Report DS Lab 3

Alexander Michielsen, Group: **#b10197**

1. Briefly explain the logic behind your Naming server.

We use spring boot for our application, just as last time. We implemented following functions:

```java
// Getting all the hosts
@GetMapping(path ="/hosts")
public static TreeMap<Integer,Inet4Address> getHosts() { return namingService.getDatabase(); }

// Adding 1 host
@PostMapping(path ="/host")
public static String addHost(@RequestParam(value = "host") String hostname,@RequestParam(value = "ip") String ip ) {
    JSONObject jsonObject = new JSONObject();
    Integer hash = namingService.addIpAddress(hostname, ip);
    jsonObject.put("hostname", hostname);
    jsonObject.put("ip", ip);
    if (hash == -1) {
        jsonObject.put("status", "failed - entry already exists");
        return jsonObject.toString();
    }
    jsonObject.put("hash", hash);
    jsonObject.put("status", "success");
    return jsonObject.toString();
}

// Delete 1 host
@DeleteMapping(path ="/host")
public static String deleteHost(@RequestParam(value = "host") String host) {
    JSONObject jsonObject = new JSONObject();
    String status = namingService.deleteIpAddress(host) ? "success": "failed";

    jsonObject.put("status", status);
    return jsonObject.toJSONString();
}

// Get IP from filename
@GetMapping(path ="/file2host")
public static String getHostIp(@RequestParam(value = "filename") String filename) {
    JSONObject jsonObject = new JSONObject();
    String ip = namingService.getIpAddress(filename).getHostAddress();
    System.out.println(Hash.generateHash(filename));
    jsonObject.put("ip", ip);
    jsonObject.put("filename", filename);
    return jsonObject.toString();
}
```

Our naming server is built using a TreeMap which contain a <Hash Value (int), IP address> pair.

```java
@Service
public class NamingService {

    private TreeMap<Integer, Inet4Address> database;
    private final Hash hashGen = new Hash();
    private ReadWriteLock lock = new ReentrantReadWriteLock();
    private Lock writeLock = lock.writeLock();
    private Lock readLock = lock.readLock();
    public NamingService() { database = XMLRead.serverList(); }

    public Inet4Address getIpAddress(String filename){
        try{
            readLock.lock();
            NodeFinder nodeFinder = new NodeFinder(hashGen, database);
            int value = nodeFinder.findNodeFromFile(filename);
            return database.get(value);
        }finally {
            readLock.unlock();
        }

    }

    public Integer addIpAddress(String hostname, String ip) {
        try {
            if (!database.containsKey(Hash.generateHash(hostname))) {
                Integer hash = Hash.generateHash(hostname);
                try{
                    writeLock.lock();
                    database.put(hash, (Inet4Address) InetAddress.getByName(ip));
                }finally {
                    writeLock.unlock();
                }
                XMLWrite.serverList(database);
                return hash;
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
        return -1;
    }
```

```
    public boolean deleteIpAddress(String ip) {
        boolean status = false;
        try{
            writeLock.lock();
            status = database.keySet().removeIf(key -> key == Hash.generateHash(ip));
        }finally {
            writeLock.unlock();
        }
        XMLWrite.serverList(database);
        return status;
    }

    public TreeMap<Integer, Inet4Address> getDatabase() {
        try{
            readLock.lock();
            return database;
        }finally {
            readLock.unlock();
        }
    }
}
```

We use a TreeMap so that the keys (hash values) are ordered. This makes it easier to find the node that holds the file later on. We use locks to deal with concurrency so that no two threads can access the map simultaneous when it's being modified.

Every time an entry is added/removed from the TreeMap, the corresponding XML file gets updated. This is done to maintain the information after the program closes. When the application is started, the XML automatically gets read in into the map (as can be seen in the constructor).

Whenever we add an entry to our service we pass a hostname and IP address. We will generate a hash value from our hostname and save this together with the IP address in the database. (On our first iteration, we hashed the IP address, this is also an option). We return the hash indicating success.

We also check whether the hash value is already present in the database. If that's the case, the function returns -1 indicating failure.

To get an IP address from a filename we use our own Nodefinder class:

```java
/**
 * Class for finding the corresponding node given a filename
 * --Alexander Michielsen
 */
public class NodeFinder {

    private Hash hashGenerator;
    private Map<Integer, Inet4Address> nodeMap;

    public NodeFinder(Hash hashGenerator, Map<Integer, Inet4Address> nodeMap) {
        this.hashGenerator = hashGenerator;
        this.nodeMap = nodeMap;
    }

    public int findNodeFromFile(String filename) {
        int hashCode = hashGenerator.generateHash(filename);
        Set<Integer> nodes = nodeMap.keySet();
        TreeSet<Integer> treeNodes = new TreeSet<>(nodes);
        Integer smaller = treeNodes.lower(hashCode);
        return Objects.requireNonNullElse(smaller, treeNodes.last()); //Wrap around
    }
}
```

This works by generating the hash of the filename. Then we put the hash values of the nodes in a TreeSet. And then we can just call the lower method which returns the greatest element strictly less.

If there is no such element, which means that the value is lower than the lowest node hash, it will return the last element in the list = highest node hash value.

2. How did you implement hashing?

```java
public class Hash {

    public static int generateHash(String hostname){
        int hashCode = hostname.hashCode();
        double max = Integer.MAX_VALUE;
        double min = Integer.MIN_VALUE;
        double hashValue  = (hostname.hashCode()+ max)*(32768/(max +abs(min)));
        return (int)hashValue;
    }
}
```

For the hashing function we implemented the one from the slides.

This one limits the range to **(0, 32768)** instead of **(-2147483648, 2147483648)** which is what the standard Java hashing function returns.

This is done as follows:

-We add max to the output to ensure the result will be a positive value

-Then we divide it by the total possible range (max + abs(min))

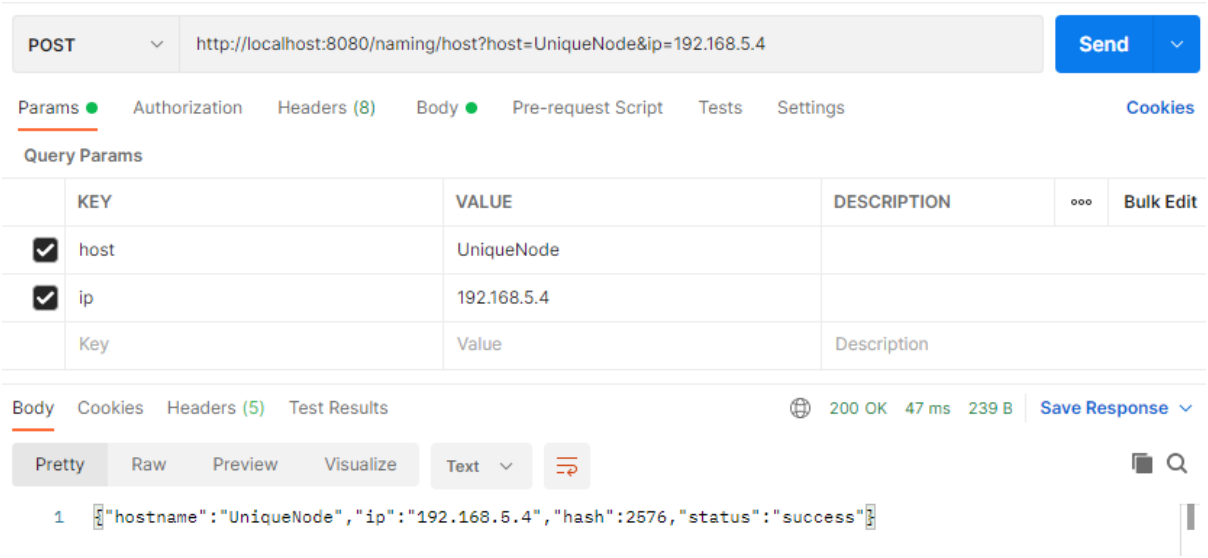
Now we have normalized the hash value (the range is **(0,1)**)

-Finally we multiply this result with 32768 to set the range to **(0,32768)**.

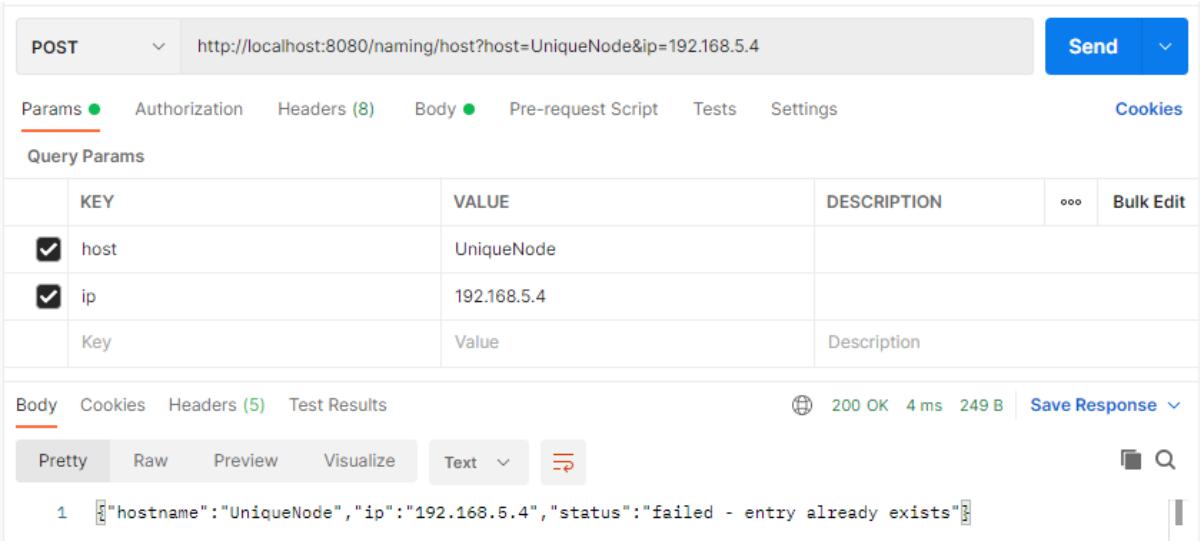
To generate a hash value, you can just call this function and pass the String you want to hash.

3.Provide screenshot for at least three tests out of the following:

1. Add a node with a unique node name.

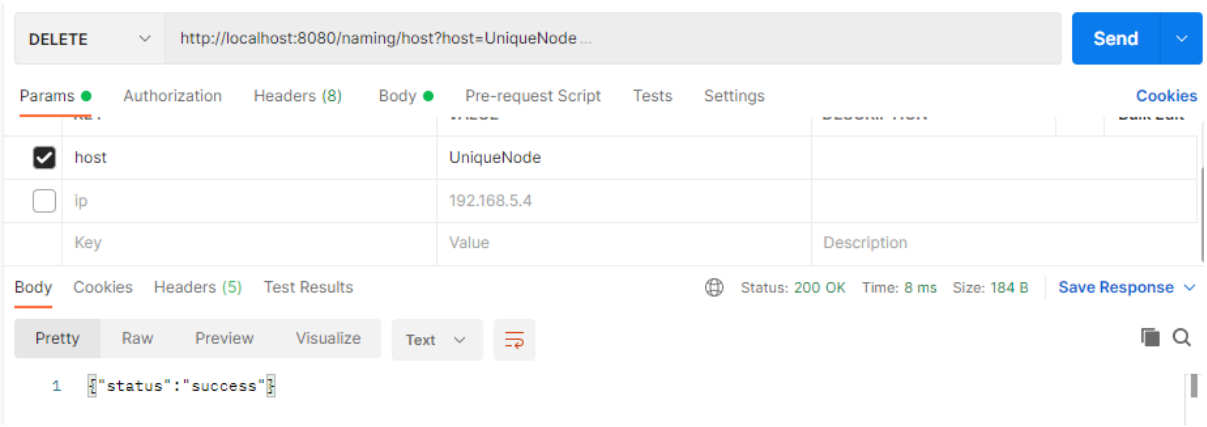


2. Add a node with an existing node name.

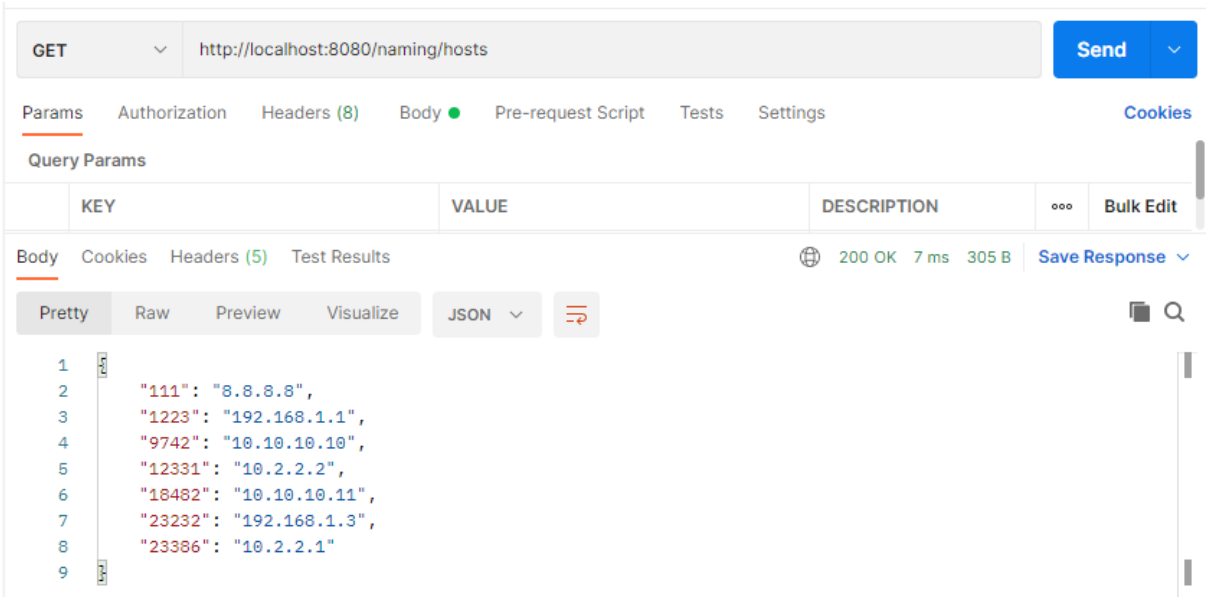


Failure when trying to add a node with the same name

3. Remove a node from the MAP.

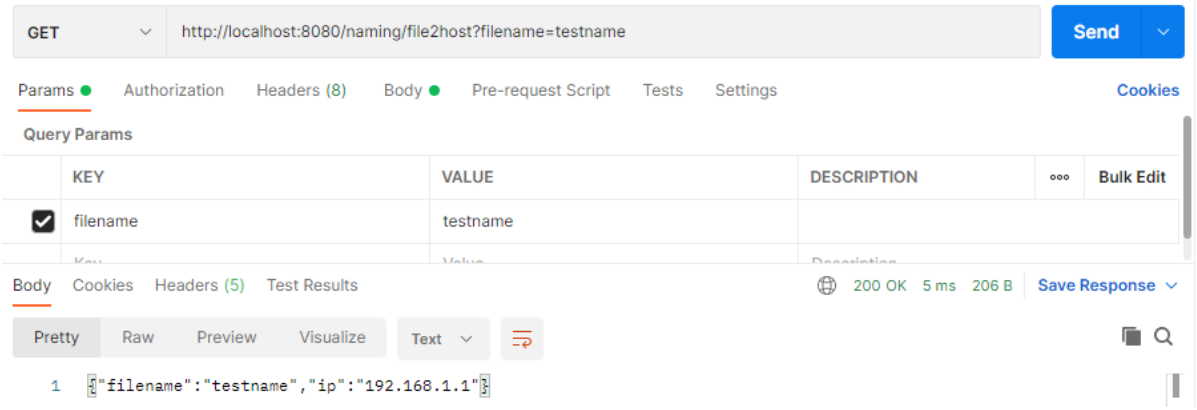


We can check all the current hosts to ascertain deletion



The host has been removed

4.Send a filename and ping the IP address.

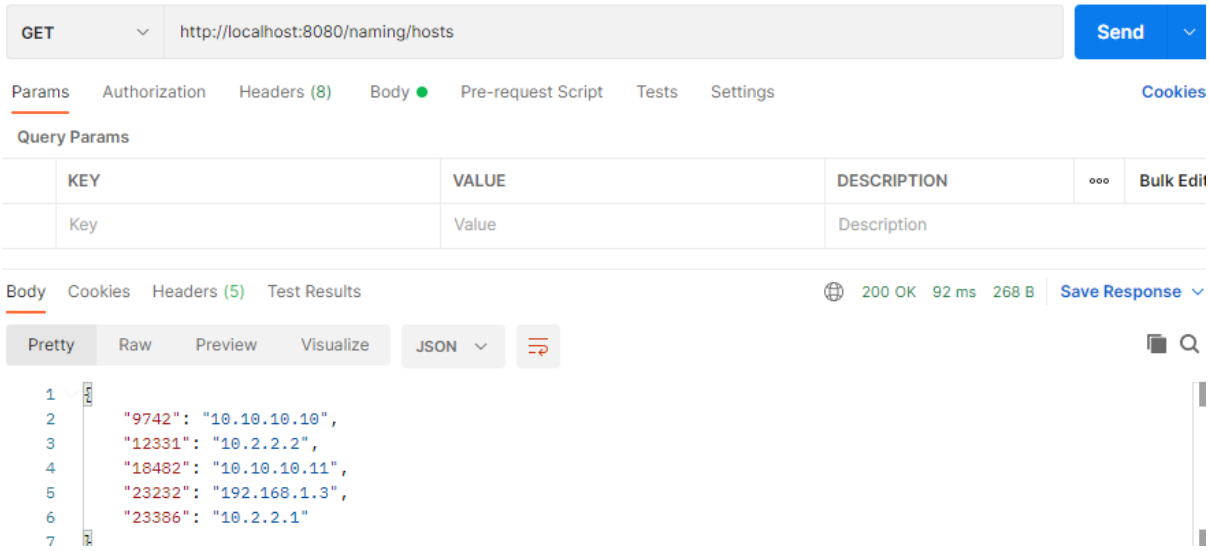


We could try to ping the address but this is actually a mock-up so it wouldn't do much.

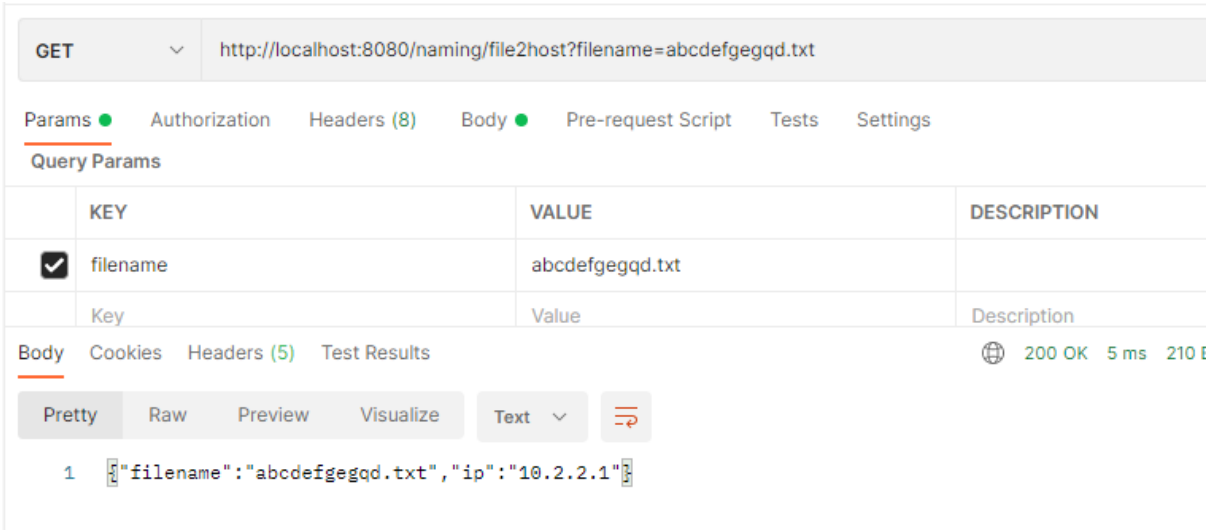
5.Send a filename with a hash smaller than the smallest hash of the nodes.

From the list above, it can be seen that the smallest hash is 1111. We remove this one and the next for this test because it's hard to find a string with lower hash value.

So the database consists of:



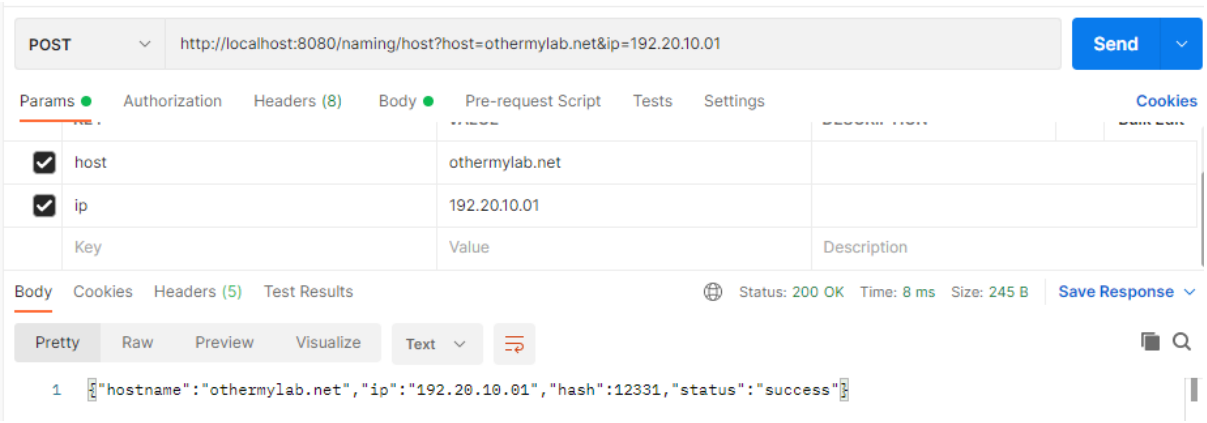
If we try to search for filename `"abcdefgegqd.txt"` (hash = 898):

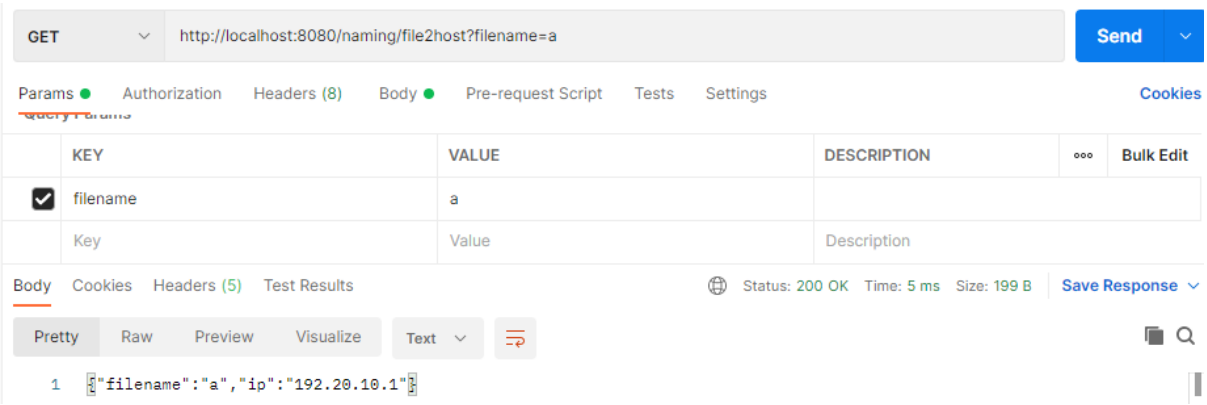


It actually returns the last one in the list

6. Send a filename and at the same time remove the node.

In order to test this, we first add the following hostname





Filename ‘a’ will map to this one

Now we remove the node and send the file name simultaneous using curl

```
C:\Users\amich>curl --location --request DELETE "http://localhost:8080/naming/host?host=othermylab.net" & curl --locatio
n --request GET "http://localhost:8080/naming/file2host?filename=a"
{"status":"success"}{"filename":"a","ip":"10.2.2.1"}
```

We see that the node gets deleted successfully and that the new IP address is another one.

However, sometimes it’s the other way around and the old IP address is actually returned. It just depends on which request reaches the server first because they will never actually be exactly simultaneous