A REST Server in Python

To illustrate REST, let us work through a Payroll sample. For simplicity this example is designed so that resources are all database entities (rows in tables), but as we shall see, this does not mean that client requests follow the database schema!

An Employee management system

The tables we will use are basically as follows (see Mark II of these tables in the next section):

EMPLOYEE: (ID INTEGER PRIMARY KEY, NAME VARCHAR(20), NOTES VARCHAR(1000))

POST: (EMPID REFERENCES EMPLOYEE, FROM DATE, GRADE VARCHAR(4), MANAGER REFERENCES EMPLOYEE)

HOLIDAY: (EMPID REFERENCES EMPLOYEE, FROM DATE, TO DATE, AGREED DATE)

And let's give them some initial data:

EMPLOYEE

1562	John Black	Sales
1567	Mary White	Finance
1569	Paul Green	HR

POST

1562	1/2/2012	A1	1569
1562	1/4/2012	A2	
1567	1/2/2012	B1	1569
1569	1/2/2012	A2	

HOLIDAY

1567	2/3/2012	7/3/2012	5/2/2012
1569	4/4/2012	18/4/2012	3/4/2012

To keep things simple, let's write a very simple Python program to set up the database we want. We will do this in a moment: first we modify this design to take account of transactions.

Transactions and data currency

Before launching in to this tutorial, let us consider the transaction issues in this design, which is of course the back-office end of one or more web applications (for noting a new employee, a new post/resignation/new grade/new manager for an employee, and for booking and approving holidays). The only access to this data will be through our database server, and we will use single threading, so actual concurrency control on the database is not an issue. Obviously the web application will want to deal with authentication and authorisation issues, so that employees can book their holidays but approve them or change their grade, managers can approve holidays but not change them etc; and we can support that using the techniques described later in this tutorial.

However, because HTTP is stateless, it is quite likely that web clients have stale information on their screens, because other clients may have updated something. For 6 of the 7 use cases mentioned in the above paragraph we need a way of checking that the information shown is current: if not, the client should be asked whether they want to refresh it before they proceed with the changes they want to make.

If all the information on the client's screen came from a single record in the database, the check could use row versioning. Some DBMS including SQLServer, Postgres and PyrrhoDB do this for you, and more generally triggers can be used to maintain row version information. But doing this misses two important points:

(a) This approach will not work well if the data comes from several tables.
(b) Our database is embedded: no other programs can possibly make changes, so we don't need anything as complicated as triggers

So (although PyrrhoDB has a CHECK function that might work well for us) no matter what database we are using, we need to consider a custom mechanism for checking data currency. In addition to placing some timestamp-like fields in the database, we can add some extra version information to messages between client and server. As we will see this will mean that the data in a REST PUT will not be same as the entries in the row.

Let's design all of that before building the API.

A Versioning table

RVV: (SEQ INTEGER)

RVV

SEQ
0

We will implement a simple method in our REST server to increment this version number in the database every time we retrieve it. (This is much better than keeping a static/global counter in the REST server itself.)

And we modify our base tables to contain a RVV field:

EMPLOYEE: (ID INTEGER PRIMARY KEY, NAME VARCHAR(20), NOTES VARCHAR(1000), RVV INTEGER)

POST: (EMPID REFERENCES EMPLOYEE, FROM DATE, GRADE VARCHAR(4), MANAGER REFERENCES EMPLOYEE, RVV INTEGER)

HOLIDAY: (EMPID REFERENCES EMPLOYEE, FROM DATE, TO DATE, AGREED DATE, RVV INTEGER)

The REST API

To a first approximation our REST API will be simply a set of CRUD operations. There will be GETs of course, and also the 7 use cases mentioned above.

New employee: The server will need to assign the new employee id.

New post: This is a POST to the Posts table, containing the information about the new post, but will include the rvv of the Employee information as a check.

Resignation: Although this is basically a PUT to the Posts table, the data sent to the server should include the id and rvv of the Employee as well as the rvv of the most recent Post. If all of that checks out, the resignation can be processed.

New grade/new manager: Here the data sent to the server should include the Employee's id, new grade/manager and the rvv of the Employee and the current Post.

New holiday: This is a POST to the Holiday table, containing the requested dates and will include the Employee rvv.

Approve holiday: This is a PUT to the Holiday table, identifying the holiday but will also include the rvvs of Employee and Post information, as only the current Manager can approve the holiday, and will want to check the entitlement based on grade etc.

Getting Python

If our example had issues of scalability I would be configuring a compiler for Python to machine code, but for this example a single server thread will be fine, and there is no need to worry about multithreading or compilation.

If you haven't already done so, download Python from Python.org, which is available for a great many platforms. Python 3.4 was the latest released version at the time of writing, and has been used for this tutorial*. Comments show the alternative coding for Python 2.7.*

I use Visual Studio, whose Python tools I find helpful, but anything will do: many authors recommend Notepad++. In favour of Visual Studio, the debugger is great: if you can, use a system with a Python debugger that is as good. So the instructions in this tutorial will just use phrases like "create a new file", "add a method to" etc, rather than being Visual Studio-specific. Be careful with indentation though: always use tabs rather than spaces at the start of lines.

The names of my starting files will be Payroll1.py and Payroll1Setup.py, but you can use any names you want.

Create the database we want

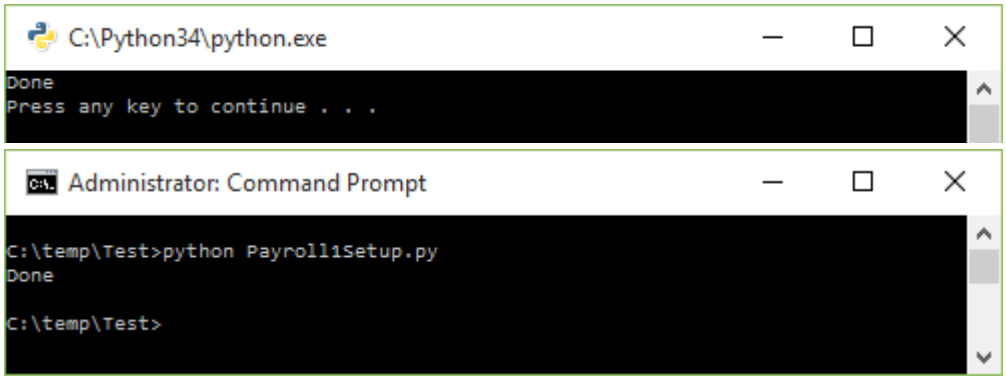
1. In Payroll1Setup.py, add the following text:

```
import sqlite3

conn = sqlite3.connect('Payroll.db')
conn.execute('create table employee(id integer primary key,'+
                     'name varchar(20),notes varchar(100),rvv integer)')
conn.execute('create table post(empid references employee,efrom date, '+
                    'grade varchar(4),manager references employee,rvv integer)')
conn.execute('create table holiday(empid references employee,hfrom date,'+
                    ' hto date, agreed date,rvv integer)')
conn.execute('create table rvv(seq integer)')
conn.execute('create table user(uname varchar(20) primary key,password varchar(20))')
conn.execute("insert into employee values(1562,'John Black','Sales',1)")
conn.execute("insert into employee values(1567,'Mary White','Finance',2)")
conn.execute("insert into employee values(1569,'Paul Green','HR',3)")
conn.execute("insert into post values(1562,'2012-02-01','A1',1569,4)")
conn.execute("insert into post values(1562,'2012-04-01','A2',null,5)")
conn.execute("insert into post values(1567,'2012-02-01','B1',1569,6)")
conn.execute("insert into post values(1569,'2012-02-01','A2',null,7)")
conn.execute("insert into holiday values(1567,'2012-03-02','2012-03-07'," +
                    "'2012-02-05',8)")
conn.execute("insert into holiday values(1569,'2012-04-04','2012-04-18'," +
                    "null,9)")
conn.execute("insert into rvv values(10)")
conn.execute("insert into user values('aUser','whosOk')")
conn.commit();
print('Done')
```

If you read this closely you will see a user table: we will come to authentication at step 33.

2. Run this program. It will create Payroll.db, so if you need to run it again for some reason you had better delete the file first. It should output the single line DoneIf anything else happens, check everything and try again. . (The first screenshot below is from Visual Studio, the second from Windows and your environment may be different.)



3. We are finished with Payroll1Setup.py. We can rerun it later to regenerate the Payroll.db file, or we can just save this file somewhere.

Create a web server

4. Create file Payroll1.py if necessary, and replace any contents with:

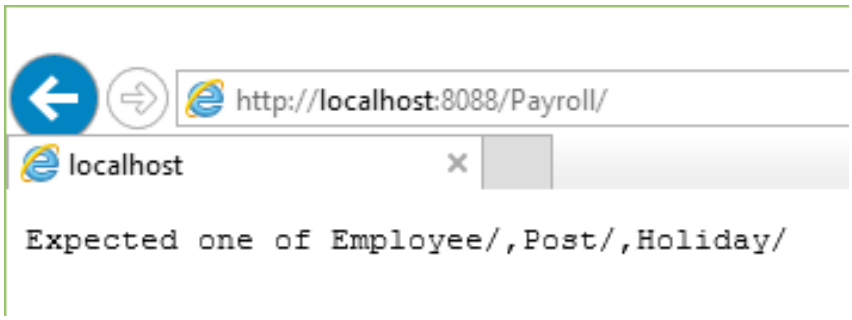
```
from http.server import * #3.4
# from BaseHTTPServer import * #2.7
import sys
# import codecs #2.7

class myHandler(BaseHTTPRequestHandler):
    def Send200(self,mess):
        self.send_response(200)
        self.send_header('Content-type','text/plain')
        self.end_headers()
        self.wfile.write(bytes(mess,'utf-8')) #3.4
  #      self.wfile.write(codecs.encode(mess,'utf-8')) #2.7
        return
    def SendError(self,status,mess):
        self.send_response(status)
        self.send_header('Content-type','text/plain')
        self.end_headers()
        self.wfile.write(bytes(mess,'utf-8')) #3.4
   #     self.wfile.write(codecs.encode(mess,'utf-8')) #2.7
        return
    def GetData(self):
   #     h = self.headers.getheader('Content-Length') #2.7
   #     if h == None #2.7
        if not self.headers.__contains__('Content-Length'): #3.4
            return None
        h = self.headers.__getitem__('Content-Length') #3.4
        n = int(h)
        return str(self.rfile.read(n),'utf-8') #3.4
   #     return codecs.decode(self.rfile.read(n),'utf-8') #2.7
    def do_GET(self):
        try:
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Exception as e:
            self.SendError(403,sys.exc_info()[1]])
        return
    def do_POST(self):
        try:
            s = self.GetData()
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Exception as e:
            self.SendError(500,sys.exc_info()[1]
        return
```

```
    def log_request(code,size):
        return
try:
    server = HTTPServer(('',8088),myHandler)
    server.serve_forever()
except KeyboardInterrupt:
    print ('Exiting')
    server.socket.close()
```

The log_request method is an override: without this you would get notes about http requests to the application window. We return to this aspect later.

5. Check this works. Run the program as administrator: you initially get no output. In your browser try http://localhost:8088/Payroll



Exit the program with ^C or by closing the window.

Add the database we want

6. Copy Payroll.db to the folder containing Payroll1.py. At the top of Payroll1.py add
   `import sqlite3`
7. Just before the server= near the end, add
   `conn = sqlite3.connect('Payroll.db')`

Add the database classes we want

8. Create a new file EMPLOYEE.py, and add the following text:

```
class EMPLOYEE:
    """Python class representing a row of the EMPLOYEE table"""
    def __init__(self):
        self.id = 0
        self.name = ''
        self.notes = ''
        self.rvv = 0
        return
```

9. Create a new file POST.py and add the following text:

```
from datetime import *

class POST:
    """Python class representing a row of the POST table"""
    def __init__(self):
        self.empid = 0
        self.efrom = datetime(1900,1,1)
        self.grade = ''
        self.manager = 0
        self.rvv = 0
        return
```

10. Create a similar file for HOLIDAY.py:

```
from datetime import *

class HOLIDAY(object):
    """Python class representing a row of the HOLIDAY table"""
    def __init__(self):
        self.empid = 0
        self.hfrom = datetime(1900,1,1)
        self.hto = datetime(1900,1,1)
        self.agreed = datetime(1900,1,1)
        self.rvv = 0
        return
```

First steps with Reflection/Introspection

11. Let's create some general purpose classes for converting database data to and from JavaScript objects and strings. Create a file Json.py with the following text:

```
from types import *

"""JSON formatters etc"""
def Stringify(ob):
    if ob == None:
        return '<null>'
    sb = '{'
    cm = ''
    for f in dir(ob):
        if f[0]!='_':
            v = getattr(ob,f)
            sb += cm + '"' +f + '": '
            cm = ', '
            if isinstance(v,str):
                sb += '"'
            sb += str(v)
            if isinstance(v,str):
                sb += '"'
    return sb + '}'

def GetOne(x,ob,r):
    if r==None:
        return ''
    d = dir(r)
    for i in range(len(r)):
        f = x.description[i][0]
        setattr(ob,f,r[i])
    return Stringify(ob)

def GetAll(ob,conn):
    sb = '['
    cm = ''
    tp = ob.__class__.__name__
    c = conn.cursor()
    x = c.execute('select * from '+tp)
    for r in x:
        ob = ob.__class__()
        GetOne(x,ob,r)
        sb += cm + Stringify(ob)
        cm = ','
    c.close()
    return sb+']'
```

```
def GetAllWith(ob,conn,cond):
    tp = ob.__class__.__name__
    sb = '['
    cm = ''
    c = conn.cursor()
    x = c.execute('select * from '+tp+' where '+cond)
    for r in x:
        ob = ob.__class__()
        GetOne(x,ob,r)
        sb += cm + Stringify(ob)
        cm = ','
    c.close()
    return sb + ']'
```

GetEmployee

12. At the top of EMPLOYEE.py add

```
from types import *
import Json
```

13. Add a method to EMPLOYEE.py:

```
    def _Find(id,conn):
        c = conn.cursor()
        x = c.execute('select * from EMPLOYEE where id='+str(id))
        e = EMPLOYEE()
        Json.GetOne(x,e,c.fetchone())
        c.close()
        return e
```

The initial underscore is to make it easy to distinguish such methods from the names of fields. See the code in Stringify (step 11 above).

14. At the top of Payroll1.py add:

```
from EMPLOYEE import *
from POST import *
from HOLIDAY import *
import Json
```

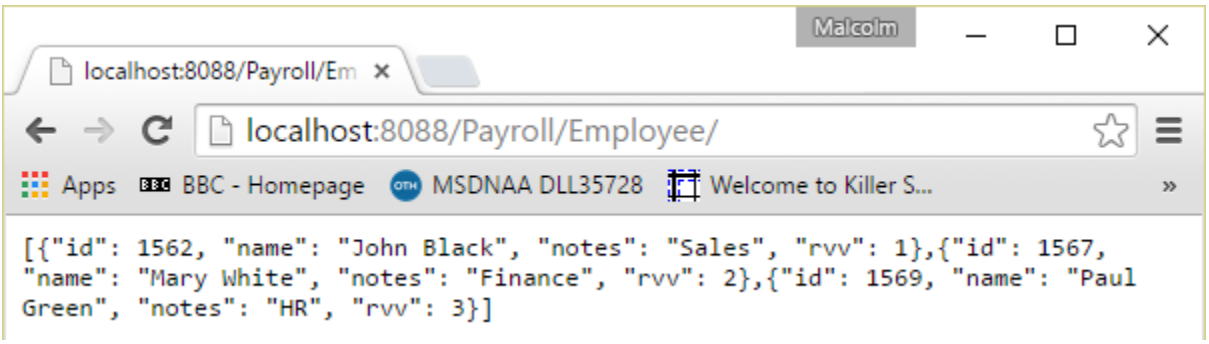
15. Add a method to Payroll1.py:

```
    def GetEmployee(self,p):
        if len(p)==4:
            return Json.GetAll(EMPLOYEE(),conn)
        id = int(p[3])
        return Json.Stringify(EMPLOYEE._Find(id,conn))
```

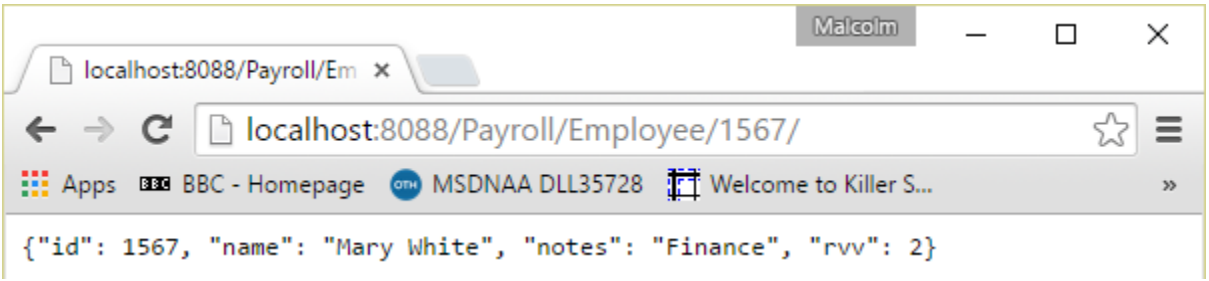
16. And change do_GET in Payroll1.py:

```
    def do_GET(self):
        try:
            p = self.path.split('/')
            if len(p)>=3:
                if p[2]=='Employee':
                    self.Send200(self.GetEmployee(p))
                    return
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Exception as e:
            self.SendError(403, sys.exc_info()[1])
        return
```

17. Now run the program (as Administrator) and check the following URLs with your browser:
    http://localhost:8088/Payroll/Employee/



    http://localhost:8088/Payroll/Employee/1567/



GetPost and GetHoliday

18. Stop the program and add similar methods for GETting Post and Holidays. Aim to try out

    http://localhost:8088/Payroll/Post/

    http://localhost:8088/Payroll/Post/1562/

    http://localhost:8088/Payroll/Holiday/

    http://localhost:8088/Payroll/Holiday/1569/

    Note however that for our sample data Post/1562/ should return two Json objects, and an employee can have several holidays booked, so use Json.GetAllWith instead of having a _Find method.

19. It might be good to allow another path element to the URL to pick up the nth such. So for POST and HOLIDAY we should have _FindNth methods like

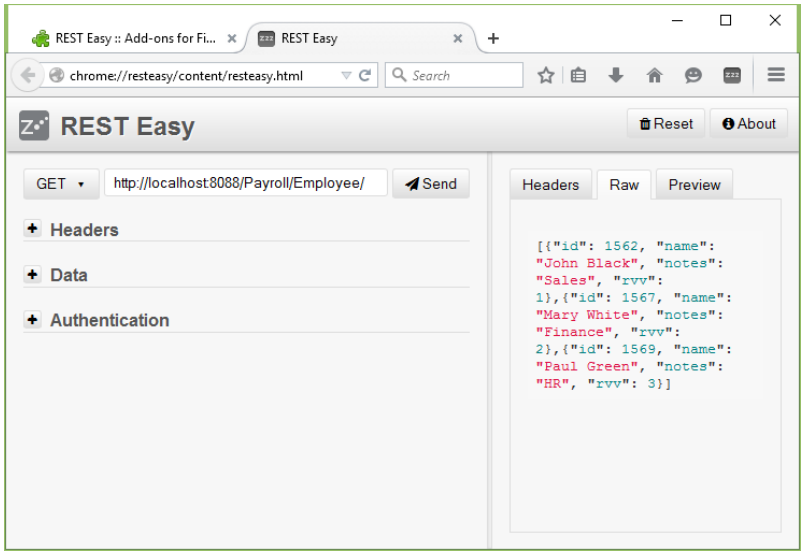
    ```
    def _FindNth(id,n,conn):
    ```

    Modify GetPost and GetHoliday to call use this method when appropriate. Check for http://localhost:8088/Payroll/Post/1562/2/

Make sure all this works before continuing.

Introducing a Rest client application

20. We will use the REST Easy Add-on for Firefox. To get into using it, run the server and try GET on the last few URLS (change to the Raw tab):



Getting started with POST

21. Let's define methods in Json.py to initialise an object from JSON format, and to create an INSERT statement for any type matching a database table. At the top add

```
from datetime import *
```

and then add methods

```
def Fill(ob,s):
    off = 1
    while off < len(s)-1:
        ch = s[off]
        if s[off]=='}':
            return
        c = s.index(':',off)
        if s[off]=='"':
            f = s[off+1:c-1]
        else:
            f = s[off:c]
        off = c+2
        c = s.find(',',off)
        b = s.find('}',off)
        if c<0 or (b>0 and b<c):
            c = b
        v = getattr(ob,f)
        if isinstance(v,str):
            setattr(ob,f,s[off+1:c-1])
        elif isinstance(v,int):
            setattr(ob,f,int(s[off:c]))
        elif isinstance(v,float):
            setattr(ob,f,float(s[off:c]))
        elif isinstance(v,datetime):
            d = datetime.strptime(s[off+1:c-1],'%Y-%m-%d')
            setattr(ob,f,d)
        off = c;
        if s[off]=='}':
            return
        off+= 2
    return
def PostSQL(ob):
    tp = ob.__class__.__name__
    sb = 'insert into '+tp+'('
```

```
    sc = ') values ('
    cm = ''
    for f in dir(ob):
        if f[0]!='_':
            v = getattr(ob,f)
            if v==None:
                continue
            sb += cm+f
            sc += cm
            cm = ','
            if isinstance(v,datetime):
                v = v.strftime('%Y-%m-%d')
            if isinstance(v,str):
                sc+="'"
            sc += str(v)
            if isinstance(v,str):
                sc+="'"
    return sb+sc+')'
```

22. We need a method for updating the RVV/SEQ number. In Payroll1.py add the following:

```
    def GetRvv(self):
        c = conn.cursor()
        c.execute('update rvv set seq=seq+1')
        c.close()
        c = conn.cursor()
        rn = int(c.execute('select seq from rvv').fetchone()[0])
        c.close()
        return rn
```

Implement POST

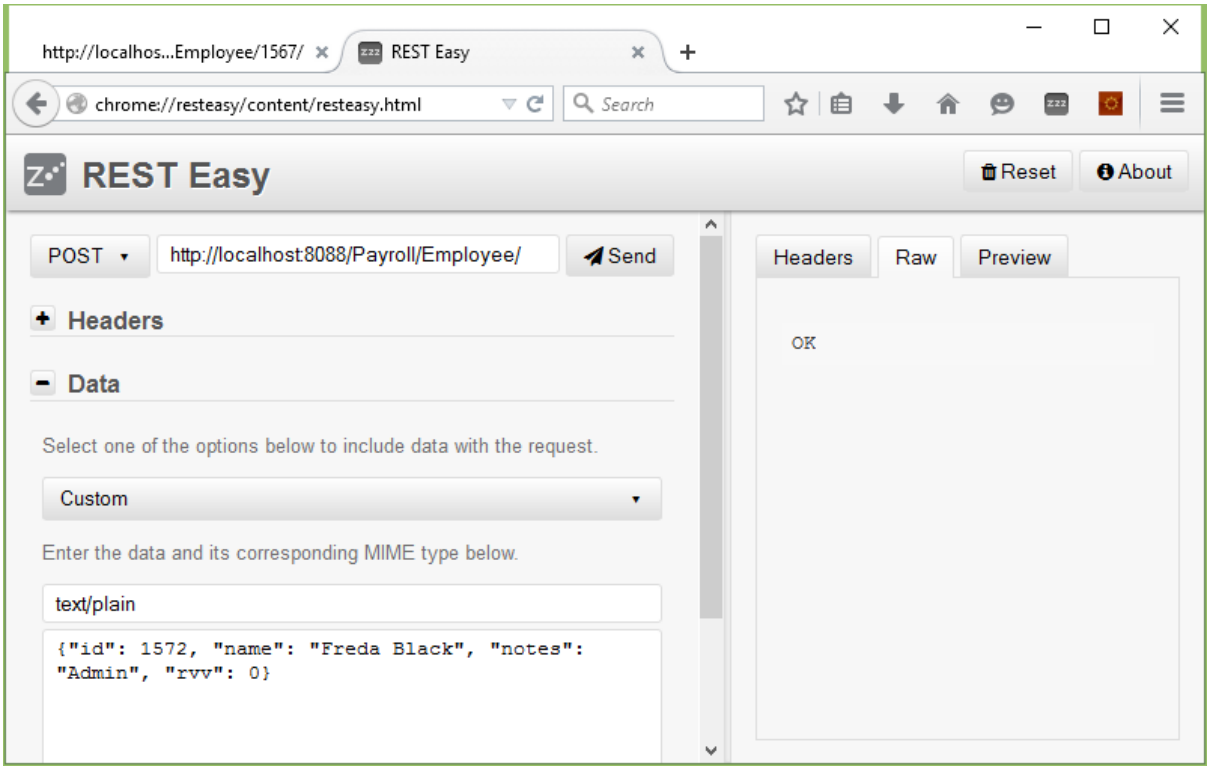
23. Add the following method to Payroll1.py:

```
    def PostEmployee(self,s):
        e = EMPLOYEE()
        Json.Fill(e,s)
        e.rvv = self.GetRvv()
        conn.execute(Json.PostSQL(e))
        conn.commit()
        return 'OK'
```

24. Update the do_POST method in Payroll1.py:

```
    def do_POST(self):
        try:
            s = self.GetData()
            p = self.path.split('/')
            if len(p)>=3:
                if p[2]=='Employee':
                    self.Send200(self.PostEmployee(s))
                    return
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Exception as e:
            self.SendError(500, sys.exc_info()[1])
        return
```

25. Try out the new machinery using the REST client. The coding in Fill is very naive and does not tolerate unexpected white space. So retrieve an employee, change the fields and then Post the new employee. You do this by expanding the Data section, selecting Custom from the drop-down, giving the MIME type as text/plain, and pasting/modifying the textbox. Try this out:



Subclassing POST and HOLIDAY

26. For POST for Post and Holidays, as discussed earlier, we also need to check other information at the client is still current. For implementing these operations, we need to define subclasses of POST and HOLIDAY to contain the extra rvv fields that the client will send, although these will not end up in the database.

    At the end of POST.py, add

```
class POST1(POST):
    def __init__(self):
        self.emp_rvv = 0
        return super(POST1, self).__init__()
```

27. Similarly, at the end of HOLIDAY, add

```
class HOLIDAY1(HOLIDAY):
    def __init__(self):
        self.emp_rvv = 0
        self.post_rvv = 0
        return super(HOLIDAY1, self).__init__()
```

Implementing PostPost

As discussed at the start of this tutorial, the information for a new Post will be not quite what goes in the database. So we need to be careful about what fields are included in the reflection-based conversion from the HTTP Request and to the SQL. What comes from the client will not be a POST but a POST1. On the other hand we want to send PostSQL a genuine POST not a POST1.

28. At this stage we will also need a method for checking the rvv field of a database table. In Json.py add

```
def CheckRvv(ob,rvv,conn):
    tp = ob.__class__.__name__
    c = conn.cursor()
```

```
        r = c.execute('select count(*) from '+tp+' where rvv='+str(rvv)).fetchone()
        c.close()
        if int(r[0])!=1:
            raise Exception('Data has changed')
        return
```

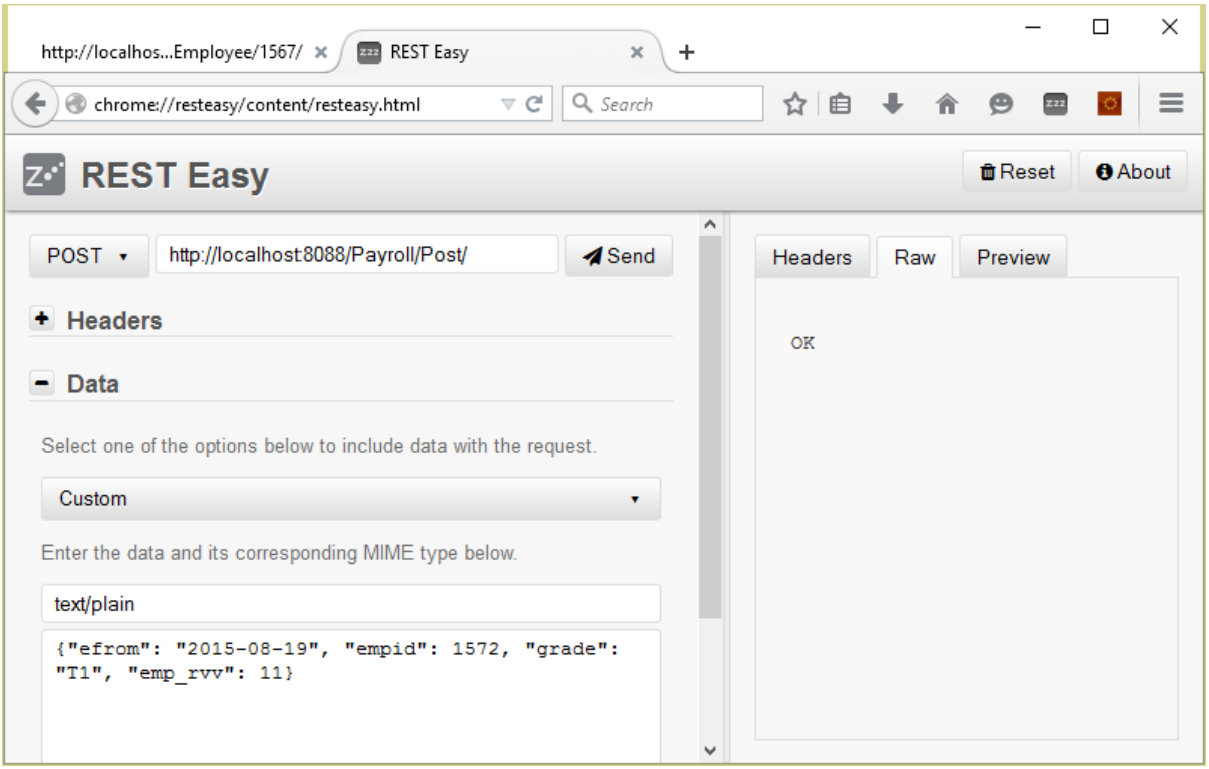
29. Implement POST POST. In Payroll1.py add

```
    def PostPost(self,s):
        p1 = POST1()
        Json.Fill(p1,s)
        Json.CheckRvv(EMPLOYEE(),p1.emp_rvv,conn)
        p = POST()
        p.empid = p1.empid
        p.efrom = p1.efrom
        p.grade = p1.grade
        p.manager = p1.manager
        p.rvv = self.GetRvv()
        conn.execute(Json.PostSQL(p))
        conn.commit()
        return 'OK'
```

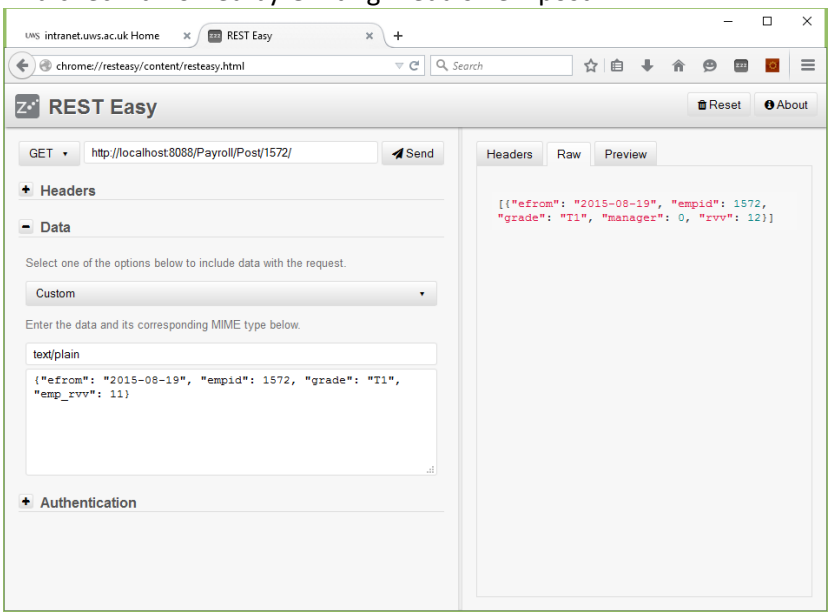
30. Change do_POST as follows:

```
    def do_POST(self):
        try:
            s = self.GetData()
            p = self.path.split('/')
            if len(p)>=3:
                if p[2]=='Employee':
                    self.Send200(self.PostEmployee(s))
                    return
                elif p[2]=='Post':
                    self.Send200(self.PostPost(s))
                    return
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Exception:
            self.SendError(500,sys.exc_info()[1])
        return
```

31. Now try this out. The web application would handle the emp_rvv field, but in our testing just now we have to fill it in manually. So use the REST to GET Employee/1572 for the rvv field (for me it was 11), and GET Post/1567 to ensure we have the right format for posting the new post. Try it:



And check it worked by GETting Freda's new post:



32. Now do the same for holidays. (You will need to provide post_rvv and emp_rvv.)

Authentication

33. In Payroll1.py, add at the top

```
import base64
```

and add an Authenticate method:

```
    def Authenticate(self):
        if not 'Authorization' in self.headers: #3.4
            raise Json.RestException(401,'No authorization header') #3.4
        h = self.headers['Authorization'] #3.4
#       h = self.headers.getheader('Authorization') #2.7
```

```
#     if h==None: #2.7
#         raise Json.RestException(401,'No authorization header') #2.7
        d = str(base64.b64decode(h[6:len(h)]),'utf-8') #3.4
#     d = codecs.decode(base64.b64decode(h[6:len(h)]),'utf-8') #2.7
        s = d.split(':')
        c = conn.cursor()
        r = c.execute("select count(*) from user where uname='"+s[0]+
"' and password='"+s[1]+"'").fetchone()
        if int(r[0])!=1:
            raise Json.RestException(401, 'Not authenticated')
        return
```

34. In Json.py we need to declare RestException. At the bottom add

```
class RestException(Exception):
    def __init__(self,err,mess):
        self.error = err
           return super(RestException,self).__init__(mess)
```

35. Now call this Authenticate at the start of do_GET and do_Post, and handle the exception:

```
    def do_GET(self):
        try:
            self.Authenticate()
            p = self.path.split('/')
            if len(p)>=3:
                if p[2]=='Employee':
                    self.Send200(self.GetEmployee(p))
                    return
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Json.RestException as e:
            self.SendError(e.error,e.args[0])
        except Exception as e:
            self.SendError(403,sys.exc_info()[1])
        return
```

36. We also need to change SendError to get the client to pop up the authorization dialog. Change it as follows:

```
    def SendError(self,status,mess):
        self.send_response(status)
        if status==401:
            self.send_header('WWW-Authenticate','Basic realm="Payroll"')
        self.send_header('Content-type','text/plain')
        self.end_headers()
        self.wfile.write(bytes(mess,'utf-8')) #3.4
#     self.wfile.write(codecs.encode(mess,'utf-8')) #2.7
        return
```

37. Try this out (the USER table has credentials for aUser with password whosOk). There is an opportunity here for authorisation in addition to authentication: the database is just for this application, so extra fields in the user table could identify the employee ID and/or be more specific about what changes a user is allowed to make.

PUT methods

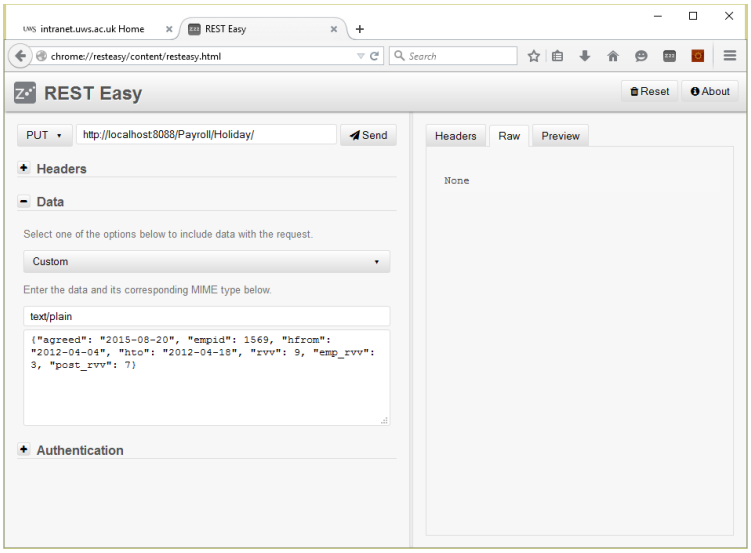
38. Python's http.server package allows a wide range of HTTP verbs. Although the manual says nothing about do_PUT, it will work fine to add a do_PUT method to Payroll1.py. We will work through one of the use cases (approving a holiday) from the introduction and leave the rest as an exercise.
    Let's begin by adding a method in Json.py to construct an SQL Update statement:

```
def PutSQL(ob,oldrvv):
    tp = ob.__class__.__name__
    sb = 'update '+tp+' set '
    cm = ''
    for f in dir(ob):
        if f[0]!='_':
            v = getattr(ob,f)
            if v==None:
                continue
            if isinstance(v,datetime):
                v = v.strftime('%Y-%m-%d')
            sb += cm+f+'='
            cm = ','
            if isinstance(v,str):
                sb+="'"
            sb += str(v)
            if isinstance(v,str):
                sb+="'"
    return sb+' where rvv='+str(oldrvv)
```

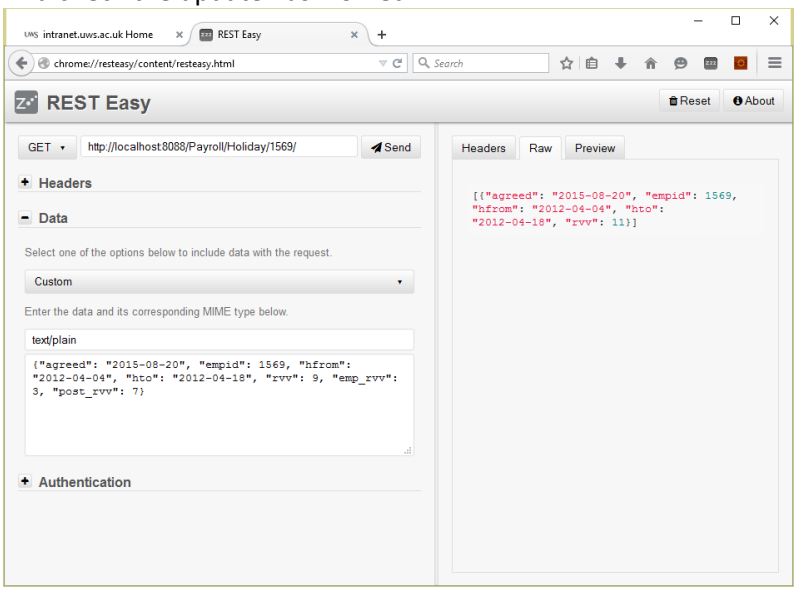
39. For simplicity the REST service will also allow changing the end date of a holiday but obviously the ideas in step 35 could restrict the employee to changing the length oftheir holiday (if it has not yet been approved) and restrict the manager to approving the holiday but not change the dates. Add the following methods to Payroll1.py:

```
    def PutHoliday(self,s):
        h1 = HOLIDAY1()
        Json.Fill(h1,s)
        Json.CheckRvv(EMPLOYEE(),h1.emp_rvv,conn)
        Json.CheckRvv(POST(),h1.post_rvv,conn)
        oldrvv = h1.rvv
        h = HOLIDAY()
        h.empid = h1.empid
        h.hfrom = h1.hfrom
        h.hto = h1.hto
        h.rvv = self.GetRvv()
        conn.execute(Json.PutSQL(h,oldrvv))
        conn.commit()
        return
    def do_PUT(self):
        try:
            self.Authenticate()
            s = self.GetData()
            p = self.path.split('/')
            if len(p)>=3:
                if p[2]=='Holiday':
                    self.Send200(self.PutHoliday(s))
                    return
            self.SendError(400,'Expected one of Employee/,Post/,Holiday/')
        except Json.RestException as e:
            self.SendError(e.error,e.args[0])
        except Exception as e:
            self.SendError(500 ,sys.exc_info()[1])
        return
```

40. Try this out by approving the first holiday for employee 1569 using PUT to url http://localhost:8088/Payroll/Holiday/1569/1/
    As before, first do a get for the Employee and latest Post to get the right values for emp_rvv and post_rvv which you will need to supply. Then do a get for the holday. This seems awkward because we haven't (yet?) written the web client front end – the appropriate rvvs would come for free in that context.



And check the update has worked:



A Transaction Log

41. Change the following methods in Payroll1.py. In Authenticate capture the user id:

```
    def Authenticate(self):
    #     h = self.headers.getheader('Authorization') #2.7
    #     if h==None: #2.7
    #         raise Json.RestException(401,'No authorization header') #2.7
        d = str(base64.b64decode(h[6:len(h)]),'utf-8') #3.4
    #     d = codecs.decode(base64.b64decode(h[6:len(h)]),'utf-8') #2.7
        s = d.split(':')
        self.user = s[0]
        c = conn.cursor()
        r = c.execute("select count(*) from user where uname='"+s[0]+
"' and password='"+s[1]+"'").fetchone()
        if int(r[0])!=1:
            raise Json.RestException(401, 'Not authenticated')
        return
```

42. In do_GET:

```
    def do_GET(self):
        try:
            self.Authenticate()
```

```
            self.data = ''
            p = self.path.split('/')
```

43. In do_POST:

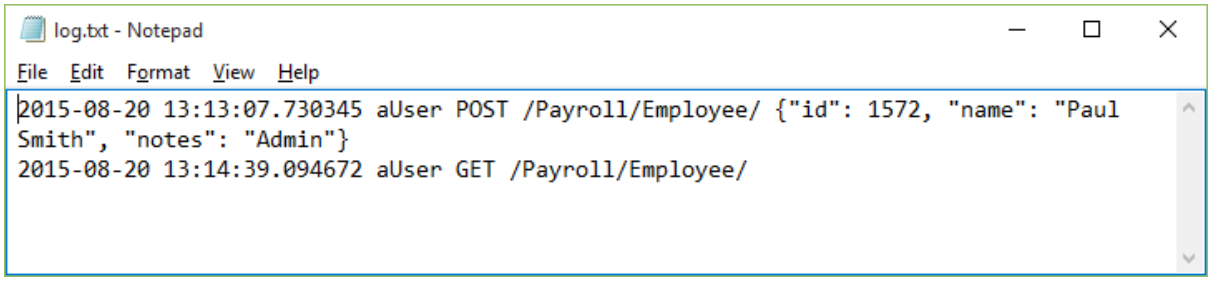
```
    def do_POST(self):
        try:
            self.Authenticate()
            s = self.GetData()
            self.data = s
            p = self.path.split('/')
```

44. In do_PUT:

```
    def do_PUT(self):
        try:
            self.Authenticate()
            s = self.GetData()
            self.data = s
            p = self.path.split('/')
```

45. Change the log_request method to read:

```
    def log_request(self,code):
        if code!=200:
            return
        log = open('log.txt','a')
        print(str(datetime.now()),self.user,self.command,self.path, #3.4
              self.data,file=log) #3.4
    #     s = str(datetime.now())+' '+self.user+' '+self.command+' '+self.path+' ' #2.7
    #         +self.data+'\n' #2.7
    #     log.write(s) #2.7
        log.close()
        return
```



If you find any issues with this tutorial, please write to me: Malcolm.Crowe@tawqt.com .