Neo4j Rest API

https://neo4j.com/docs/rest-docs/current/

What is REST?

- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000.
- Key abstraction of information in REST is a resource
- A resource identifier is used to identify the particular resource involved in an interaction between components
- Typically used in conjunction with HTTP protocol
  - GET — retrieve a specific resource (by id) or a collection of resources
  - POST — create a new resource
  - PUT — update a specific resource (by id)
  - DELETE — remove a specific resource by id

curl

- curl is a command-line utility that lets you execute HTTP requests with different parameters and methods.
- https://curl.se/download.html

- **curl http://localhost:7474/db/data/**

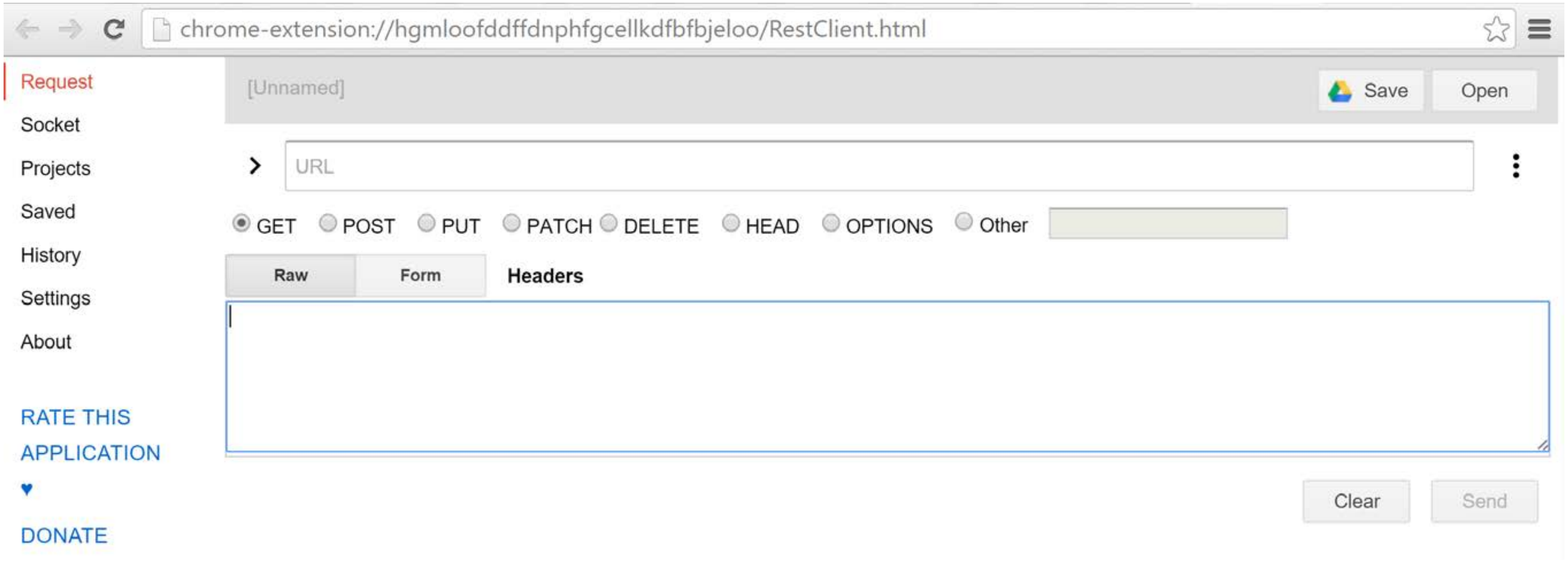
Using curl with Windows

If you’re using Windows, note the following formatting requirements when using curl:

- Use double quotes in the Windows command line. (Windows doesn’t support single quotes.)
- Don’t use backslashes (\) to separate lines. (This is for readability only and doesn’t affect the call on Macs.)
- By adding -k in the curl command, you can bypass curl’s security certificate, which may or may not be necessary.

Alternative

Google Chrome Advanced REST Client
After installing and launching the Google Chrome Advanced REST Client application, your browser should appear as follows:



chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo/RestClient.html

● GET

Accessing Neo4j Data using REST API

- Service root is the starting point to discover the REST API
  - GET http://localhost:7474/db/data/

    Accept: application/json

Creating Nodes Using REST

- Creating a node requires a POST to the /db/data/node path
- with JSON data. As a matter of convention, it pays to give each node a name
- property. This makes viewing any node’s information easy: just call name.
- **$ curl -i -X POST http://localhost:7474/db/data/node **
- **-H** *"Content-Type: application/json"* ****
- **-d '{**
- "name": "P.G. Wodehouse"
- "genre": "British Humour"
- }’

- **$ curl http://localhost:7474/db/data/node/1**
- **$ curl http://localhost:7474/db/data/node/1/properties/genre**

- **Add another node with these properties** ["name" : "Jeeves Takes Charge", "style" : "short story“]

Creating Relationships Using REST

- P. G. Wodehouse wrote the short story “Jeeves Takes Charge,” we
- can make a relationship between them:

- **curl -i -XPOST http://localhost:7474/db/data/node/9/relationships \**
- **-H** *"Content-Type: application/json"* \
- **-d '{**
- "to": "http://localhost:7474/db/data/node/1",
- "type": "WROTE",
- "data":{"published":"November 28, 1916"}
- }

- **$ curl http://localhost:7474/db/data/node/2**

Finding a Path

- You can find the path between two nodes by posting the request data to the starting node’s /paths URL. The POST request data must be a JSON string denoting the node you want the path to, the type of relationships you want to follow, and the path-finding algorithm to use.

```
curl -X POST http://localhost:7474/db/data/node/2/paths \
-H "Content-Type: application/json" \
-d ‘{
    "to": "http://localhost:7474/db/data/node/2",
    "relationships": {"type": "WROTE"}, "algorithm":
    "shortestPath",
    "max_depth": 10
}'
```

Indexing

- Neo4j indexes have a different path because the indexing service is a separate service.
- To create a key-value or hash style index:

```
$ curl -X POST http://localhost:7474/db/data/index/node/authors \
-H "Content-Type: application/json" \
-d '{
"uri": "http://localhost:7474/db/data/node/9",
"key": "name",
"value": "P.G.+Wodehouse"
}’
```

```
curl http://localhost:7474/db/data/index/node/authors/name/P.G.+Wodehouse
```

Full-text indexing

- Neo4j incorporates Lucene to build an inverted index over the entire dataset.

```
curl -X POST http://localhost:7474/db/data/index/node \
-H "Content-Type: application/json" \
-d '{
"name": "fulltext",
"config": {"type": "fulltext", "provider": "lucene"}
}’
```

- Add Wodehouse to the full-text index, you get this:

```
$ curl -X POST http://localhost:7474/db/data/index/node/fulltext \
-H "Content-Type: application/json" \
-d '{
"uri": "http://localhost:7474/db/data/node/9",
"key": "name",
"value" : "P.G.+Wodehouse"
}'
```

- Then you can query using the Lucene syntax on the index URL

```
$ curl http://localhost:7474/db/data/index/node/fulltext?query=name:P*
```

REST and Cypher

- Neo4j REST interface has a Cypher plugin

```
$ curl -X POST \
http://localhost:7474/db/data/cypher \
-H "Content-Type: application/json" \
-d '{
"query": "MATCH ()-[r]-() RETURN r;"
}'
{
"columns" : [ "n.name" ],
"data" : [ [ "Prancing Wolf" ], [ "P.G. Wodehouse" ] ]
}
```

Using Transactional Cypher HTTP Endpoint

- Allows you to execute a series of Cypher statements within the scope of a transaction
- The transaction may be kept open across multiple HTTP requests, until the client chooses to commit or roll back
- Each HTTP request can include a list of statements
- https://neo4j.com/docs/http-api/3.5/actions/

Using Drivers to Access Neo4j

- https://neo4j.com/developer/language-guides/
- Binary Bolt protocol (starting with Neo4j 3.0)
- Binary protocol is enabled in Neo4j by default and can be used in any language driver that supports it
- Drivers implement all low level connection and communication tasks

```
import org.neo4j.driver.v1.*;

public class Neo4j
{
 public static void javaDriverDemo() {
   Driver driver = GraphDatabase.driver("bolt://ganxis.nest.rpi.edu", "neo4j", "neo4j"));
   Session session = driver.session();

   StatementResult result = session.run("MATCH (a)-[]-(b)-[]-(c)-[]-(a) WHERE a.id < b.id AND b.id < c.id
RETURN DISTINCT a,b,c");
   int counter = 0;
   while (result.hasNext())
   {
      counter++;
      Record record = result.next();
      System.out.println(record.get("a").get("id") + " \t" + record.get("b").get("id") + " \t" +
record.get("c").get("id"));
   }
   System.out.println("Count: " + counter);
   session.close();
   driver.close();
 }
 public static void main(String [] args)
 {
   javaDriverDemo();
 }
}
```

Using Core Java API

- Native Java API performs database operations directly with Neo4j core

```
import java.io.*;
import java.util.*;
import org.neo4j.graphdb.*

public class Neo4j
{
  public enum NodeLabels implements Label { NODE; }
  public enum EdgeLabels implements RelationshipType{ CONNECTED; }
  public static void javaNativeDemo(int nodes, double p) {
    Node node1, node2; Random randomgen = new Random();
    GraphDatabaseFactory dbFactory = new GraphDatabaseFactory();
    GraphDatabaseService db = dbFactory.newEmbeddedDatabase(new File("TestNeo4jDB"));
    try (Transaction tx = db.beginTx()) {
      for (int i = 1; i <= nodes; i++) {
        Node node = db.createNode(NodeLabels.NODE);
        node.setProperty("id", i);
      }
      for (int i = 1; i <= nodes; i++)
        for (int j = i + 1; j <= nodes; j++) {
          if (randomgen.nextDouble() < p) {
            node1 = db.findNode(NodeLabels.NODE, "id", i);
            node2 = db.findNode(NodeLabels.NODE, "id", j);
            Relationship relationship =
node1.createRelationshipTo(node2,EdgeLabels.CONNECTED);
            relationship = node2.createRelationshipTo(node1,EdgeLabels.CONNECTED);
          }
        }
      tx.success();
    }
    db.shutdown();
  }
  public static void main(String [] args) {
    javaNativeDemo(100, 0.2); }
}
```

Phyton

- https://neo4j.com/developer/python/

- Py2neo is a client library and comprehensive toolkit for working with Neo4j from within Python applications.

```
pip install py2neo
from py2neo import Graph
graph = Graph("bolt://localhost:7687", auth=("neo4j",
"asdfgh123"))
query = "MATCH (n) return n"
graph.run(query).data()
```

For today…

- Download and install redis http://redis.io
- From the command line start the server by calling:
- **$ redis-server**
- It won’t run in the background by default, but you can append &, or just open another terminal.
- Next, run the command line tool, which should connect to the default port 6379 automatically.
- **$ redis-cli**
- After you connect, ping the server (it should reply PONG):

```
redis 127.0.0.1:6379> PING
```

- Submit to ICON a screenshot of the redis-cli running