CS 549—Fall 2017
Distributed Systems and Cloud Computing
Assignment Two—RESTful DHT

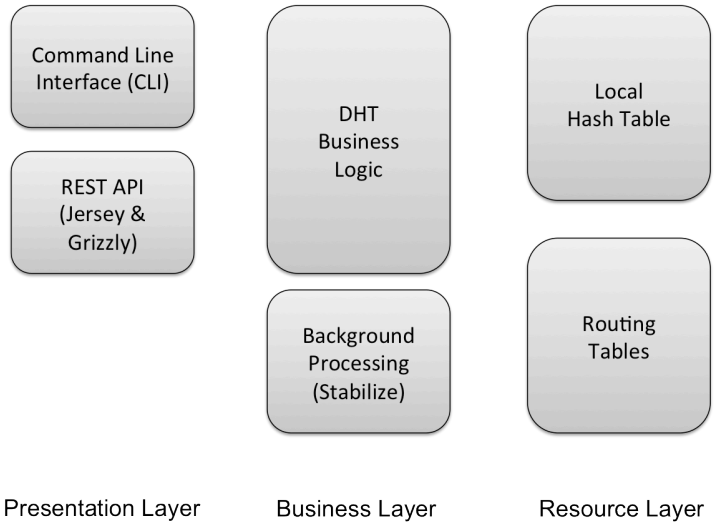
For this assignment, you will complete a simple peer-to-peer structured overlay network, a distributed hash table (DHT), to be partly deployed into the cloud. You are provided with the partial implementation of a DHT using REST for communication among nodes. Your assignment is to complete this implementation using JAX/RS, specifically using the Jersey implementation of JAX/RS. The nodes of your network will run as standalone Java programs, using an embedded Grizzly Web server.

Each node has its own "state server." This state server could be implemented as a singleton RMI object, located in a separate process that functions as a lightweight "database server." For simplicity, the server in this case is an in-process singleton object. When an application starts, it creates a state server object and registers it internally. When a Web service request arrives, the handler for that request looks up the state server in the main program state. Concurrent requests synchronize their accesses to the node's state using locks in the state server.

The code is structured as a Maven project called "`dht-standalone`," with several packages:

1. `edu.stevens.cs549.dht.activity` contains business logic for the overlay network.
2. `edu.stevens.cs549.dht.main` contains the main program, as well as a CLI and a class encapsulating Web client requests.
3. `edu.stevens.cs549.dht.resource` contains Web resources for the REST API.
4. `edu.stevens.cs549.dht.state` contains the state server implementation.

The overall structure of the application is described by this picture:



The POM file for the main project defines several properties that are used in the project. You do not need to learn Maven or POM files, and in this case you should not need to modify this root POM.

If you build the jar file for the project ("`Run as | maven install`" in Eclipse), it is placed in

```
${server.home}/tmp/cs549/dht-test/dht.jar
```

Make sure that this directory is defined. To run the program, type

```
java -jar dht.jar --http http --id id
```

The parameter *`http`* is the port number for the Web server that runs as part of the app, while *`id`* is the identity that this node takes on in the DHT. This will start a command line interface once it has started to the Web server. Type "`help`" for information about the CLI commands. In contrast to the previous assignment, the application you develop is both a client and a server, each with its own CLI. You run multiple instances of the application, and they communicate with each other based on the commands you provide in the CLI. The more instances you run, the better. It should even be possible for instances belonging to different students in the class to interoperate.

To incorporate the code you are provided with into Eclipse, select "`File | Import…`" You are given a pop-up window with a list of import wizards. Select the "`Maven`" tab, then choose "`Existing Maven Projects,`" and navigate to the directory you obtained when you unzipped the archive file you have been provided with. This will import the Maven project "`dht-standalone`" into your Eclipse workspace.

Once you have your code running locally, you should test it on an EC2 instance that you define. Copy the server jar file to the EC2 instance using "`scp`," using the "`-i`" option to define the private key file that you use to authenticate yourself to the instance. Then ssh to the instance and run the server script. It should be in a directory of the form

```
/home/ec2-user/tmp/cs549/dht-test
```

that you should have created.

To complete the assignment, you need to complete the DHT logic, and you need to complete the RESTful API. Your application should provide the following API:

Verb	**URI**	**Specification**
GET	/dht/info	Retrieve the key and URI for the node
GET	/dht/find?id=ID	Search the network for the successor of ID.
GET	/dht?key=KEY	Retrieve the bindings for a (string) key at a node. This operation, and the two operation below, are only local and do not involve searching the network.
PUT	/dht?key=KEY&val=VAL	Add a binding for a key at a node.
DELETE	/dht?key=KEY&val=VAL	Delete the (first) binding for VAL under KEY.
GET	/dht/succ	Get the successor for the node
GET	/dht/pred	Get the predecessor for the node
GET	/dht/finger?id=ID	Get the closest preceding finger table entry for ID.
PUT	/dht/notify	Notify the node of a new predecessor node, specified in the input element (representation). If the node accepts the new predecessor, it returns a representation of its bindings, some of which are taken over by the predecessor.

The semantics of these operations are described in the lectures on P2P networks, particularly for the Chord system.

Once you have your code working, please follow these instructions for submitting your assignment:

- Export your Eclipse project to the file system.
- Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory `Humphrey_Bogart`.
- In that directory you should provide the zip archive that contains your sources, and the server jar file, `dht.jar`.
- Also include in the directory a report of your submission. This report should be in PDF format. **Do not provide a Word document.**

The content of the report is very important. A missing or sloppy report will hurt your final grade, even if you get everything working. In this report, describe how you completed the code to obtain a working implementation. You should include critical code snippets with explanation, with a focus on the REST API and the business logic for the DHT. You should also describe how you tested the code. Again, it is very important for your grade that you do adequate testing. You should at a minimum demonstrate testing of these scenarios:

1. Joining *at least three nodes* together into a DHT network. Run these nodes on separate EC2 instances under your control.
2. Adding, retrieving and deleting bindings, at both local and remote nodes (with respect to the CLI of the instance where you are testing). Use the `bindings` and `routes` commands of the CLI to show the functioning of your network, including the states of the nodes after you perform operations on the network.
3. Record one or more short videos demonstrating your code working. Make sure that your name appears at the beginning of the video. For example, display the contents of a file that provides your name. *Do not provide private information such as your email or cwid in the video*.
4. For example, test with the following bindings, with nodes with keys 23, 37 and 45:

```
foo -> 6
foo2 -> 44
foo3 -> 44
bar -> 19
baz -> 27
baz2 -> 55
fubar -> 36
xyz -> 25
```

Finally note any other changes you made to the materials provided, with an explanation.

Remember the format of the submission: A zip archive file, named after you, with a directory named after you. In this directory, provide three files: a zip archive of your source files and resources, a server jar file, and a report for your submission. Be sure to include videos demonstrating your app working, covering the scenarios described above, in your documentation.