Technische Universität München
Fakultät für Informatik

Exercise Sheet 8: Distributed Systems and HTTP Requests with Python

Welcome to the eighth week of Advanced Topics of Software Engineering. For the theoretical exercises, we will analyze the trade-offs of stored procedure and sharding. Additionally, we will practice the MapReduce concept to execute SQL Queries.

In this week's practical exercise, we will learn how to consume a REST API with a Python script running inside a Raspberry Pi. We will also practice how to read the value of a photoresistor (brightness sensor) from a Raspberry Pi. Finally, we will work on a code piece to consume endpoints that are provided from the last week's Project application.

Theoretical Exercise

Exercise 1: Stored Procedure

In the lecture, you have learned about the advantages and drawbacks of stored procedures in SQL Databases. Let us further analyze the trade-offs of this technique.

1. Justify how stored procedures improve the following aspects of an application:
    - Security
    - Maintainability
    - Reusability
2. Why stored procedures reduce the flexibility of database queries?

Exercise 2: Scaling and Sharding

In this exercise, we will apply the scaling and sharding techniques from the lecture to expand an e-commerce system in the following scenario.

VeggieChain is a chain of vegetable stores operating in Germany across 16 states with about 500 stores selling over 6000 items that vary in price, categories. Each item is supplied by a particular supplier. VeggieChain has an e-commerce platform to sell its product online and is planning to extend its business in Bremen and Frankfurt (Hesse). Thus the software architects are considering the best strategy to scale the system.

60 percent of VeggieChain's profit come from Hamburg (15%), Berlin (20%), and Bavaria (25%) states. The remaining 40 percent are distributed across the other 13 states.

Figure 1 illustrates the Data Model of VeggieChain, which uses MongoDB to store their data. In this system, VeggieChain persists:

1. **Product**s and their **Suppliers**. The product categories are vegetable, fruit or seasoning. A **Supplier** can offer products in one or all of these categories. For example, Supplier A can offer spinach, apple, and salt.
2. Disjoint retail **Store**s in different states.
3. The **Inventory** of each store to track available or sold out items.
4. The **Invoice** of each store. Each invoice contains items sold to a customer. The sold items are stored as a **Sale** document.
5. The **Purchase** that each store made for a product.

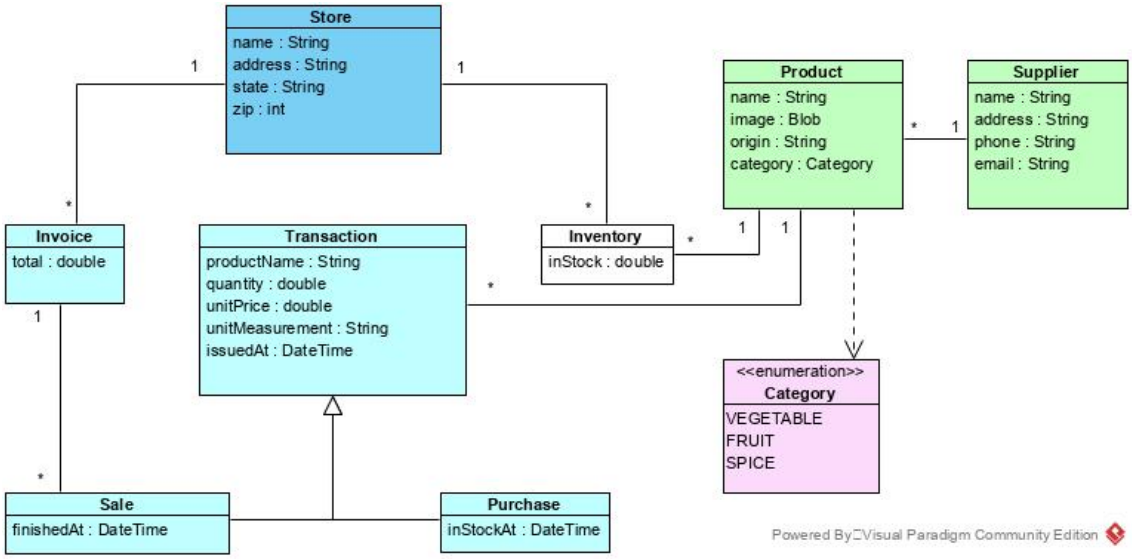

Figure 1: Data Model of Veggie Chain

Given the above scenario, which strategy (horizontal or vertical scaling) will you apply to scale the application? Justify your decision and answer the following questions:

- Which sharding technique from the lecture (slide. 54-56) will you use?
- How do you divide the data into shard?
- What should you concern when applying the sharding technique?

Exercise 3: Map Reduce

As you learned in the lecture, *MapReduce* is used to analyze large data sets in a distributed manner. The following exercises shall give you an idea on how to write simple mappers and reducers.

But first, describe the main phases of MapReduce and name what the execution framework handles.

The input/output domains of map and reduce are as follows[1]:

- $map(k1, v1) \rightarrow list(k2, v2)$
- $reduce(k2, list(v2)) \rightarrow list(v2)$

In between the mapper and the reducer, the MapReduce framework will merge tuples with equal *intermediate key* $k2$ and provide their $v2$ values (i.e. *intermediate values*) as a set.

Table 1: `employees` Table Example

ID	employee	base_salary	salary_level	bonus
1	Alice	50000	4	3000
2	Bob	40000	2	500
3	Carry	55000	8	6000
4	Dorian	45000	2	1000
...	...	...	...	...

Below we provide you with some SQL queries for which you implement the corresponding map reduce functions in **pseudo-code**. To familiarize yourself with the syntax of map reduce functions, refer to the following examples for a sample database table (see table 1):

1. `SELECT employee, salary_level FROM employees WHERE salary_level > 3`

   As you can see, this example can be implemented using a map function only. The `map` function acts like a filter for those records that satisfy the predicate.

```
map (key, record):
  if record.salary_level > 3:
    selection = (record.employee, record.salary_level)
    emit(key, selection)
```

2. `SELECT salary_level, SUM(bonus) FROM employees GROUP BY salary_level`

   Here, the `map` function again filters the entries of interest. The `reduce` function then makes sure that only the according bonuses are summed up.

```
map (key, record):
  emit(record.salary_level, record.bonus)

reduce(ikey, ivalues):
  sum = 0
  for each ivalue in ivalues:
    sum += ivalue
  emit(ikey, sum)
```

Now, it is your task to translate the following SQL queries into equivalent map reduce tasks. Table 2 contains a sample from a database table that depicts the schema for the queries.

Table 2: `attendances` Table

ID	student_id	class_id	date_created	presented
1	101	201	23.12.2019	1
2	102	201	23.12.2019	0
3	103	202	07.01.2020	0
4	104	203	08.01.2020	1
...	...	...	...	...

1. `SELECT * FROM attendances WHERE date_created > 01.01.2020`

2. `SELECT student_id, COUNT(*) FROM attendances WHERE class_id = 201 GROUP BY student_id`

3. `SELECT student_id, COUNT(*) FROM attendances GROUP BY student_id HAVING COUNT(*)>=10 AND SUM(presented)>0`

[1] Dean and Ghemawat, MapReduce: Simplified Data Processing on Large Clusters (2004)

Practical Exercise

In this week's practical exercise, you will learn how to interact with a REST API using Python's *requests*[2] library. This library is the standard way for making HTTP requests with Python. For this exercise, you will only need the most common HTTP methods, but for a deeper understanding of the requests library, you can visit https://realpython.com/python-requests/. You can install requests library with `sudo pip3 install requests` command.

To make things more interesting, you will also use a photoresistor (brightness sensor) with your Raspberry Pi. Using this device, one can measure the light level of an environment and take an action based on the value. Just like you did with an LED and an RFID scanner, you can also control a photoresistor from your Raspberry Pi, using the GPIO pins. However, you must be careful to set up the pin which controls the photoresistor as an "input" pin (unlike LED and RFID scanner pins which are set up as "output" pins). You can read the value of the pin with `GPIO.input(<GPIO #>)` function. This function will return **1** in light conditions and **0** in dark conditions. (To create a dark condition, just cover the sensor with your hands). Visit https://tutorials-raspberrypi.com/photoresistor-brightness-light-sensor-with-raspberry-pi/ to learn more about how to work with a photosensor.

Exercise 4: Consuming a REST API with Python

As a self-motivated developer, you have built yourself a smart mirror that displays weather, calendar, news, and some other information. For this project, you have used a Raspberry Pi 3 Model B. Lately, you have become interested in Asteroids, especially the ones getting closer to Earth (check out Figure 2). For this matter, you plan to add new functionality to your smart mirror such that, when the nighttime arrives, it shows a list of Asteroids based on their closest approach date to Earth.

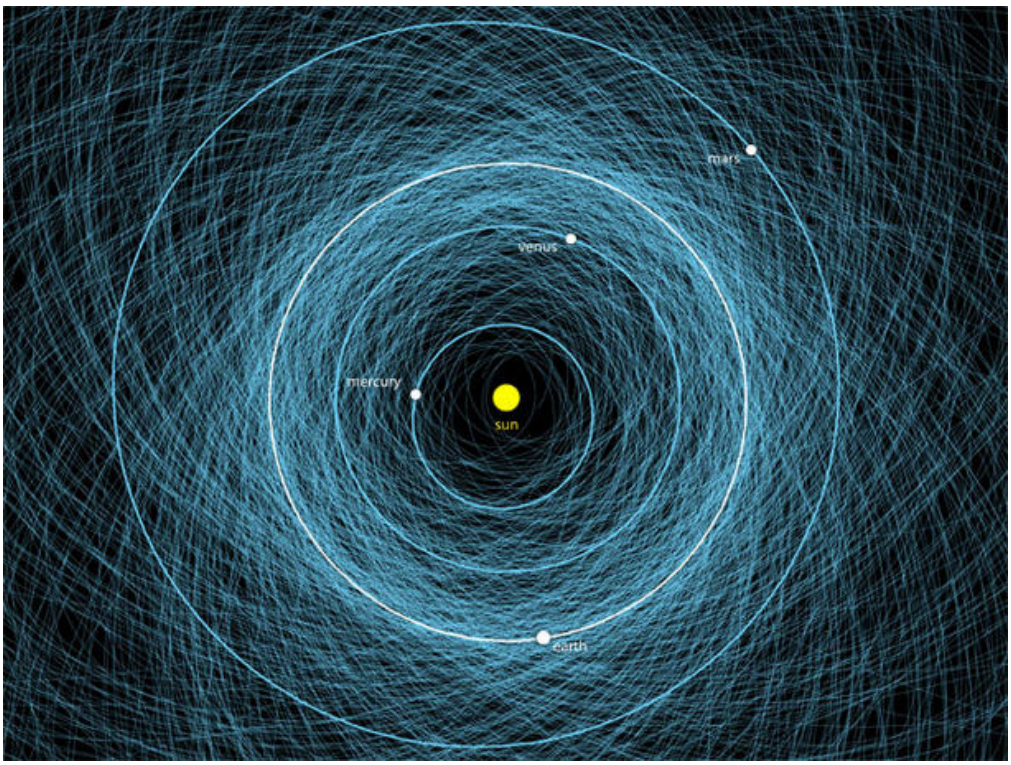

Figure 2: A visualization of Asteroids in our solar system.[3]

1. Search the NASA API Portal (https://api.nasa.gov/) and find a suitable REST API for this task.
2. Write a Python script that checks whether it is nighttime by reading the value of the photosensor connected to your Raspberry Pi. If it is nighttime, your script should retrieve the top 3 Asteroids (for the current day) that have the closest approach date to Earth.

[2] https://docs.python-requests.org/en/latest/
[3] https://www.amnh.org/exhibitions/beyond-planet-earth-space-exploration/exploring-near-earth-asteroids/death-from-the-skies

3. For each Asteroid, output:
    - reference id
    - name
    - absolute magnitude (brightness of a celestial object if it was placed 32.6 (10 parsecs) light-years away from Earth)
    - relative velocity (km)
    - miss distance (km)
    - hazard condition

    You can assume that the rest of the functionalities of the smart mirror are already implemented.
    An example terminal output:

```
burakoz@Buraks-MacBook-Pro-13 Desktop % python3 main.py
---------Asteroids Approaching Earth---------
Date: 2021-10-13
*****************************
reference id: 3428531
name: (2008 SY150)
absolute magnitude (H): 24.7
miss distance (km): 38212310.608361122
relative velocity (km/h): 56440.4154705896
potentially hazardous: False
*****************************
reference id: 3673167
name: (2014 KA91)
absolute magnitude (H): 25.5
miss distance (km): 24287902.733656462
relative velocity (km/h): 20714.1627635631
potentially hazardous: False
*****************************
reference id: 3836100
name: (2018 VF4)
absolute magnitude (H): 25.3
miss distance (km): 31043596.355817898
relative velocity (km/h): 40740.4735817098
potentially hazardous: False
```

Exercise 5: Interacting with the Backend of the Project Application

In last week's practical exercise, you have built a *Spring* backend for the Project application. Now, you are going to practice how to consume the provided endpoints from a Python script.

The following code piece is responsible for creating a new project. For this purpose, it first gets a new XSRF token. Then, by using this token, it calls the auth endpoint to receive a JWT token. Finally, using the JWT token and a project object calls the respective endpoint for creating a new project. Follow the instructions and complete the code piece.

```
import requests
from requests import Session

hostname = 'localhost'
port = 8080
hostUrl = "http://" + hostname + ":" + str(port)

# Use session so we don't have to rewrite the cookies and JWT for every request
session = requests.Session()

params = {
    'mode': "cors",
    "cache":"no-cache",
    "credentials":"include",
    "redirect":"follow",
    "referrerPolicy":"origin-when-cross-origin"
}

def httpRequest(method, url, params, headers='', content='', auth=''):
    if method == 'GET':
        res = session.get(url,params=params)
        return res
    elif method == 'POST':
        if auth == '':
            res = session.post(url,params=params, headers=headers, json=content)
        else:
            res = session.post(url,params=params, headers=headers, auth=auth)
        return res
    else:
        raise ValueError('Method Not Found')

def getBaseHeaders(xsrf_token):
    return {
        "Content-Type": "application/json",
        "X-XSRF-TOKEN": xsrf_token
    }

def getXSRFToken ():
    r = httpRequest('GET', hostUrl + '/auth/csrf', params)

    print(r.status_code)
    # 1. CHECK RESPONSE STATUS AND RETURN THE XSRF TOKEN OR THROW AN EXCEPTION

def auth(xsrf_token):
    r = httpRequest(
        'POST',
        hostUrl + '/auth',
        params,
        # 2. INCLUDE THE BASE HEADERS
        # 3. USE BASIC AUTH
    )

    print('authStatusCode=', r.status_code)
    # 4. CHECK RESPONSE STATUS AND RETURN THE JWT TOKEN OR THROW AN EXCEPTION

def createProject(content, xsrf_token):
    r = httpRequest(
        'POST',
        hostUrl + '/project',
        params,
        # 5. INCLUDE THE BASE HEADERS
        # 6. ADD THE REQUEST BODY
    )

    print("Status code insert project", r.status_code)
    # 7. CHECK RESPONSE STATUS AND RETURN PROJECTS OR THROW AN EXCEPTION

```

```
68      token = getXSRFToken()
69      jwt = auth(token)
70      res = createProject({"name": "ASE Smart PC3"}, token)
```