Acceptance Test Cases

# grading repository

## GradeP4.java

package ece448.grading;

import java.util.Arrays;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.apache.http.client.fluent.Request;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import ece448.iot\_sim.SimConfig;

import ece448.grading.GradeP3.MqttController;

import ece448.iot\_hub.HubConfig;

public class GradeP4 implements AutoCloseable {

private static final String broker = "tcp://127.0.0.1";

private static final String topicPrefix = System.currentTimeMillis()+"/grade\_p4/iot\_ece448";

private static final List<String> plugNames = Arrays.asList("a", "b", "c");

private static final List<String> plugNamesEx = Arrays.asList("d", "e", "f", "g");

private static final List<String> allPlugNames = Arrays.asList("a", "b", "c", "d", "e", "f", "g");

private static final ObjectMapper mapper = new ObjectMapper();

private static final Logger logger = LoggerFactory.getLogger(GradeP4.class);

private final MqttController mqtt;

private GradeP4() throws Exception {

this.mqtt = new MqttController(broker, "grader/iot\_hub", topicPrefix);

this.mqtt.start();

}

@Override

public void close() throws Exception {

mqtt.close();

}

public static void main(String[] args) throws Exception {

SimConfig config = new SimConfig(8080, plugNames, broker, "testee/iot\_sim", topicPrefix);

SimConfig configEx = new SimConfig(8081, plugNamesEx, broker, "ex\_testee/iot\_sim", topicPrefix);

HubConfig hubConfig = new HubConfig(8088, broker, "testee/iot\_hub", topicPrefix);

try (

GradeP4 p4 = new GradeP4();

ece448.iot\_sim.Main m = new ece448.iot\_sim.Main(config);

ece448.iot\_sim.Main mex = new ece448.iot\_sim.Main(configEx);

ece448.iot\_hub.Main hub = new ece448.iot\_hub.Main(hubConfig, new String[0]))

{

Grading.run(p4, 10);

}

}

static String getSim(String pathParams) throws Exception {

return Request.Get("http://127.0.0.1:8080" + pathParams)

.userAgent("Mozilla/5.0").connectTimeout(1000)

.socketTimeout(1000).execute().returnContent().asString();

}

static String getSimEx(String pathParams) throws Exception {

return Request.Get("http://127.0.0.1:8081" + pathParams)

.userAgent("Mozilla/5.0").connectTimeout(1000)

.socketTimeout(1000).execute().returnContent().asString();

}

static String getHub(String pathParams) throws Exception {

return Request.Get("http://127.0.0.1:8088" + pathParams)

.userAgent("Mozilla/5.0").connectTimeout(1000)

.socketTimeout(1000).execute().returnContent().asString();

}

static String getStates1() throws Exception {

TreeMap<String, String> states = new TreeMap<>();

for (String name: allPlugNames)

{

Map<String, Object> plug = mapper.readValue(getHub("/api/plugs/" + name),

new TypeReference<Map<String, Object>>() {});

if (!name.equals((String)plug.get("name")))

throw new Exception("invalid name " + name);

states.put(name, "off".equals((String)plug.get("state"))? "0": "1");

}

String ret = String.join("", states.values());

logger.debug("GradeP4: getState1 {}", ret);

return ret;

}

static String getStates2() throws Exception {

TreeMap<String, String> states = new TreeMap<>();

HashSet<String> known = new HashSet<>(allPlugNames);

List<Map<String, Object>> plugs = mapper.readValue(getHub("/api/plugs"),

new TypeReference<List<Map<String, Object>>>() {});

for (Map<String, Object> plug: plugs)

{

String name = (String)plug.get("name");

String state = (String)plug.get("state");

if (!known.contains(name))

throw new Exception("invalid plug " + name);

known.remove(name);

states.put(name, "off".equals(state)? "0": "1");

}

if (!known.isEmpty())

throw new Exception("missing plugs");

String ret = String.join("", states.values());

logger.debug("GradeP4: getState2 {}", ret);

return ret;

}

static String getStates3() throws Exception {

TreeMap<String, String> states = new TreeMap<>();

for (String name: plugNames)

{

String ret = getSim("/"+name);

if ((ret.indexOf(name+" is off") != -1) && (ret.indexOf(name+" is on") == -1))

{

states.put(name, "0");

}

else

{

states.put(name, "1");

}

}

for (String name: plugNamesEx)

{

String ret = getSimEx("/"+name);

if ((ret.indexOf(name+" is off") != -1) && (ret.indexOf(name+" is on") == -1))

{

states.put(name, "0");

}

else

{

states.put(name, "1");

}

}

String ret = String.join("", states.values());

logger.debug("GradeP4: getState3 {}", ret);

return ret;

}

static String getStates4(MqttController mqtt) throws Exception {

TreeMap<String, String> states = new TreeMap<>();

for (String name: allPlugNames)

{

states.put(name, "off".equals(mqtt.getState(name))? "0": "1");

}

String ret = String.join("", states.values());

logger.debug("GradeP4: getState4 {}", ret);

return ret;

}

static boolean verifyStates(String states, MqttController mqtt) throws Exception {

return states.equals(getStates1())

&& states.equals(getStates2())

&& states.equals(getStates3())

&& states.equals(getStates4(mqtt));

}

public boolean testCase00() throws Exception {

return "0000000".equals(getStates1());

}

public boolean testCase01() throws Exception {

getHub("/api/plugs/a?action=on");

getHub("/api/plugs/c?action=toggle");

Thread.sleep(1000);

return "1010000".equals(getStates1());

}

public boolean testCase02() throws Exception {

getHub("/api/plugs/a?action=toggle");

getHub("/api/plugs/c?action=off");

getHub("/api/plugs/e?action=on");

getHub("/api/plugs/g?action=toggle");

Thread.sleep(1000);

return "0000101".equals(getStates1());

}

public boolean testCase03() throws Exception {

getHub("/api/plugs/a?action=off");

getHub("/api/plugs/b?action=on");

getHub("/api/plugs/c?action=off");

getHub("/api/plugs/d?action=toggle");

getHub("/api/plugs/e?action=on");

getHub("/api/plugs/f?action=off");

getHub("/api/plugs/g?action=toggle");

Thread.sleep(1000);

return "0101100".equals(getStates2());

}

public boolean testCase04() throws Exception {

getHub("/api/plugs/b?action=off");

getHub("/api/plugs/d?action=on");

getHub("/api/plugs/f?action=on");

Thread.sleep(1000);

return "0001110".equals(getStates2());

}

public boolean testCase05() throws Exception {

getSim("/b?action=on");

Thread.sleep(1000);

return verifyStates("0101110", mqtt);

}

public boolean testCase06() throws Exception {

getSimEx("/d?action=off");

Thread.sleep(1000);

return verifyStates("0100110", mqtt);

}

public boolean testCase07() throws Exception {

mqtt.publishAction("c", "on");

mqtt.publishAction("e", "off");

Thread.sleep(1000);

return verifyStates("0110010", mqtt);

}

public boolean testCase08() throws Exception {

getSim("/a?action=toggle");

mqtt.publishAction("d", "toggle");

getSimEx("/e?action=toggle");

mqtt.publishAction("g", "toggle");

Thread.sleep(1000);

return verifyStates("1111111", mqtt);

}

public boolean testCase09() throws Exception {

getHub("/api/plugs/a?action=off");

mqtt.publishAction("b", "toggle");

getSim("/c?action=off");

getSimEx("/d?action=toggle");

getHub("/api/plugs/e?action=toggle");

mqtt.publishAction("f", "off");

getSimEx("/g?action=off");

Thread.sleep(1000);

return verifyStates("0000000", mqtt);

}

}