport%2Fstop-area",
            "name": "/transport/stop-area",
            "title": [
              "Bus stop area"
            ],
            "value": "/transport/stop-area"
          }
        ],
        "self": {
          "href": "/places/search?q=aldates&facet=type&type=%2Ftransport&count=35&start=0"
        }
      },
      "query": "aldates",
      "size": 11
    }
```

Query Parameters

- **q** – what to search for
- **type** – filter by a specific type in the hierarchy of types (will search within subtypes too)
- **type_exact** – filter by exact types (as opposite to the type parameter), you can have this parameter multiple times.
- **start** – first result to retrieve
- **count** – number of results to retrieve
- **lat** – latitude (as an alternative to the Geo-Position header if spatial search required)
- **lon** – longitude (as an alternative to the Geo-Position header if spatial search required)
- **inoxford** – only get results within Oxford (value will be ignored)
- **university_only** – only get results from the University (value will be ignored)
- **exclude_university** – exclude results from the University (value will be ignored) i.e. only amenities, transport...

Accessibility filtering

Below are filters specific to accessibility features, coming from the university' access guide.

Query Parameters

- **accessibility_has_adapted_furniture** – only get POIs known to have "adapted furniture"
- **accessibility_has_cafe_refreshments** – only get POIs known to have "accessible cafe / refreshments"
- **accessibility_has_computer_access** – only get POIs known to have "accessible computer access"
- **accessibility_has_hearing_system** – only get POIs known to have a "hearing system"
- **accessibility_has_lifts_to_all_floors** – only get POIs known to have "lift access to all floors"
- **accessibility_has_quiet_space** – only get POIs known to have "accessible quiet space"
- **accessibility_has_accessible_toilets** – only get POIs known to have "accessible toilets"
- **accessibility_has_accessible_parking_spaces** – only get POIs known to have "accessible parking spaces"

Application specific filtering

Below are filters made specifically for an application. It is not recommended to use these parameters, as it is mainly experimental and might change to be generic in the future.

Query Parameters

- **is_display_in_maps_department_list** – only get POIs manually selected / curated as "featured" university departments

If no geolocation is passed (either by header or query parameters), and if there is no full-text search (`q` parameter), the result will be sorted by name (A-Z).

Status Codes

- 200 OK – query found
- 400 Bad Request – Bad request (could happen if some parameters are used in combination e.g. type and type_exact)
- 503 Service Unavailable – Service not available

GET /places/types
Display a list of types.

Status Codes

- 200 OK – display a list of types

GET /places/suggest
Suggest places based on name and alternative names. Results can be filtered by specific types.

Example request:

```
GET /places/suggest?q=sec&type_exact=/university/department HTTP/1.1
Host: api.m.ox.ac.uk
Accept: application/json
```

Example response:

The response only contains a subset of properties available in the search method to reduce the length of the response.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "_embedded": {
    "pois": [
      {
        "_links": {
          "self": {
            "href": "/places/oxpoints:23233517"
          }
        },
        "address": "Wellington Square OX1 2JD",
        "distance": 0,
        "id": "oxpoints:23233517",
        "identifiers": [],
        "name": "Council Secretariat",
        "type": [
          "/university/department"
        ],
        "type_name": [
          "Department"
        ]
      },
      {
        "_links": {
          "self": {
            "href": "/places/oxpoints:59801811"
          }
        },
        "address": "Parks Road OX1 3QD",
        "distance": 0,
```

```
            "id": "oxpoints:59801811",
            "identifiers": [],
            "name": "Cyber Security Centre",
            "type": [
              "/university/department"
            ],
            "type_name": [
              "Department"
            ]
          },
          {
            "_links": {
              "self": {
                "href": "/places/oxpoints:58455192"
              }
            },
            "address": "off South Parks Road OX1 3RQ",
            "distance": 0,
            "id": "oxpoints:58455192",
            "identifiers": [],
            "name": "Oxford University Security Services",
            "type": [
              "/university/department"
            ],
            "type_name": [
              "Department"
            ]
          },
        ]
      },
      "_links": {
        "curies": [
          {
            "href": "http://moxie.readthedocs.org/en/latest/http_api/relations/{rel}.html",
            "name": "hl",
            "templated": true
          }
        ],
        "hl:first": {
          "href": "/places/suggest?q=sec&count=20"
        },
        "hl:last": {
          "href": "/places/suggest?q=sec&count=20"
        },
        "self": {
          "href": "/places/suggest?q=sec&count=20&start=0"
        }
      },
      "query": "sec",
      "size": 13
    }
```


Query Parameters

- **q** – what to search for
- **type_exact** – filter by exact types (as opposite to the type parameter), you can have this parameter multiple times.

- **start** – first result to retrieve
- **count** – number of results to retrieve

Status Codes

- 200 OK – query found
- 400 Bad Request – Bad request (e.g. missing parameters)
- 503 Service Unavailable – Service not available

1.4 Transport endpoint

Endpoint to retrieve information about transport.

GET /transport/park-and-rides
Get real-time information about status of park and rides

Example request:

```
GET /transport/park-and-rides HTTP/1.1
Host: api.m.ox.ac.uk
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "park_and_rides": [
    {
      "capacity": 1389,
      "identifier": "osm:2809915",
      "name": "Redbridge Park & Ride OX1 4XG",
      "percentage": 0,
      "spaces": 0,
      "unavailable": true   // real-time information not available
    },
    [...]
    {
      "capacity": 758,
      "identifier": "osm:4329908",
      "name": "Water Eaton Park & Ride OX2 8HA",
      "percentage": 48,
      "spaces": 390,
      "unavailable": false
    }
  ]
}
```

Status Codes

- 200 OK – resource found
- 503 Service Unavailable – Service not available

List of standard relations describing links between resources.

1.5 Relations

List of standard relations (rel) generally used in our API. See the HAL specification about link relations.

1.5.1 child

Child document of a resource. This is usually represented as an array.

1.5.2 facet

Return an overview of your search results for a given field by listing the possible values this field may take and the number of results with each value.

For example: `/places/search?q=union&facet=type_exact`. Produces facets something like this:


```
"facet:type_exact": [
  {
    "count": 9,
    "href": "/places/search?q=union&facet=type_exact&type_exact=%2Funiversity%2Fsub-library",
    "name": "/university/sub-library",
    "title": "Sub-library",
    "value": "/university/sub-library"
  },
  {
    "count": 1,
    "href": "/places/search?q=union&facet=type_exact&type_exact=%2Funiversity%2Fsite",
    "name": "/university/site",
    "title": "Site",
    "value": "/university/site"
  },
  {
    "count": 1,
    "href": "/places/search?q=union&facet=type_exact&type_exact=%2Funiversity%2Flibrary",
    "name": "/university/library",
    "title": "Library",
    "value": "/university/library"
  },
  {
    "count": 1,
    "href": "/places/search?q=union&facet=type_exact&type_exact=%2Funiversity%2Fdepartment",
    "name": "/university/department",
    "title": "Department",
    "value": "/university/department"
  }
],
```


1.5.3 first

First page of a result set. This relation is always present.

1.5.4 last

Last page of a result set. This relation is always present.

1.5.5 next

Next page of a result set. This relation is only present if there is a next page (regarding URL parameters *start* and *count*).

1.5.6 parent

Parent document

1.5.7 poi

Point of Interest related to a resource.

This usually refers to a place from the Places endpoint.

1.5.8 prev

Previous page of a result set. This relation is only present if there is a previous page (regarding URL parameters *start* and *count*).

1.5.9 primary_place

In the case of a POI, indicates the primary place of the POI. In some cases, the primary place might be equivalent to self.

1.5.10 rti

Real-Time information attached to a resource.

This is usually referenced in a place from the Places endpoint.

1.5.11 search

Search related to present resource.

List of real-time information resources.

1.6 Real-time Information

Moxie supports an API for providing RTI from Service providers. Currently all RTI is represented in a similar way:

```
{
 "messages": [],
 "services": [],
 "type": '',
 "title": ''
}
```

The contents of this structure is left to the `provider` itself. Here are some guidelines for each key, `messages`, which is used for human readable messages from the RTI provider. For example in the case of rail travel it might be a message informing you of delays on a line due to maintenance work.

`services` should contain the RTI itself. The choice of format is left to the providers to expose in whatever form makes the most sense.

The `type` attribute should be a unique identifier for that RTI representation and the `title` attribute should be a human readable title for the information. Here are examples of RTI representations.

1.6.1 rail-arrivals

Rail live arrivals board information, for example:


```
{
  "messages": [],
  "services": [
    {
      "destination": {
        "location": [
          {
            "crs": "PAD",
            "futureChangeTo": null,
            "locationName": "London Paddington",
            "via": null
          }
        ]
      },
      "eta": "11:54",
      "operator": "First Great Western",
      "operatorCode": "GW",
      "origin": {
        "location": [
          {
            "crs": "GMV",
            "futureChangeTo": null,
            "locationName": "Great Malvern",
            "via": null
          }
        ]
      },
      "platform": "1",
      "serviceID": "kgCLsr4sZvhigTZPO8doPQ==",
      "sta": "11:28"
    },
        ]
      },
      "serviceID": "bMcFsZG5AMloV3FVpYkZpA==",
      "sta": "13:02"
    }
  ],
  "type": "rail-arrivals",
  "title": "Arrivals"
}
```

1.6.2 rail-departures

Rail live departures board information, for example:


```
{
  "messages": [],
  "services": [
    {
      "destination": {
        "location": [
          {
            "crs": "PAD",
            "futureChangeTo": null,
            "locationName": "London Paddington",
            "via": null
          }
        ]
      },
      "etd": "11:55",
      "operator": "First Great Western",
      "operatorCode": "GW",
      "origin": {
        "location": [
          {
            "crs": "GMV",
            "futureChangeTo": null,
            "locationName": "Great Malvern",
            "via": null
          }
        ]
      },
      "platform": "1",
      "serviceID": "kgCLsr4sZvhigTZPO8doPQ==",
      "std": "11:31"
    }
  ],
  "type": "rail-departures",
  "title": "Departures"
}
```


1.6.3 bus

Bus stop current real time information, for example:


```
{
  "messages": [
    "traffic incidents in Oxford some delays to X39/X40 possible<div class=\"stopLine\">-traffic inci
  ],
  "services": [
    {
      "destination": "Didcot & Harwell",
      "following": [],
      "next": "10 mins",
      "service": "X32"
    },
    {
      "destination": "Gloucester Green",
```

```
      "following": [
        "30 mins",
        "55 mins",
        "65 mins",
      ],
      "next": "15 mins",
      "service": "X90"
    },
    {
      "destination": "City Centre",
      "following": [
        "27 mins",
        "41 mins",
        "67 mins",
        "72 mins",
        "82 mins",
        "91 mins",
      ],
      "next": "19 mins",
      "service": "TUBE"
    },
    {
      "destination": "Oxford City Centre",
      "following": [
        "69 mins",
        "126 mins",
        "156 mins",
      ],
      "next": "30 mins",
      "service": "OXF"
    },
    {
      "destination": "Reading",
      "following": [],
      "next": "30 mins",
      "service": "X39"
    },
    {
      "destination": "Reading via W'dcote",
      "following": [],
      "next": "60 mins",
      "service": "X40"
    }
  ],
  "type": "bus",
  "title": "Live bus timetable information"
}
```

CHAPTER 2

Internal API

2.1 Overview

2.1.1 Blueprints

Encapsulates an instance of an Application.

- Domain
- Representation
- Service
- ServiceView
- Provider
- Importer

2.2 Cache

Moxie provides cache using Flask-Cache.

2.2.1 Configuration

See Flask-Cache documentation.

2.2.2 Caching the result of a function

To use it on a view, you have to decorate your method with *@cache.cached* (imported from *moxie.core.cache*) and specify a *timeout* (in seconds).

```
from moxie.core.views import ServiceView
from moxie.core.cache import cache

class NearRealTimeInformation(ServiceView):

    @cache.cached(timeout=10)
    def handle_request(self):
        return {'near': 'real-time'}
```

Warning: This acts on the path of the request only, **not** the arguments of the request. See below for more information on that topic.

2.2.3 Cache key that includes arguments as well

You can customize the cache key used depending on your requirements by passing the keyword argument *key_prefix* to *@cache.cached*.

We provide an helper to create the key from the path of the request **and** the arguments, you can use it by importing *args_cache_key* from *moxie.core.cache*. See example below.

```
from flask import request

from moxie.core.views import ServiceView
from moxie.core.cache import cache, args_cache_key

class LookupView(ServiceView):

    @cache.cached(timeout=60, key_prefix=args_cache_key)
    def handle_request(self):
        identifier = request.args.get('identifier', None)
        return {'lookup': identifier}
```

2.3 Configurator

class moxie.core.configurator.**Configurator**(*app*)

Provides basic configuration within Moxie. Currently we handle our configuration through YAML files.

from_envvar(*envvar*, *silent=False*)

Lifted from *Flask.config*

from_yaml(*yaml_path*, *silent=False*)

Read in the file and parse (safely) as YAML. Update the Flask conf, blueprints, services.

register_blueprints(*blueprints*)

Expects a dictionary of blueprints, something like this:

```
{'courses': {
    'url_prefix': '/courses',
    'factory': 'moxie_courses.create_blueprint'},
}
```

Here the *factory* should point to a *callable* which reflects the following function signature.

create_blueprint(*name*, *conf*) → Flask.blueprint

2.4 Domain

Domain objects are used in the Service layer, they are passed from/to Provider and resources.

2.5 Health checks

Health checks are used to test the status of some services at runtime.

Result is exposed in an URL that can be given to a monitoring solution based on HTTP status code, providing a quick health check of your API.

2.5.1 Configuration

Your configuration file can have a *healthchecks* section that has a dictionary of services to check. See following example.

```
healthchecks:
    Places index:
        moxie.core.search.SearchService:
            backend_uri: 'solr+http://url/solr/places'
    Events index:
        moxie.core.search.SearchService:
            backend_uri: 'solr+http://url/solr/events'
```

2.5.2 Defining an health check on a service

A method *healthcheck* will be called on every service defined in the configuration section *healthchecks*.

This method shouldn't take any argument, and should return a tuple with:

- a boolean value True / False: True if the service answered as expected, else False
- a string that represents a "friendly" message to represent the answer of the service

The code below is an example from a check to the search server Apache Solr.

```
def healthcheck(self):
    try:
        response = requests.get('{url}{core}/{method}'.format(url=self.server_url,
            core=self.core, method=self.methods['healthcheck']), timeout=2,
            config={'danger_mode': True})
        return response.ok, response.json['status']
    except Exception as e:
        return False, e
```

2.5.3 Running health checks

Health checks are available at */_health* and are exposed as a list (in plain text) with the status of each service.

The response has a status code of 200 if all services returned a correct value, otherwise the status code will be 500.

Inspired from Dropwizard Health Checks.

2.6 Importer

Writes data from an external (potentially cached) data source into our data layer. Generally running periodically from Tasks.

2.7 Key-value Store

class moxie.core.kv.**KVService**(*backend_uri*)

Service for accessing a Key-Value store. This is the secondary datastore used within Moxie. General usage should be for caching and as a non-critical data store.

Most development has taken place with redis being used as the KV store. Currently this is the only fully supported KV store, see: moxie.core.kv.SUPPORTED_KV_STORES for details.

__getattr__(*name*)

The KV Service proxies all calls through to the underlying backend. Since we only have one moxie.core.kv.SUPPORTED_KV_STORES for the time being it doesn't make sense to restrict the functionality.

Note: In future we may need to consider this API if we want to have other supported backends. This might involve implementing a compatibility layer.

__init__(*backend_uri*)

__module__ = 'moxie.core.kv'

static _get_backend(*kv_uri*)

Following the same pattern found in *moxie.core.search.SearchService._get_backend()*

Parameters **kv_uri** – URI to the Key-value store for example redis://foo.bar/bucket.

healthcheck()

Healthcheck query to the backend

2.8 Metrics

Moxie provides metrics using Flask-StatsD, a wrapper around Statsd python client.

2.8.1 Configuration

The following configuration variables are available (in the Flask section):

- *STATSD_HOST* hostname of the statsd instance
- *STATSD_PORT* port of statsd (8125 by default)
- *STATSD_PREFIX* prefix to set for all metrics

See Flask-Statsd documentation for more information.

2.8.2 Views timing

All views are automatically timed (from *moxie.core.views*), metrics are sent in the form of <module name>.<view name> e.g. *moxie_events.views.Search*.

2.8.3 E.g. timing some code execution

To time some code execution, you should use a context manager *statsd.timer* (imported from *moxie.core.metrics* and specify the name of the metric.

```
from moxie.core.views import ServiceView
from moxie.core.metrics import statsd

class TimedView(ServiceView):

    def handle_request(self):
        self.expensive_method()
        return {'near': 'real-time'}

    def expensive_method(self):
        with statsd.timer('expensive'):
            # some code
```

2.9 OAuth

class moxie.oauth.services.**OAuth1Service**(*oauth_endpoint*, *client_identifier*, *client_secret*, *request_token_path='request_token'*, *access_token_path='access_token'*, *authorize_path='authorize'*)

Enables using 3-legged authentication with OAuth v1 the Moxie client handles making actual redirections and user interactions.

Note: OAuth1 terminology varies, between the original spec and the formal RFC the terminology is far from consistent. We have tried to follow the RFC where possible however, we use `requests.auth` which uses differing terminology.

Parameters

- **oauth_endpoint** – URL of the form http://service.foo/oauth/
- **client_identifier** – Client token identifier.
- **client_secret** – Shared secret paired with the above identifier.

authorization_url(*token_param='oauth_token'*, *callback_uri=None*)

Convenience method to both generate a new temporary credential and return a URL where a user can continue the OAuth workflow authentication. Always generates new temporary credentials.

authorized

Returns `True` if resource owner credentials are available.

refresh_temporary_credentials(*callback_uri=None*)

Requests new temporary credentials from the OAuth server.

Parameters **callback_uri** – the request is (optionally) signed with this URL and the user should be redirected back here upon completing the OAuth workflow.

signer

Returns a `OAuth1` object which can be used to sign http requests bound for protected resources:

```
        oa = OAuth1Service('http://private.foo/oauth', 'private', 'key')
        requests.get('http://private.foo/private_resource', auth=oa.signer)
```

verify(*verifier*)
Sends a signed request to the OAuth server trading in your temporary credentials for *access credentials* these can be used to sign requests for the users protected resources.

Parameters **verifier** – Verification code passed from the OAuth server through the users OAuth workflow. Can be either passed in a redirect or the user could be instructed to copy it over.

class moxie.oauth.services.**OAuthCredential**(*key*)
Descriptor for caching our OAuth credentials in a user session. Only if one is available. Caches on the *OAuth1Service* object in an attribute named by OAuthCredential.credential_store

2.10 Provider

As opposed to Importer providers do not create persistent data. Instead data from providers should be safely cached and re-requested at an appropriate interval.

2.11 Representation

Representation of a Domain object, it is often a simplified view that is sent to clients in a specific format.

2.12 Search

class moxie.core.search.**SearchResponse**(*raw_response*, *query*, *size*, *results=None*, *query_suggestion=None*, *facets=None*)

as_dict
Raw response as a dict :rtype dict

as_json
Raw response as JSON :rtype string of JSON

facets
Facets of the query :rtype list of facets

query
String of the query (FTS) :rtype string

query_suggestion
Suggestion of a new query :return query suggestion :rtype string

results
Response documents :return list of dict or None if no results :rtype list of dict

size
Size of the search result

exception moxie.core.search.**SearchServerException**(*message=None*, *status_code=None*, *payload=None*, *headers=None*)

message = 'Search service not available'

class moxie.core.search.**SearchService**(*backend_uri*)

Represents the abstraction between our Search implementation (by default, Apache Solr) and the public API. For configuration details, see the *Service* documentation.

All Search requests should be made through this service.

static _get_backend(*backend_uri*)

Parse the URI and imports the appropriate Search implementation The backend_uri schema is as follows: implementation+transport://domain/path/collection where:

implementation is the name of a supported scheme in moxie.core.search.SEARCH_SCHEMES.

collection name used by the backend to identify your index.

Parameters **backend_uri** – URI Representing your search implementation e.g. solr+http://example.com/solr/collection

Returns Searcher implementation.

commit()

get_by_ids(*ids*)

healthcheck()

index(*document*, ***kwargs*)

search(*query*, *fq=None*, *start=0*, *count=10*)

Generic search query :param query: dict of k/v corresponding to parameters to search :return *moxie.core.search.SearchResponse*

search_for_ids(*id_key*, *identifiers*)

suggest(*query*, *fq=None*, *start=0*, *count=10*)

2.13 Service

class moxie.core.service.**ProviderService**(*providers={}*)

Used where a *Service* deals with many external providers. Example usage can be found in the TransportService

get_provider(*doc*, **args*, ***kwargs*)

Returns a Provider which can handle your doc.

If no (single) approrpriate provider can be found for your document we raise a ProviderException. Two subclasses are currently raised:

- NoSuitableProviderFound if we can't find *any* provider.
- MultipleProvidersFound if we find more than one provider.

class moxie.core.service.**Service**

Services are HTTP (transport layer) agnostic instead operating at the Application Layer. Services encapsulate all operations made on the data. Views should never directly access data sources without going through a Service.

Configuration of services can be done for each Blueprint. Within the Application context they will be cached, this means the following code accesses the same Service object.:

```
    with app.app_context():
        service_one = MyService.from_context()
        service_two = MyService.from_context()
        assert(service_one is service_two)
```

classmethod **from_context**(*blueprint_name=''*)

Create a *Service* from the application and request context. args and kwargs for the *Service* are read from the `Flask.config`. Configuration should follow this pattern:

```
        SERVICES = {
            'my_blueprint': {
                'MyService': (args, kwargs),
                'MySecondService: ((1,2,3), {'foo': 'bar'},
                }
            }
```

Parameters **blueprint_name** – Override the blueprint name so it isn't read from the request context.

2.14 Tasks

Celery task definitions which can optionally run periodically. Can be for anything but generally run an Importer.

2.15 ServiceView

Provides set of resources which can be accessed by defined routes. Currently most of our views represent data as json over HTTP.

Handles CORS, content-negotiation and geo-location awareness. Calls Service to access data.

2.15.1 Caching headers

You can control the value of the HTTP headers *Expires* and *Cache-Control* by setting the property *expires* of your view to either a *timedelta* or a *datetime*.

```
from datetime import timedelta, datetime
from moxie.core.views import ServiceView

class NearRealTimeInformation(ServiceView):

    expires = timedelta(seconds=5)
    # expires = datetime.utcnow().replace(hour=23, minute=59)

    def handle_request(self):
        return {'real': 'time'}
```

2.15.2 Exceptions

You should raise exceptions in case of an error in your application.

Moxie provides *ServiceUnavailable*, *BadRequest* and *NotFound* in *moxie.core.exceptions*, to respectfully provide 503, 400 and 404 responses.

The generic *ApplicationException* is also available, *message* and *status_code* parameters can be passed to have a more personalised exception.

```
from moxie.core.exceptions import NotFound, ApplicationException

class DetailView(ServiceView):

    def handle_request(self, uuid):
        # pseudo logic: book = service.get(uuid)
        if not book:
            raise NotFound()    # HTTP 404

        # pseudo logic:
        # user = view.get_user()
        # authorized = service.view_book(user, book)
        if not authorized:
            raise ApplicationException(message="You are not authorized to see this book",
                                       status_code=401)
```

CHAPTER 3

Developer

3.1 Configuration

3.1.1 Logging

Sentry/Raven can be used with Moxie, the key *SENTRY_DSN* has to be set in the *flask* section of the configuration.

An optional key *SENTRY_LEVEL* can be used to define the level of logging (*WARNING* by default).

3.2 Metrics

Metrics are useful to understand the performances of your application.

3.2.1 Solr (search server) metrics

Display the average QTime (number of milliseconds to execute a search) of Solr from a log file:

```
cat /var/log/jetty/solr-0.log | grep QTime | awk '{print $NF}' | awk -F\= '{ s += $2} END {print s/NR
```

Display request handlers used per core:

```
cat /var/log/jetty/solr-0.log | grep path | awk '{print $2 $4}' | sort | uniq -c
```

Display search queries (parameter "q") which are not search for "identifiers", and not browsing (q=":"):

```
cat /var/log/jetty/solr-0.log | grep "places" | grep "path=/select" | grep "q=" | awk '{print substr
```

CHAPTER 4

List of apps available

4.1 List of apps

Listing of apps currently used by Moxie.

4.1.1 Contact search

Search for contacts, see documentation.

4.1.2 Graduate courses search

Search for courses, browse by subject...

See documentation.

4.1.3 Library search

Search for books, get real-time availability information...

See documentation.

4.1.4 Dates

Display current (Oxford) date.

See documentation.

4.1.5 Places search

Search for places, browse by categories...

See Places endpoint

4.1.6 Transport real-time information

Get real-time information about transports

See Transport endpoint

4.1.7 Weather

See documentation.

CHAPTER 5

Data

5.1 OxPoints importer

The OxPoints importer is using the RDF/XML representation of the full dataset of OxPoints, and also requires the extension of OxPoints containing shapes of buildings as WKT (Well-Known Text).

The goal of Moxie is to provide a "simplified" view of OxPoints, easier to understand from an end-user point of view. The following transformations have been done:

1. A **Thing** has been merged with its **primary site** if they have the same name (e.g. Colleges)
2. Some types have been regrouped (see table below)

5.1.1 Types mapping

The following table explains the transformation on types between OxPoints and Moxie.

OxPoints type	Mapped type
Building	Building
Carpark	**University** carpark
College	College
Department	Department
Division	Division
Faculty	**Department**
Hall	Hall
Library	Library
Museum	Museum
OpenSpace	**Not imported**
Outside	**Not imported**
Place	
Room	Room
Site	Site
Space	Space
StudentGroup	**Not imported**
SubLibrary	SubLibrary
Unit	**Department**
University	University

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

/places

/transport

m

Symbols

A

C

F

G

H

I

K

M

O

P

Q

R

S

V