# ~/iot\_ece448/src/main/java/ece448/iot\_hub

## App.java

package ece448.iot\_hub;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication

@EnableScheduling

public class App {

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

@Bean

public PlugsModel plugsModel() {

return new PlugsModel(null);

}

}

## HubConfig.java

package ece448.iot\_hub;

import com.fasterxml.jackson.annotation.JsonCreator;

import com.fasterxml.jackson.annotation.JsonProperty;

public class HubConfig {

private final int httpPort;

private final String mqttBroker;

private final String mqttClientId;

private final String mqttTopicPrefix;

@JsonCreator

public HubConfig(

@JsonProperty(value = "httpPort", required = true) int httpPort,

@JsonProperty(value = "mqttBroker", required = true) String mqttBroker,

@JsonProperty(value = "mqttClientId", required = true) String mqttClientId,

@JsonProperty(value = "mqttTopicPrefix", required = true) String mqttTopicPrefix) {

this.httpPort = httpPort;

this.mqttBroker = mqttBroker;

this.mqttClientId = mqttClientId;

this.mqttTopicPrefix = mqttTopicPrefix;

}

public int getHttpPort() {

return httpPort;

}

public String getMqttBroker() {

return mqttBroker;

}

public String getMqttClientId() {

return mqttClientId;

}

public String getMqttTopicPrefix() {

return mqttTopicPrefix;

}

}

## Main.java

package ece448.iot\_hub;

import java.io.File;

import java.util.HashMap;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.context.ConfigurableApplicationContext;

public class Main implements AutoCloseable {

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

// load configuration file

String configFile = args.length > 0 ? args[0] : "hubConfig.json";

HubConfig config = mapper.readValue(new File(configFile), HubConfig.class);

logger.info("{}: {}", configFile, mapper.writeValueAsString(config));

try (Main m = new Main(config, args))

{

// loop forever

for (;;)

{

Thread.sleep(60000);

}

}

}

public Main(HubConfig config, String[] args) throws Exception {

// Spring app

HashMap<String, Object> props = new HashMap<>();

props.put("server.port", config.getHttpPort());

props.put("mqtt.broker", config.getMqttBroker());

props.put("mqtt.clientId", config.getMqttClientId());

props.put("mqtt.topicPrefix", config.getMqttTopicPrefix());

SpringApplication app = new SpringApplication(App.class);

app.setDefaultProperties(props);

this.appCtx = app.run(args);

}

@Override

public void close() throws Exception {

appCtx.close();

}

private final ConfigurableApplicationContext appCtx;

private static final ObjectMapper mapper = new ObjectMapper();

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

}

## MockEnvironment.java

package ece448.iot\_hub;

import org.springframework.core.env.Environment;

import java.util.HashMap;

import java.util.Map;

public class MockEnvironment implements Environment {

private final Map<String, String> properties = new HashMap<>();

@Override

public boolean containsProperty(String key) {

return properties.containsKey(key);

}

@Override

public String getProperty(String key) {

return properties.get(key);

}

@Override

public String getProperty(String key, String defaultValue) {

return containsProperty(key) ? getProperty(key) : defaultValue;

}

public void setProperty(String key, Object value) {

properties.put(key, String.valueOf(value));

}

public void put(String key, Object value) {

setProperty(key, value);

}

@Override

public <T> T getProperty(String key, Class<T> targetType) {

throw new UnsupportedOperationException("Unimplemented method 'getProperty'");

}

@Override

public <T> T getProperty(String key, Class<T> targetType, T defaultValue) {

throw new UnsupportedOperationException("Unimplemented method 'getProperty'");

}

@Override

public <T> Class<T> getPropertyAsClass(String key, Class<T> targetType) {

throw new UnsupportedOperationException("Unimplemented method 'getPropertyAsClass'");

}

@Override

public String getRequiredProperty(String key) throws IllegalStateException {

throw new UnsupportedOperationException("Unimplemented method 'getRequiredProperty'");

}

@Override

public <T> T getRequiredProperty(String key, Class<T> targetType) throws IllegalStateException {

throw new UnsupportedOperationException("Unimplemented method 'getRequiredProperty'");

}

@Override

public String resolvePlaceholders(String text) {

throw new UnsupportedOperationException("Unimplemented method 'resolvePlaceholders'");

}

@Override

public String resolveRequiredPlaceholders(String text) throws IllegalArgumentException {

throw new UnsupportedOperationException("Unimplemented method 'resolveRequiredPlaceholders'");

}

@Override

public String[] getActiveProfiles() {

throw new UnsupportedOperationException("Unimplemented method 'getActiveProfiles'");

}

@Override

public String[] getDefaultProfiles() {

throw new UnsupportedOperationException("Unimplemented method 'getDefaultProfiles'");

}

@Override

public boolean acceptsProfiles(String... profiles) {

throw new UnsupportedOperationException("Unimplemented method 'acceptsProfiles'");

}

}

## PlugsModel.java

package ece448.iot\_hub;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import org.springframework.stereotype.Component;

import ece448.grading.GradeP3.MqttController;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Component

public class PlugsModel {

private final ConcurrentHashMap<String, Plug> plugs = new ConcurrentHashMap<>();

private final MqttController mqtt;

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

public PlugsModel(MqttController mqtt) {

this.mqtt = mqtt;

// Subscribe to state updates

mqtt.subscribeForUpdates((plugName, state, power) -> {

updatePlug(plugName, state, power);

logger.debug("Updated plug {}: state={}, power={}", plugName, state, power);

});

}

public List<String> getPlugs() {

return new ArrayList<>(plugs.keySet());

}

public String getPlugState(String plug) {

Plug p = plugs.get(plug);

return (p != null) ? p.getState() : mqtt.getState(plug);

}

public String getPlugPower(String plug) {

Plug p = plugs.get(plug);

return (p != null) ? p.getPower() : mqtt.getPower(plug);

}

public void updatePlug(String plug, String state, String power) {

plugs.put(plug, new Plug(plug, state, power));

}

public void publishAction(String plug, String action) {

mqtt.publishAction(plug, action);

}

public Map<String, Plug> getAllPlugs() {

// Ensure we have all plugs that MQTT knows about

for (String plugName : mqtt.getStates().keySet()) {

if (!plugs.containsKey(plugName)) {

updatePlug(plugName, mqtt.getState(plugName), mqtt.getPower(plugName));

}

}

return new HashMap<>(plugs);

}

public static class Plug {

private final String name;

private final String state;

private final String power;

public Plug(String name, String state, String power) {

this.name = name;

this.state = state;

this.power = power;

}

public String getName() {

return name;

}

public String getState() {

return state;

}

public String getPower() {

return power;

}

}

}

## PlugsResource.java

package ece448.iot\_hub;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@RestController

public class PlugsResource {

private final PlugsModel plugsModel;

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

public PlugsResource(PlugsModel plugsModel) {

this.plugsModel = plugsModel;

}

@GetMapping("/api/plugs")

public List<Map<String, Object>> getAllPlugs() {

List<Map<String, Object>> result = new ArrayList<>();

Map<String, PlugsModel.Plug> allPlugs = plugsModel.getAllPlugs();

for (String plugName : allPlugs.keySet()) {

result.add(convertPlugToMap(allPlugs.get(plugName)));

}

logger.debug("getAllPlugs: returning {} plugs", result.size());

return result;

}

@GetMapping("/api/plugs/{plugName:.+}")

public Map<String, Object> getPlug(

@PathVariable("plugName") String plugName,

@RequestParam(value = "action", required = false) String action) {

if (action != null) {

if (action.equals("on") || action.equals("off") || action.equals("toggle")) {

logger.info("Controlling plug {}: action={}", plugName, action);

plugsModel.publishAction(plugName, action);

} else {

logger.warn("Invalid action for plug {}: {}", plugName, action);

}

}

// Get the latest state

String state = plugsModel.getPlugState(plugName);

String power = plugsModel.getPlugPower(plugName);

// Create and return the response

Map<String, Object> result = new HashMap<>();

result.put("name", plugName);

result.put("state", state);

result.put("power", Integer.parseInt(power));

logger.debug("getPlug {}: state={}, power={}", plugName, state, power);

return result;

}

private Map<String, Object> convertPlugToMap(PlugsModel.Plug plug) {

Map<String, Object> map = new HashMap<>();

map.put("name", plug.getName());

map.put("state", plug.getState());

map.put("power", Integer.parseInt(plug.getPower()));

return map;

}

}

# ~/iot\_ece448/src/main/java/ece448/iot\_sim

## HttpCommands.java

package ece448.iot\_sim;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import ece448.iot\_sim.http\_server.RequestHandler;

public class HTTPCommands implements RequestHandler {

// Use a map so we can search plugs by name.

private final TreeMap<String, PlugSim> plugs = new TreeMap<>();

public HTTPCommands(List<PlugSim> plugs) {

for (PlugSim plug: plugs)

{

this.plugs.put(plug.getName(), plug);

}

}

@Override

public String handleGet(String path, Map<String, String> params) {

// list all: /

// do switch: /plugName?action=on|off|toggle

// just report: /plugName

logger.info("HTTPCmd {}: {}", path, params);

if (path.equals("/"))

{

return listPlugs();

}

PlugSim plug = plugs.get(path.substring(1));

if (plug == null)

return null; // no such plug

String action = params.get("action");

if (action == null)

return report(plug);

// P2: add your code here, modify the next line if necessary

if (action.equals("on")) {

plug.switchOn();

return report(plug);

}

else if (action.equals("off")) {

plug.switchOff();

return report(plug);

}

else if (action.equals("toggle")) {

if (plug.isOn()) {

plug.switchOff();

}

else {

plug.switchOn();

}

return report(plug);

}

else {

return report(plug);

}

}

protected String listPlugs() {

StringBuilder sb = new StringBuilder();

sb.append("<html><body>");

for (String plugName: plugs.keySet())

{

sb.append(String.format("<p><a href='/%s'>%s</a></p>",

plugName, plugName));

}

sb.append("</body></html>");

return sb.toString();

}

protected String report(PlugSim plug) {

String name = plug.getName();

return String.format("<html><body>"

+"<p>Plug %s is %s.</p>"

+"<p>Power reading is %.3f.</p>"

+"<p><a href='/%s?action=on'>Switch On</a></p>"

+"<p><a href='/%s?action=off'>Switch Off</a></p>"

+"<p><a href='/%s?action=toggle'>Toggle</a></p>"

+"</body></html>",

name,

plug.isOn()? "on": "off",

plug.getPower(), name, name, name);

}

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

}

## Main.java

package ece448.iot\_sim;

import java.io.File;

import java.util.ArrayList;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import ece448.iot\_sim.http\_server.JHTTP;

import org.eclipse.paho.client.mqttv3.IMqttClient;

import org.eclipse.paho.client.mqttv3.MqttClient;

import org.eclipse.paho.client.mqttv3.MqttException;

import org.eclipse.paho.client.mqttv3.MqttCallback;

import org.eclipse.paho.client.mqttv3.MqttMessage;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

public class Main implements AutoCloseable {

private final MqttClient mqttClient;

private final JHTTP http;

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

// load configuration file

String configFile = args.length > 0 ? args[0] : "simConfig.json";

SimConfig config = mapper.readValue(new File(configFile), SimConfig.class);

logger.info("{}: {}", configFile, mapper.writeValueAsString(config));

try (Main m = new Main(config))

{

// loop forever

for (;;)

{

Thread.sleep(60000);

}

}

}

public Main(SimConfig config) throws Exception {

// create plugs

ArrayList<PlugSim> plugs = new ArrayList<>();

for (String plugName: config.getPlugNames()) {

plugs.add(new PlugSim(plugName));

}

// start power measurements

MeasurePower measurePower = new MeasurePower(plugs);

measurePower.start();

// start HTTP commands

this.http = new JHTTP(config.getHttpPort(), new HTTPCommands(plugs));

this.http.start();

//MQTT setup

mqttClient = new MqttClient(config.getMqttBroker(), "iot\_sim");

mqttClient.connect();

MqttCommands mqttCmd = new MqttCommands(plugs, config.getMqttTopicPrefix());

mqttClient.setCallback(new MqttCallback() {

@Override

public void connectionLost(Throwable cause) {

logger.info("Connection Lost: " + cause.getMessage());

}

@Override

public void messageArrived(String topic, MqttMessage message) throws Exception {

logger.info("Recieved MQTT Message on topic : " + topic);

mqttCmd.handleMessage(topic, message);

}

@Override

public void deliveryComplete(IMqttDeliveryToken token) {

logger.info("Delivery complete for token: " + token);

}

});

mqttClient.subscribe(mqttCmd.getTopic(), 0);

//Publishing the updates

MqttUpdates mqttUpd = new MqttUpdates(config.getMqttTopicPrefix(), mqttClient);

for (PlugSim plug : plugs) {

plug.addObserver((name, key, value) -> {

try {

mqttClient.publish(mqttUpd.getTopic(name, key), mqttUpd.getMessage(value));

} catch (Exception e) {

logger.error("Failed to publish {} {} {}", name, key, value, e);

}

});

}

}

@Override

public void close() throws Exception {

http.close();

mqttClient.disconnect();

}

private static final ObjectMapper mapper = new ObjectMapper();

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

}

## MeasurePower.java

package ece448.iot\_sim;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

/\*\*

\* Take power measurements every 1 second.

\*/

public class MeasurePower {

private final List<PlugSim> plugs;

public MeasurePower(List<PlugSim> plugs) {

this.plugs = plugs;

}

public void start() {

Thread t = new Thread(() -> {

try

{

for (;;)

{

measureOnce();

}

}

catch (Throwable th)

{

logger.error("Power: exit {}", th.getMessage(), th);

System.exit(-1);

}

});

// make sure this thread won't block JVM to exit

t.setDaemon(true);

// start measuring

t.start();

}

/\*\*

\* Measure and wait 1s.

\*/

protected void measureOnce() {

try

{

for (PlugSim plug: plugs)

{

plug.measurePower();

}

Thread.sleep(1000);

}

catch (InterruptedException e)

{

}

}

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

public void interrupt() {

// TODO Auto-generated method stub

throw new UnsupportedOperationException("Unimplemented method 'interrupt'");

}

}

## MqttCommands.java

package ece448.iot\_sim;

import java.util.List;

import java.util.TreeMap;

import org.eclipse.paho.client.mqttv3.MqttMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class MqttCommands {

protected final TreeMap<String, PlugSim> plugs;

private final String topicPrefix;

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

public MqttCommands(List<PlugSim> plugs, String topicPrefix) {

this.plugs = new TreeMap<>();

for (PlugSim plug : plugs) {

this.plugs.put(plug.getName(), plug);

}

this.topicPrefix = topicPrefix;

}

public String getTopic() {

return topicPrefix + "/action/#";

}

// Handling incoming MQTT messages

public void handleMessage(String topic, MqttMessage message) {

try {

String[] parts = topic.split("/");

if (parts.length < 2) {

logger.warn("Invalid topic format: {}", topic);

return;

}

String plugName = parts[parts.length-2];

String action = parts[parts.length-1];

PlugSim plug = plugs.get(plugName);

if (plug != null) {

switch (action) {

case "on":

plug.switchOn();

break;

case "off":

plug.switchOff();

break;

case "toggle":

plug.toggle();

break;

default:

logger.warn("Unknown action: {}", action);

}

}

} catch (Exception e) {

logger.error("Error handling MQTT message: {}", e.getMessage(), e);

}

}

public void addPlug(PlugSim plug) {

plugs.put(plug.getName(), plug);

}

}

## MqttUpdates.java

package ece448.iot\_sim;

import org.eclipse.paho.client.mqttv3.MqttClient;

import org.eclipse.paho.client.mqttv3.MqttMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class MqttUpdates {

private final String topicPrefix;

private final MqttClient mqttClient;

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

public MqttUpdates(String topicPrefix, MqttClient mqttClient) {

this.topicPrefix = topicPrefix;

this.mqttClient = mqttClient;

}

// Generating topic for given plug and key

public String getTopic(String name, String key) {

return topicPrefix + "/update/" + name + "/" + key;

}

// Generating MQTT message for given value

public MqttMessage getMessage(String value) {

MqttMessage msg = new MqttMessage(value.getBytes());

msg.setRetained(true);

return msg;

}

// Publishing update to the MQTT broker

public void publishUpdate(String name, String key, String value) {

try {

String topic = getTopic(name, key);

MqttMessage msg = getMessage(value);

mqttClient.publish(topic, msg);

logger.info("Published update: {} -> {}", topic, value);

} catch (Exception e) {

logger.error("Failed to publish update for {} {} {}", name, key, value, e);

}

}

}

## PlugSim.java

package ece448.iot\_sim;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.List;

import java.util.ArrayList;

/\*\*

\* Simulate a smart plug with power monitoring.

\*/

public class PlugSim {

private final String name;

private boolean on = false;

private double power = 0; // in watts

public PlugSim(String name) {

this.name = name;

}

/\*\*

\* No need to synchronize if read a final field.

\*/

public String getName() {

return name;

}

public static interface Observer {

void update (String name, String key, String value);

}

private final List<Observer> observers = new ArrayList<>();

public void addObserver(Observer observer) {

observers.add(observer);

observer.update(name, "state", on ? "on" : "off" );

observer.update(name, "power", String.format("%.3f", power));

}

/\*\*

\* Switch the plug on.

\*/

synchronized public void switchOn() {

// P1: add your code here

on = true;

measurePower();

notifyObservers("state", "on");

}

/\*\*

\* Switch the plug off.

\*/

synchronized public void switchOff() {

// P1: add your code here

on = false;

notifyObservers("state", "off");

}

/\*\*

\* Toggle the plug.

\*/

synchronized public void toggle() {

// P1: add your code here

on = !on;

notifyObservers("state", on ? "on" : "off");

if(on) {

measurePower();

notifyObservers("power", String.format("%.3f", power));

}

}

/\*\*

\* Measure power.

\*/

synchronized public void measurePower() {

if (!on) {

updatePower(0);

return;

}

// a trick to help testing

if (name.indexOf(".") != -1)

{

updatePower(Integer.parseInt(name.split("\\.")[1]));

}

// do some random walk

else if (power < 100)

{

updatePower(power + Math.random() \* 100);

}

else if (power > 300)

{

updatePower(power - Math.random() \* 100);

}

else

{

updatePower(power + Math.random() \* 40 - 20);

}

notifyObservers("power", String.format("%.3f", power));

}

private void notifyObservers(String key, String value) {

for (Observer observer : observers) {

observer.update(name, key, value);

}

}

protected void updatePower(double p) {

power = p;

logger.debug("Plug {}: power {}", name, power);

}

/\*\*

\* Getter: current state

\*/

synchronized public boolean isOn() {

return on;

}

/\*\*

\* Getter: last power reading

\*/

synchronized public double getPower() {

return power;

}

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

}

## SimConfig.java

package ece448.iot\_sim;

import java.util.List;

import com.fasterxml.jackson.annotation.JsonCreator;

import com.fasterxml.jackson.annotation.JsonProperty;

public class SimConfig {

private final int httpPort;

private final List<String> plugNames;

private final String mqttBroker;

private final String mqttClientId;

private final String mqttTopicPrefix;

@JsonCreator

public SimConfig(

@JsonProperty(value = "httpPort", required = true) int httpPort,

@JsonProperty(value = "plugNames", required = true) List<String> plugNames,

@JsonProperty(value = "mqttBroker", required = false) String mqttBroker,

@JsonProperty(value = "mqttClientId", required = false) String mqttClientId,

@JsonProperty(value = "mqttTopicPrefix", required = false) String mqttTopicPrefix) {

this.httpPort = httpPort;

this.plugNames = plugNames;

this.mqttBroker = mqttBroker;

this.mqttClientId = mqttClientId;

this.mqttTopicPrefix = mqttTopicPrefix;

}

public int getHttpPort() {

return httpPort;

}

public List<String> getPlugNames() {

return plugNames;

}

public String getMqttBroker() {

return mqttBroker;

}

public String getMqttClientId() {

return mqttClientId;

}

public String getMqttTopicPrefix() {

return mqttTopicPrefix;

}

}

# ~/iot\_ece448/src/test/java/ece448/iot\_sim

## HttpCommandsTests.java

package ece448.iot\_sim;

import static org.junit.Assert.\*;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.junit.Before;

import org.junit.Test;

public class HTTPCommandsTests {

private HTTPCommands httpCommands;

private PlugSim plug1;

private PlugSim plug2;

private PlugSim plugWithSpecialChar;

@Before

public void setUp() {

plug1 = new PlugSim("plug1");

plug2 = new PlugSim("plug2");

plugWithSpecialChar = new PlugSim("zzzz.789");

List<PlugSim> plugs = new ArrayList<>();

plugs.add(plug1);

plugs.add(plug2);

plugs.add(plugWithSpecialChar);

httpCommands = new HTTPCommands(plugs);

}

@Test

public void testPlugReportDisplay() {

String response = httpCommands.handleGet("/plug1", new HashMap<>());

assertTrue(response.contains("plug1"));

assertTrue(response.contains("plug1 is off"));

assertTrue(response.contains("Power reading is 0.000"));

assertTrue(response.contains("action=on"));

assertTrue(response.contains("action=off"));

assertTrue(response.contains("action=toggle"));

}

@Test

public void testSwitchOnAction() {

Map<String, String> params = new HashMap<>();

params.put("action", "on");

String response = httpCommands.handleGet("/plug1", params);

assertTrue(response.contains("plug1 is on"));

assertTrue(plug1.isOn());

String checkResponse = httpCommands.handleGet("/plug1", new HashMap<>());

assertTrue(checkResponse.contains("plug1 is on"));

}

@Test

public void testSwitchOffAction() {

plug1.switchOn();

Map<String, String> params = new HashMap<>();

params.put("action", "off");

String response = httpCommands.handleGet("/plug1", params);

assertTrue(response.contains("plug1 is off"));

assertFalse(plug1.isOn());

String checkResponse = httpCommands.handleGet("/plug1", new HashMap<>());

assertTrue(checkResponse.contains("plug1 is off"));

}

@Test

public void testToggleActionOffToOn() {

plug1.switchOff();

Map<String, String> params = new HashMap<>();

params.put("action", "toggle");

String response = httpCommands.handleGet("/plug1", params);

assertTrue(response.contains("plug1 is on"));

assertTrue(plug1.isOn());

}

@Test

public void testToggleActionOnToOff() {

plug1.switchOn();

Map<String, String> params = new HashMap<>();

params.put("action", "toggle");

String response = httpCommands.handleGet("/plug1", params);

assertTrue(response.contains("plug1 is off"));

assertFalse(plug1.isOn());

}

@Test

public void testPowerReadingUpdate() {

plugWithSpecialChar.switchOn();

plugWithSpecialChar.updatePower(789.0);

String response = httpCommands.handleGet("/zzzz.789", new HashMap<>());

assertTrue(response.contains("Power reading is 789.000"));

}

@Test

public void testMultiplePlugsIndependence() {

plug1.switchOn();

plug2.switchOff();

Map<String, String> params = new HashMap<>();

params.put("action", "toggle");

httpCommands.handleGet("/plug1", params);

assertFalse(plug1.isOn());

assertFalse(plug2.isOn());

String plug2Response = httpCommands.handleGet("/plug2", new HashMap<>());

assertTrue(plug2Response.contains("plug2 is off"));

}

@Test

public void testSpecialCharactersInPlugNames() {

String initialResponse = httpCommands.handleGet("/zzzz.789", new HashMap<>());

assertTrue(initialResponse.contains("zzzz.789"));

Map<String, String> params = new HashMap<>();

params.put("action", "on");

String updatedResponse = httpCommands.handleGet("/zzzz.789", params);

assertTrue(updatedResponse.contains("zzzz.789 is on"));

assertTrue(plugWithSpecialChar.isOn());

}

@Test

public void testInvalidActionParameter() {

Map<String, String> params = new HashMap<>();

params.put("action", "invalid");

String response = httpCommands.handleGet("/plug1", params);

assertTrue(response.contains("plug1 is off"));

assertFalse(plug1.isOn());

}

@Test

public void testConcurrentActions() {

Map<String, String> onParams = new HashMap<>();

onParams.put("action", "on");

httpCommands.handleGet("/plug1", onParams);

assertTrue(plug1.isOn());

Map<String, String> offParams = new HashMap<>();

offParams.put("action", "off");

httpCommands.handleGet("/plug1", offParams);

assertFalse(plug1.isOn());

Map<String, String> onAgainParams = new HashMap<>();

onAgainParams.put("action", "on");

String finalResponse = httpCommands.handleGet("/plug1", onAgainParams);

assertTrue(finalResponse.contains("plug1 is on"));

assertTrue(plug1.isOn());

}

@Test

public void testListPlugs() {

String response = httpCommands.handleGet("/", new HashMap<>());

assertTrue(response.contains("href='/plug1'"));

assertTrue(response.contains("href='/plug2'"));

assertTrue(response.contains("href='/zzzz.789'"));

}

@Test

public void testNonExistentPlug() {

String response = httpCommands.handleGet("/nonexistent", new HashMap<>());

assertNull(response);

}

}

## MqttTests.java

package ece448.iot\_sim;

import static org.junit.Assert.\*;

import org.junit.Test;

import org.eclipse.paho.client.mqttv3.MqttMessage;

import org.eclipse.paho.client.mqttv3.MqttClient;

import org.eclipse.paho.client.mqttv3.MqttException;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class MqttTests {

// PlugSim Tests

@Test

public void testSwitchOn() {

PlugSim plug = new PlugSim("testPlug");

plug.switchOn();

assertTrue(plug.isOn());

}

@Test

public void testSwitchOff() {

PlugSim plug = new PlugSim("testPlug");

plug.switchOn();

plug.switchOff();

assertFalse(plug.isOn());

}

@Test

public void testToggle() {

PlugSim plug = new PlugSim("testPlug");

plug.toggle();

assertTrue(plug.isOn());

plug.toggle();

assertFalse(plug.isOn());

}

@Test

public void testMeasurePower() {

PlugSim plug = new PlugSim("testPlug");

plug.switchOn();

plug.measurePower();

assertNotEquals(0.0, plug.getPower(), 0.001);

}

@Test

public void testMeasurePowerWithDotInName() {

PlugSim plug = new PlugSim("test.123");

plug.switchOn();

plug.measurePower();

assertEquals(123.0, plug.getPower(), 0.001);

}

private static class TestObserver implements PlugSim.Observer {

private String lastName;

private String lastKey;

private String lastValue;

@Override

public void update(String name, String key, String value) {

this.lastName = name;

this.lastKey = key;

this.lastValue = value;

}

public boolean receivedStateUpdate(String state) {

return "state".equals(lastKey) && state.equals(lastValue);

}

public boolean receivedPowerUpdate() {

return "power".equals(lastKey);

}

}

@Test

public void testObserverNotificationOnSwitchOn() {

PlugSim plug = new PlugSim("testPlug");

TestObserver observer = new TestObserver();

plug.addObserver(observer);

plug.switchOn();

assertTrue(observer.receivedStateUpdate("on"));

}

@Test

public void testObserverNotificationOnPowerChange() {

PlugSim plug = new PlugSim("testPlug");

TestObserver observer = new TestObserver();

plug.addObserver(observer);

plug.switchOn();

plug.measurePower();

assertTrue(observer.receivedPowerUpdate());

}

// MqttCommands Tests

@Test

public void testHandleMessageOn() throws Exception {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");

PlugSim plug = new PlugSim("testPlug");

mqttCmd.addPlug(plug);

String topic = "testPrefix/action/testPlug/on";

MqttMessage msg = new MqttMessage("".getBytes());

mqttCmd.handleMessage(topic, msg);

assertTrue(plug.isOn());

}

@Test

public void testHandleMessageOff() throws Exception {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");

PlugSim plug = new PlugSim("testPlug");

plug.switchOn();

mqttCmd.addPlug(plug);

String topic = "testPrefix/action/testPlug/off";

MqttMessage msg = new MqttMessage("".getBytes());

mqttCmd.handleMessage(topic, msg);

assertFalse(plug.isOn());

}

@Test

public void testHandleMessageToggle() throws Exception {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");

PlugSim plug = new PlugSim("testPlug");

mqttCmd.addPlug(plug);

String topic = "testPrefix/action/testPlug/toggle";

MqttMessage msg = new MqttMessage("".getBytes());

mqttCmd.handleMessage(topic, msg);

assertTrue(plug.isOn());

}

@Test

public void testHandleMessageInvalidTopic() {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");

String topic = "invalidTopic";

MqttMessage msg = new MqttMessage();

mqttCmd.handleMessage(topic, msg);

// Ensure no exceptions and plug remains unchanged

}

@Test

public void testHandleMessageUnknownAction() {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");

PlugSim plug = new PlugSim("testPlug");

mqttCmd.addPlug(plug);

String topic = "testPrefix/action/testPlug/invalid";

MqttMessage msg = new MqttMessage();

mqttCmd.handleMessage(topic, msg);

assertFalse(plug.isOn()); // No change as action is unknown

}

@Test

public void testHandleMessageNonExistentPlug() {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");

String topic = "testPrefix/action/nonExistentPlug/on";

MqttMessage msg = new MqttMessage();

mqttCmd.handleMessage(topic, msg);

// No plug exists, ensure no exceptions

}

// MqttUpdates Tests

@Test

public void testGetTopic() throws Exception {

MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");

MqttUpdates mqttUpd = new MqttUpdates("testPrefix", mqttClient);

String name = "testPlug";

String key = "state";

String expectedTopic = "testPrefix/update/testPlug/state";

assertEquals(expectedTopic, mqttUpd.getTopic(name, key));

}

@Test

public void testGetTopicWithMultiLevelPrefix() throws Exception {

MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");

MqttUpdates mqttUpd = new MqttUpdates("a/b/c", mqttClient);

String topic = mqttUpd.getTopic("plugName", "state");

assertEquals("a/b/c/update/plugName/state", topic);

}

@Test

public void testGetMessage() throws Exception {

MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");

MqttUpdates mqttUpd = new MqttUpdates("testPrefix", mqttClient);

String value = "on";

MqttMessage msg = mqttUpd.getMessage(value);

assertEquals("on", new String(msg.getPayload()));

assertTrue(msg.isRetained());

}

@Test

public void testPowerRandomWalk() {

PlugSim plug = new PlugSim("testRandom");

plug.switchOn();

for (int i = 0; i < 10; i++) {

plug.measurePower();

}

assertTrue(plug.getPower() >= 0);

}

@Test

public void testMessageRetentionFlag() throws Exception {

try (MqttClient client = new MqttClient("tcp://localhost:1883", "testClient")) {

MqttUpdates mqttUpd = new MqttUpdates("prefix", client);

MqttMessage msg = mqttUpd.getMessage("on");

assertTrue(msg.isRetained());

}

}

@Test

public void testMultiLevelTopicPrefix() throws Exception {

try (MqttClient client = new MqttClient("tcp://localhost:1883", "testClient")) {

MqttUpdates mqttUpd = new MqttUpdates("a/b/c", client);

String topic = mqttUpd.getTopic("plug", "state");

assertEquals("a/b/c/update/plug/state", topic);

}

}

@Test

public void testPowerCalculationWithDottedName() {

PlugSim plug = new PlugSim("test.250");

plug.switchOn();

plug.measurePower();

assertEquals(250.0, plug.getPower(), 0.001);

}

@Test

public void testConcurrentToggle() throws InterruptedException {

PlugSim plug = new PlugSim("concurrentPlug");

int numThreads = 10;

ExecutorService executor = Executors.newFixedThreadPool(numThreads);

assertFalse(plug.isOn());

for (int i = 0; i < numThreads; i++) {

executor.submit(plug::toggle);

}

executor.shutdown();

executor.awaitTermination(1, TimeUnit.SECONDS);

assertFalse(plug.isOn());

}

@Test

public void testMqttCommandsConstructor() {

List<PlugSim> plugList = new ArrayList<>();

plugList.add(new PlugSim("plug1"));

plugList.add(new PlugSim("plug2"));

MqttCommands mqttCmd = new MqttCommands(plugList, "testPrefix");

assertEquals(2, mqttCmd.plugs.size());

assertTrue(mqttCmd.plugs.containsKey("plug1"));

assertTrue(mqttCmd.plugs.containsKey("plug2"));

}

@Test

public void testGetTopic1() {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");

String expectedTopic = "testPrefix/action/#";

assertEquals(expectedTopic, mqttCmd.getTopic());

}

@Test

public void testHandleMessageExceptionHandling() {

MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");

String invalidTopic = null;

MqttMessage msg = new MqttMessage();

try {

mqttCmd.handleMessage(invalidTopic, msg);

} catch (Exception e) {

fail("Exception should have been handled gracefully.");

}

}

@Test

public void testPublishUpdateSuccess() throws Exception {

MqttClient client = new MqttClient("tcp://localhost:1883", "testClient");

client.connect();

MqttUpdates mqttUpd = new MqttUpdates("testPrefix", client);

String name = "testPlug";

String key = "state";

String value = "on";

mqttUpd.publishUpdate(name, key, value);

client.subscribe("testPrefix/update/testPlug/state", (topic, message) -> {

assertEquals("on", new String(message.getPayload()));

assertTrue(message.isRetained());

client.disconnect();

});

}

@Test

public void testPublishUpdateExceptionHandling() throws Exception {

MqttClient client = new MqttClient("tcp://localhost:1883", "testClient");

client.connect();

client.disconnect();

MqttUpdates mqttUpd = new MqttUpdates("testPrefix", client);

try {

mqttUpd.publishUpdate("testPlug", "state", "on");

} catch (Exception e) {

fail("Exception should have been handled gracefully.");

}

}

}

## PlugSimTests.java

package ece448.iot\_sim;

import static org.junit.Assert.\*;

import org.junit.Test;

public class PlugSimTests {

@Test

public void testInit() {

PlugSim plug = new PlugSim("a");

assertFalse(plug.isOn());

}

@Test

public void testSwitchOn() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

assertTrue(plug.isOn());

}

@Test

public void testGetName() {

PlugSim plug = new PlugSim("test.100");

assertEquals("test.100", plug.getName());

}

@Test

public void testSwitchOffFromOn() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

assertFalse(plug.isOn());

}

@Test

public void testMultipleSwitching() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

plug.switchOn();

assertTrue(plug.isOn());

}

@Test

public void testToggleFromOn() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.toggle();

assertFalse(plug.isOn());

}

@Test

public void testToggleFromOff() {

PlugSim plug = new PlugSim("a");

plug.toggle();

assertTrue(plug.isOn());

}

@Test

public void testPowerMeasurementWhenOn() {

PlugSim plug = new PlugSim("test.500");

plug.switchOn();

plug.measurePower();

assertEquals(500.0, plug.getPower(), 0.001);

}

@Test

public void testPowerMeasurementWhenOff() {

PlugSim plug = new PlugSim("test.500");

plug.measurePower();

assertEquals(0.0, plug.getPower(), 0.001);

}

@Test

public void testMultipleToggleAndPower() {

PlugSim plug = new PlugSim("test.300");

plug.toggle();

plug.measurePower();

assertEquals(300.0, plug.getPower(), 0.001);

plug.toggle();

plug.measurePower();

assertEquals(0.0, plug.getPower(), 0.001);

plug.toggle();

plug.measurePower();

assertEquals(300.0, plug.getPower(), 0.001);

}

@Test

public void testRandomWalkLowPower() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.updatePower(50);

double initialPower = plug.getPower();

plug.measurePower();

double newPower = plug.getPower();

assertTrue(newPower > initialPower);

assertTrue(newPower <= initialPower + 100);

}

@Test

public void testRandomWalkHighPower() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.updatePower(350);

double initialPower = plug.getPower();

plug.measurePower();

double newPower = plug.getPower();

assertTrue(newPower < initialPower);

assertTrue(newPower >= initialPower - 100);

}

@Test

public void testRandomWalkMediumPower() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.updatePower(200);

double initialPower = plug.getPower();

plug.measurePower();

double newPower = plug.getPower();

assertTrue(Math.abs(newPower - initialPower) <= 20);

}

}

# ~/iot\_ece448/src/main/java/ece448/grading

## GradeP1.java

package ece448.grading;

import ece448.iot\_sim.PlugSim;

public class GradeP1 {

public static void main(String[] args) {

Grading.run(new GradeP1(), 10);

}

public boolean testCase00() {

PlugSim plug = new PlugSim("a");

return plug.getName().equals("a");

}

public boolean testCase01() {

PlugSim plug = new PlugSim("a");

return !plug.isOn();

}

public boolean testCase02() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

return plug.isOn();

}

public boolean testCase03() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

return !plug.isOn();

}

public boolean testCase04() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

plug.switchOn();

return plug.isOn();

}

public boolean testCase05() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

plug.switchOn();

plug.toggle();

return !plug.isOn();

}

public boolean testCase06() {

PlugSim plug = new PlugSim("a");

plug.switchOn();

plug.switchOff();

plug.switchOn();

plug.toggle();

plug.toggle();

return plug.isOn();

}

public boolean testCase07() {

PlugSim plug = new PlugSim("b.200");

plug.switchOn();

plug.measurePower();

return plug.getPower() == 200;

}

public boolean testCase08() {

PlugSim plug = new PlugSim("b.200");

plug.switchOn();

plug.measurePower();

plug.switchOff();

plug.measurePower();

return plug.getPower() == 0;

}

public boolean testCase09() {

PlugSim plug = new PlugSim("cccccccc.1000");

plug.switchOn();

plug.measurePower();

plug.switchOff();

plug.measurePower();

return plug.getName().equals("cccccccc.1000")

&& !plug.isOn();

}

}

## GradeP2.java

package ece448.grading;

import java.util.Arrays;

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

import ece448.iot\_sim.SimConfig;

import ece448.iot\_sim.Main;

public class GradeP2 {

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

SimConfig config = new SimConfig(

8080, Arrays.asList("xxxx", "yyyy", "zzzz.789"), null, null, null);

try (Main m = new Main(config))

{

Grading.run(new GradeP2(), 10);

}

}

private String get(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();

}

public boolean testCase00() throws Exception {

String ret = get("/xxxx");

return (ret.indexOf("xxxx is off") != -1)

&& (ret.indexOf("xxxx is on") == -1)

&& (ret.indexOf("Power reading is 0.000") != -1);

}

public boolean testCase01() throws Exception {

String ret = get("/xxxx?action=on");

return (ret.indexOf("xxxx is on") != -1)

&& (ret.indexOf("xxxx is off") == -1);

}

public boolean testCase02() throws Exception {

String ret = get("/xxxx?action=off");

return (ret.indexOf("xxxx is off") != -1)

&& (ret.indexOf("xxxx is on") == -1);

}

public boolean testCase03() throws Exception {

String ret = get("/xxxx?action=on");

return (ret.indexOf("xxxx is on") != -1)

&& (ret.indexOf("xxxx is off") == -1);

}

public boolean testCase04() throws Exception {

String ret = get("/xxxx?action=toggle");

return (ret.indexOf("xxxx is off") != -1)

&& (ret.indexOf("xxxx is on") == -1);

}

public boolean testCase05() throws Exception {

String ret = get("/xxxx?action=toggle");

return (ret.indexOf("xxxx is on") != -1)

&& (ret.indexOf("xxxx is off") == -1);

}

public boolean testCase06() throws Exception {

String ret = get("/yyyy");

return (ret.indexOf("yyyy is off") != -1)

&& (ret.indexOf("yyyy is on") == -1);

}

public boolean testCase07() throws Exception {

String ret = get("/xxxx");

return (ret.indexOf("xxxx is on") != -1)

&& (ret.indexOf("xxxx is off") == -1);

}

public boolean testCase08() throws Exception {

String ret = get("/zzzz.789");

return (ret.indexOf("Power reading is 0.000") != -1);

}

public boolean testCase09() throws Exception {

get("/zzzz.789?action=on");

Thread.sleep(1500);

String ret = get("/zzzz.789");

return (ret.indexOf("Power reading is 789.000") != -1);

}

}

## GradeP3.java

package ece448.grading;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

import java.util.UUID;

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

import org.eclipse.paho.client.mqttv3.MqttClient;

import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

import org.eclipse.paho.client.mqttv3.MqttMessage;

import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import ece448.iot\_sim.SimConfig;

import ece448.iot\_sim.Main;

public class GradeP3 implements AutoCloseable {

public static class MqttController {

private final String broker;

private final String clientId;

private final String topicPrefix;

private final MqttClient client;

private final HashMap<String, String> states = new HashMap<>();

private final HashMap<String, String> powers = new HashMap<>();

public MqttController(String broker, String clientId,

String topicPrefix) throws Exception {

this.broker = broker;

this.clientId = clientId;

this.topicPrefix = topicPrefix;

this.client = new MqttClient(broker, clientId, new MemoryPersistence());

}

public void start() throws Exception {

MqttConnectOptions opt = new MqttConnectOptions();

opt.setCleanSession(true);

client.connect(opt);

client.subscribe(topicPrefix+"/update/#", this::handleUpdate);

logger.info("MqttCtl {}: {} connected", clientId, broker);

}

public void close() throws Exception {

client.disconnect();

logger.info("MqttCtl {}: disconnected", clientId);

}

synchronized public void publishAction(String plugName, String action) {

String topic = topicPrefix+"/action/"+plugName+"/"+action;

try

{

client.publish(topic, new MqttMessage());

}

catch (Exception e)

{

logger.error("MqttCtl {}: {} fail to publish", clientId, topic);

}

}

synchronized public String getState(String plugName) {

return states.get(plugName);

}

synchronized public String getPower(String plugName) {

return powers.get(plugName);

}

synchronized public Map<String, String> getStates() {

return new TreeMap<>(states);

}

synchronized public Map<String, String> getPowers() {

return new TreeMap<>(powers);

}

synchronized protected void handleUpdate(String topic, MqttMessage msg) {

logger.debug("MqttCtl {}: {} {}", clientId, topic, msg);

String[] nameUpdate = topic.substring(topicPrefix.length()+1).split("/");

if ((nameUpdate.length != 3) || !nameUpdate[0].equals("update"))

return; // ignore unknown format

switch (nameUpdate[2])

{

case "state":

states.put(nameUpdate[1], msg.toString());

break;

case "power":

powers.put(nameUpdate[1], msg.toString());

break;

default:

return;

}

}

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

}

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

private static final String topicPrefix = System.currentTimeMillis()+"/grade\_p3/iot\_ece448";

private final MqttController mqtt;

private GradeP3() throws Exception {

this.mqtt = new MqttController(broker, "grader/iot\_sim", 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, Arrays.asList("xx", "yy", "zz.666"),

broker, "testee/iot\_sim", topicPrefix);

try (GradeP3 p3 = new GradeP3(); Main m = new Main(config))

{

Grading.run(p3, 10);

}

}

private String get(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();

}

public boolean testCase00() throws Exception {

return "off".equals(mqtt.getState("xx"));

}

public boolean testCase01() throws Exception {

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

Thread.sleep(1000);

return "on".equals(mqtt.getState("xx"));

}

public boolean testCase02() throws Exception {

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

Thread.sleep(1000);

return "off".equals(mqtt.getState("xx"));

}

public boolean testCase03() throws Exception {

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

Thread.sleep(1000);

return "on".equals(mqtt.getState("xx"));

}

public boolean testCase04() throws Exception {

Thread.sleep(1500);

if (!"0.000".equals(mqtt.getPower("zz.666")))

return false;

mqtt.publishAction("zz.666", "on");

Thread.sleep(1500);

return "666.000".equals(mqtt.getPower("zz.666"));

}

public boolean testCase05() throws Exception {

return (mqtt.getPower("yyyy") == null)

&& (mqtt.getState("yyyy") == null)

&& "on".equals(mqtt.getState("zz.666"));

}

public boolean testCase06() throws Exception {

get("/yy?action=on");

Thread.sleep(1000);

return "on".equals(mqtt.getState("yy"));

}

public boolean testCase07() throws Exception {

get("/yy?action=off");

Thread.sleep(1000);

return "off".equals(mqtt.getState("yy"));

}

public boolean testCase08() throws Exception {

mqtt.publishAction("zz.666", "toggle");

String ret = get("/zz.666");

Thread.sleep(1000);

return (ret.indexOf("zz.666 is off") != -1)

&& (ret.indexOf("zz.666 is on") == -1)

&& "off".equals(mqtt.getState("zz.666"));

}

public boolean testCase09() throws Exception {

mqtt.publishAction("zz.666", "toggle");

String ret = get("/zz.666");

Thread.sleep(1000);

return (ret.indexOf("zz.666 is on") != -1)

&& (ret.indexOf("zz.666 is off") == -1)

&& "on".equals(mqtt.getState("zz.666"));

}

}

## 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);

}

}

## GradeP5.java

package ece448.grading;

import java.util.Arrays;

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.apache.http.entity.ContentType;

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 GradeP5 implements AutoCloseable {

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

private static final String topicPrefix = System.currentTimeMillis()+"/grade\_p5/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> groupNames = Arrays.asList("x", "y", "z");

private static final ObjectMapper mapper = new ObjectMapper();

private static final Logger logger = LoggerFactory.getLogger(GradeP5.class);

private final MqttController mqtt;

private GradeP5() 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 (

GradeP5 p5 = new GradeP5();

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(p5, 10);

}

}

static void postGroup(String group, List<String> members) throws Exception {

Request.Post("http://127.0.0.1:8088/api/groups/" + group)

.bodyByteArray(mapper.writeValueAsBytes(members), ContentType.APPLICATION\_JSON)

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

.socketTimeout(1000).execute();

}

static void delGroup(String group) throws Exception {

Request.Delete("http://127.0.0.1:8088/api/groups/" + group)

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

.socketTimeout(1000).execute();

}

static String getGroups1() throws Exception {

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

for (String name: groupNames)

{

Map<String, Object> group = mapper.readValue(GradeP4.getHub("/api/groups/"+name),

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

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

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

StringBuilder field = new StringBuilder(name+".");

@SuppressWarnings("unchecked")

List<Map<String, Object>> members = (List<Map<String, Object>>)group.get("members");

for (Map<String, Object> member: members)

{

field.append(member.get("name"));

field.append("off".equals(member.get("state"))? "0": "1");

}

if (!members.isEmpty())

fields.put(name, field.toString());

}

String ret = String.join("|", fields.values());

logger.debug("GradeP5: getGroups1 {}", ret);

return ret;

}

static String getGroups2() throws Exception {

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

List<Map<String, Object>> groups = mapper.readValue(GradeP4.getHub("/api/groups"),

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

for (Map<String, Object> group: groups)

{

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

StringBuilder field = new StringBuilder(name+".");

@SuppressWarnings("unchecked")

List<Map<String, Object>> members = (List<Map<String, Object>>)group.get("members");

for (Map<String, Object> member: members)

{

field.append(member.get("name"));

field.append("off".equals(member.get("state"))? "0": "1");

}

fields.put(name, field.toString());

}

String ret = String.join("|", fields.values());

logger.debug("GradeP5: getGroups2 {}", ret);

return ret;

}

static boolean verifyGroups(String groups) throws Exception {

return groups.equals(getGroups1())

&& groups.equals(getGroups2());

}

public boolean testCase00() throws Exception {

return verifyGroups("");

}

public boolean testCase01() throws Exception {

GradeP4.getHub("/api/plugs/a?action=off");

GradeP4.getHub("/api/plugs/b?action=on");

GradeP4.getHub("/api/plugs/c?action=off");

GradeP4.getHub("/api/plugs/d?action=toggle");

GradeP4.getHub("/api/plugs/e?action=on");

GradeP4.getHub("/api/plugs/f?action=off");

GradeP4.getHub("/api/plugs/g?action=off");

Thread.sleep(1000);

return GradeP4.verifyStates("0101100", mqtt) && verifyGroups("");

}

public boolean testCase02() throws Exception {

postGroup("z", Arrays.asList("a", "d"));

Thread.sleep(1000);

return GradeP4.verifyStates("0101100", mqtt)

&& verifyGroups("z.a0d1");

}

public boolean testCase03() throws Exception {

postGroup("y", Arrays.asList("b", "d", "f"));

Thread.sleep(1000);

return GradeP4.verifyStates("0101100", mqtt)

&& verifyGroups("y.b1d1f0|z.a0d1");

}

public boolean testCase04() throws Exception {

postGroup("x", Arrays.asList("a", "c", "e", "g"));

Thread.sleep(1000);

return GradeP4.verifyStates("0101100", mqtt)

&& verifyGroups("x.a0c0e1g0|y.b1d1f0|z.a0d1");

}

public boolean testCase05() throws Exception {

GradeP4.getHub("/api/groups/x?action=on");

GradeP4.getHub("/api/groups/y?action=off");

Thread.sleep(1000);

return GradeP4.verifyStates("1010101", mqtt)

&& verifyGroups("x.a1c1e1g1|y.b0d0f0|z.a1d0");

}

public boolean testCase06() throws Exception {

GradeP4.getHub("/api/groups/z?action=toggle");

Thread.sleep(1000);

return GradeP4.verifyStates("0011101", mqtt)

&& verifyGroups("x.a0c1e1g1|y.b0d1f0|z.a0d1");

}

public boolean testCase07() throws Exception {

GradeP4.getSim("/c?action=off");

GradeP4.getSimEx("/d?action=off");

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

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

Thread.sleep(1000);

return GradeP4.verifyStates("0000000", mqtt)

&& verifyGroups("x.a0c0e0g0|y.b0d0f0|z.a0d0");

}

public boolean testCase08() throws Exception {

delGroup("z");

Thread.sleep(1000);

return GradeP4.verifyStates("0000000", mqtt)

&& verifyGroups("x.a0c0e0g0|y.b0d0f0");

}

public boolean testCase09() throws Exception {

postGroup("x", Arrays.asList("a", "b", "c"));

postGroup("y", Arrays.asList("e", "f", "g"));

Thread.sleep(500);

GradeP4.getHub("/api/groups/x?action=toggle");

GradeP4.getHub("/api/groups/y?action=toggle");

GradeP4.getHub("/api/groups/x?action=toggle");

Thread.sleep(1000);

return GradeP4.verifyStates("0000111", mqtt)

&& verifyGroups("x.a0b0c0|y.e1f1g1");

}

}