Project 4

ECE 528: Application Software Design

Alan Palayil

Professor: Won-Jae Yi

Acknowledgement: I acknowledge all works including figures, codes and writings belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behavior, report writings and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Electronic Signature: Alan Palayil A20447935 (Due Date: 3/26/2023)

Table of Contents

Abstract:

This project involves building a server backend for an IoT hub using Spring Boot application. The RESTful services provided by the backend will be used by the frontend web application, which will be built in a subsequent project. The server backend will use MQTT broker to communicate with one or more IoT simulators for controlling smart plugs and obtaining their state updates. The project also includes user stories for reference, but the class design and implementation are left to the discretion of the developer. The red-green cycle will be used to add unit tests and implement classes, with MQTT and HTTP communications used for testing. The code snippets from GradeP3.java and GradeP4.java may be used for convenience.

Introduction:

In this project, we aim to develop a server backend for our IoT hub using Spring Boot application. The server backend will offer RESTful services to a frontend web application and use MQTT broker to communicate with IoT simulators for controlling smart plugs and obtaining their state updates. With user stories to guide us, we will use the red-green cycle to add unit tests and implement our classes. By the end of the project, we hope to have a well-functioning backend server that can efficiently handle IoT devices and user requests.

Background:

This project involves building a Spring Boot backend for an IoT hub that communicates with IoT simulators via MQTT broker. The backend will provide RESTful services for controlling smart plugs and obtaining their state updates, with user stories provided as a guide for development. The project comprises several users stories:

- State of a Single Plug: The end-user desires to retrieve the state of a specific plug by sending a GET request to /api/plugs/plugName, receiving a JSON object containing the name, state (on/off), and power level of the plug.

- States of All Plugs: The end-user desires to retrieve the states of all plugs at once through a GET request to /api/plugs in a web application. The response must consist of a JSON array of objects, with each object representing the state of a single plug.
- Control a Single Plug: The end-user desires to control the plug "plugName" through a GET request to /api/plugs/plugName with a query string in a web application, enabling them to toggle it (action=toggle), switch it on (action=on), or switch it off (action=off).

The topics and messages for each of these user stories are defined based on the plug name and configuration string. The project also includes testing procedures implemented in ece448.grading.GradeP4 to ensure that all user stories are covered.

Results and Discussion:

The project required me to create the files MqttController and PlugsResource. During the implementation of the project, I had to modify the following classes: iot_hub>Main.java, iot_sim>Main.java, and iot_sim>MeasurePower.java. The MqttController class has the following methods:

• start: starts the MQTT client for iot_hub

• publishAction: takes a plug name and action, reformats it for MQTT, and sends the MQTT messages to the IoT simulator

• getPlugs: returns a list of all plugs locally stored in the hub

• getState: returns the state corresponding to a plug locally stored in the hub

• getPower: returns the power corresponding to a plug locally stored in the hub

• getStates: returns a map of the states of all plugs

• getPowers: returns a map of the powers of all plugs

• handleUpdate: updates local copy of plugs on the hub using MQTT messages received from the IoT simulator.

The PlugsResource class has the following methods:

• getPlugs: The MqttController locally stores the plugs information obtained from the IoT simulator, which is then retrieved by sending a JSON request to "/api/plugs".

• getPlug: Sending a JSON request to "/api/plugs/" will return the stored state and power information of the plug from the local MqttController. If an action (on/off/toggle) is specified, the MqttController's publishAction method will send that action to the IoT simulator.

• obtainPlug: The helper method "getPlugs" and "getPlug" retrieves plug information (state and power) for a specific plug name from the MqttController.

The following are the screenshots of the results I got during the project.

Screenshots of the Tests results:

The screenshot of the acceptance testing ‘Gradle’ results is added which was the base testing criteria for project 4 with 10 Local Testing passed.

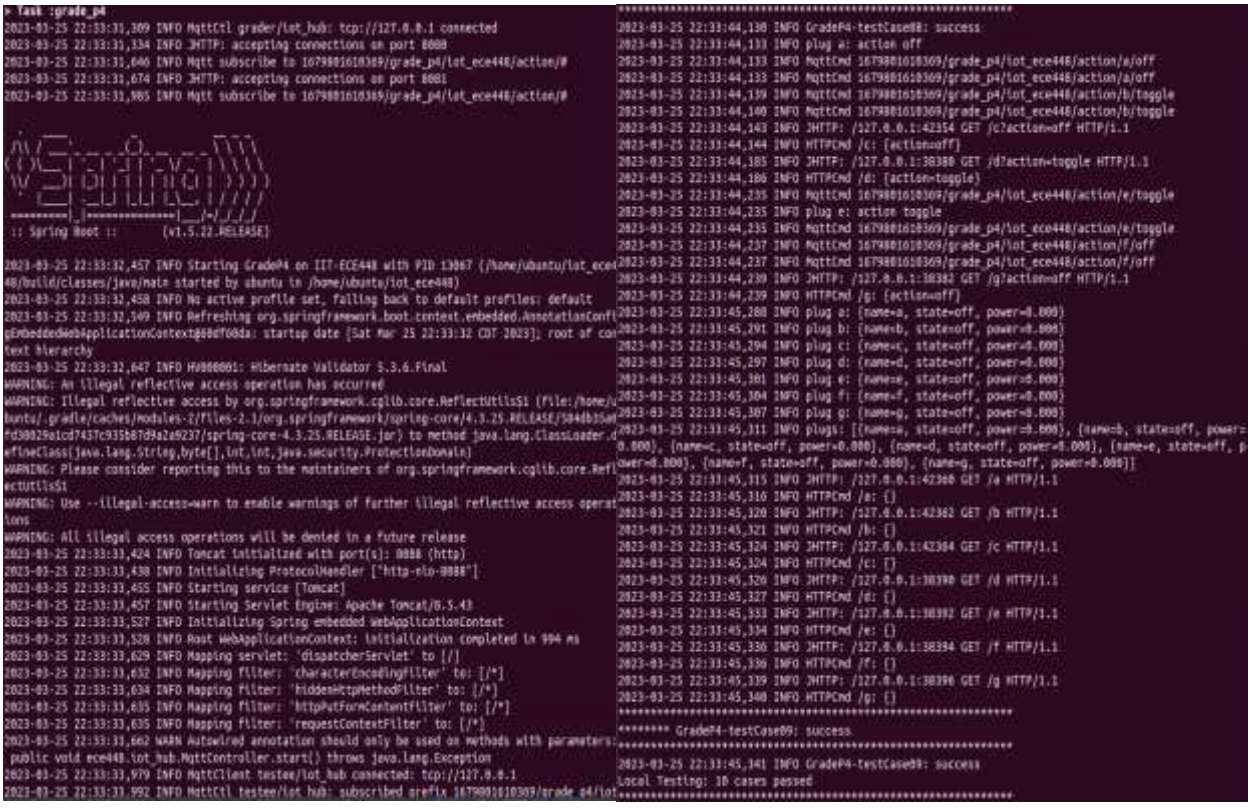

Figure 1: GradleP4- Test cases Results

Screenshots of the HTML results:

To completely utilize the MqttController and PlugsResource, I referred to grade_p4 and checked each of the elements within the IoT_Sim program to work over the unit testing. The test cases were designed to test the utilization of the program.

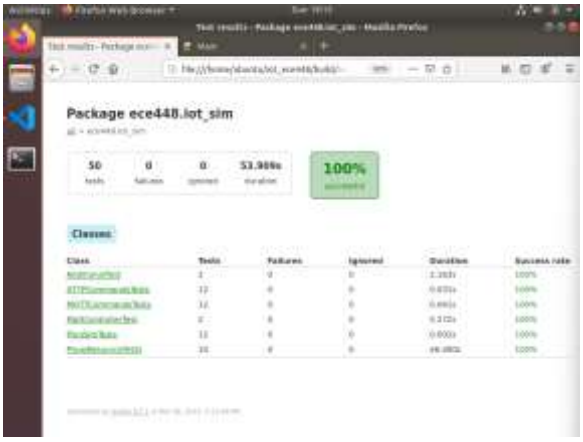

Figure 2: IoT-Sim Test Cases

Figure 3: Utilization of ece448.iot_sim

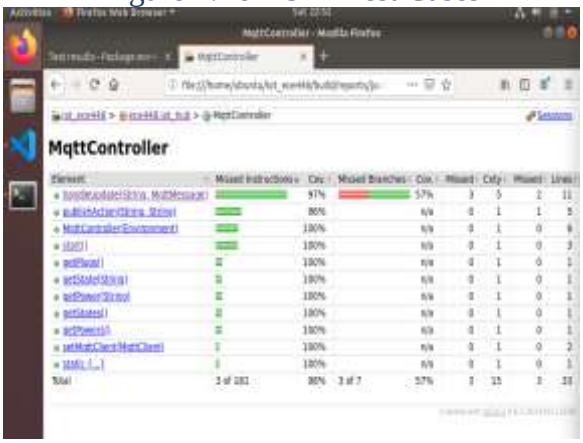

Figure 4: Utilization of Elements in MqttController

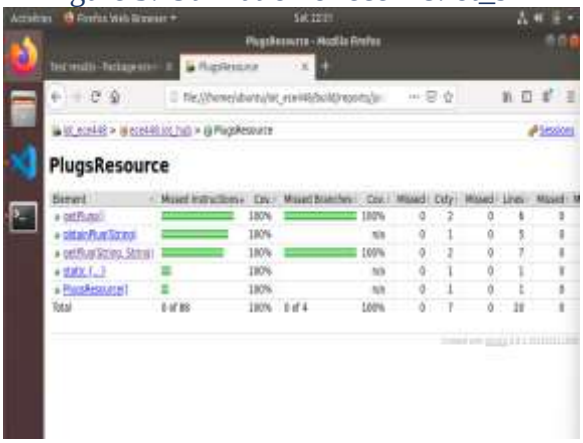

Figure 5: Utilization of Elements in PlugsResource

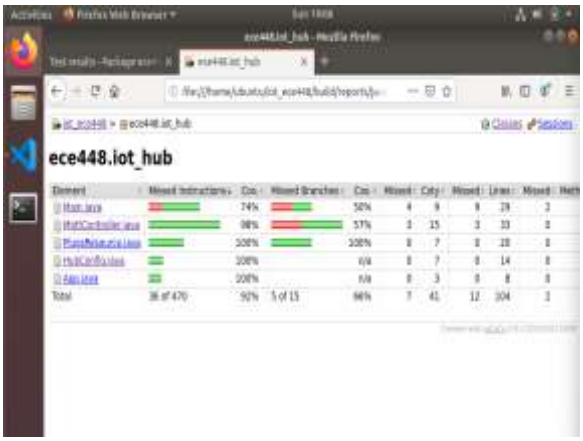

Figure 6: Utilization of ece448.iot_hub

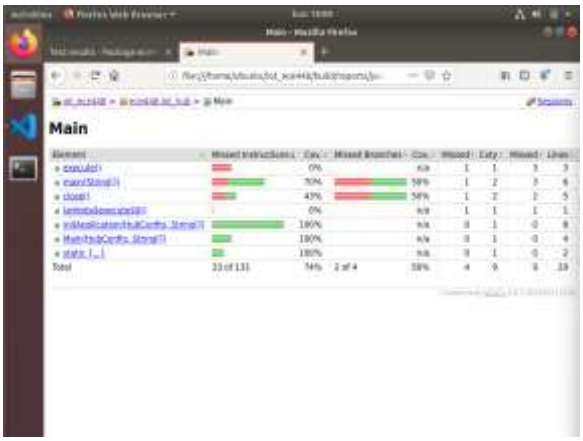

Figure 7: Utilization of Elements in Main.java

Conclusion:

In conclusion, this project focuses on building a server backend for an IoT hub that can control smart plugs and obtain their state updates using an MQTT broker. The backend will provide RESTful services to a frontend web application, which will be developed in a subsequent project. The user stories provided will guide the implementation process, although the developer is free to choose their own class design and implementation. By following the red-green cycle, adding unit tests, and using MQTT and HTTP communications, the developer should be able to implement the classes required for the server backend. The resulting server should enable end-users to query the states of individual plugs, or all plugs at once, as well as control them by switching them on/off or toggling them.

Appendix:

Source Code of the edited programs within the project:

App.java

```
package ece448.iot_hub;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.env.Environment;

@SpringBootApplication
public class App {

    @Bean(destroyMethod = "disconnect")
    public MqttClient mqttClient(Environment env) throws Exception {
        String broker = env.getProperty("mqtt.broker");
        String clientId = env.getProperty("mqtt.clientId");
        MqttClient mqtt = new MqttClient(broker, clientId, new
MemoryPersistence());
        mqtt.connect();
        logger.info("MqttClient {} connected: {}", clientId, broker);
        return mqtt;
    }

    private static final Logger logger =
LoggerFactory.getLogger(App.class);
}
```

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;
    private long sleepIntervalMillis = 60000;

    @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;
    }
    public long getSleepIntervalMillis() {
        return sleepIntervalMillis;
    }

    public void setSleepIntervalMillis(long sleepIntervalMillis) {
        this.sleepIntervalMillis = sleepIntervalMillis;
    }
}
```

Main.java

```
package ece448.iot_hub;

import java.io.File;
import java.util.HashMap;

import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.concurrent.atomic.AtomicBoolean;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.context.ConfigurableApplicationContext;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class Main implements AutoCloseable {
    public Main(HubConfig config, String[] args) throws Exception {
        appCtx = initApplication(config, args);
        sleepIntervalMillis = config.getSleepIntervalMillis();
    }

    public ConfigurableApplicationContext initApplication(HubConfig
config, String[] args) {
        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);
        return app.run(args);
    }

    public void execute() {
        scheduledExecutor = Executors.newSingleThreadScheduledExecutor();
        scheduledExecutor.scheduleAtFixedRate(() -> {
            // Add any recurring task here, if needed.
        }, 0, sleepIntervalMillis, TimeUnit.MILLISECONDS);
    }
    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)) {
            // execute the main loop
            m.execute();
        }
    }
    @Override
    public void close() throws Exception {
```

```
        if (scheduledExecutor != null) {
            scheduledExecutor.shutdown();
            scheduledExecutor.awaitTermination(5, TimeUnit.SECONDS);
    }
    appCtx.close();
    }

    private final ConfigurableApplicationContext appCtx;
    private final long sleepIntervalMillis;
    private ScheduledExecutorService scheduledExecutor;

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

}
```

PlugsResource.java

```
package ece448.iot_hub;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
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;

@RestController
public class PlugsResource {

    @Autowired
    private MqttController mqtt;

    @GetMapping("/api/plugs")
    public Collection<Object> getPlugs() throws Exception {
        ArrayList<Object> ret = new ArrayList<>();
        for (String plug: mqtt.getPlugs()) {
            ret.add(obtainPlug(plug));
        }
```

```
        logger.info("plugs: {}", ret);
        return ret;
    }

    @GetMapping("/api/plugs/{plug:.+}")
    public Object getPlug(@PathVariable("plug") String plug,
            @RequestParam(value = "action", required = false) String
action) {
        if (action == null) {
            Object ret = obtainPlug(plug);
            logger.info("plug {}: {}", plug, ret);
            return ret;
        }

        mqtt.publishAction(plug, action);
        logger.info("plug {}: action {}", plug, action);

        return null;
    }

    public Object obtainPlug(String plug) {
        HashMap<String, Object> ret = new HashMap<>();

        ret.put("name", plug);
        ret.put("state", mqtt.getState(plug));
        ret.put("power", mqtt.getPower(plug));

        return ret;
    }

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

MqttController.java

```
package ece448.iot_hub;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

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

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.stereotype.Component;

@Component
public class MqttController {

    @Autowired
    protected MqttClient client;

    public void setMqttClient(MqttClient client) {
        this.client = client;
    }


    private final String clientId;
    private final String topicPrefix;

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

    public MqttController(Environment env) throws Exception {
        this.clientId = env.getProperty("mqtt.clientId");
        this.topicPrefix = env.getProperty("mqtt.topicPrefix");
    }

    @Autowired
    public void start() throws Exception {
        client.subscribe(topicPrefix+"/update/#", this::handleUpdate);
        logger.info("MqttCtl {}: subscribed prefix {}", clientId,
topicPrefix);
    }

    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 List<String> getPlugs() {
        return new ArrayList<>(states.keySet());
    }

    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);
    }

    public synchronized 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);
}
```

AditionalTest.java

```
package ece448.iot_sim;
import ece448.iot_hub.*;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import org.springframework.boot.test.context.SpringBootTest;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.mockito.Mockito.when;

import ece448.iot_hub.Main;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

import static org.junit.Assert.assertTrue;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

@RunWith(MockitoJUnitRunner.class)
@SpringBootTest
public class AdditionalTest {
```

```
    @Mock
    private MqttController mqttController;

    @InjectMocks
    private PlugsResource plugsResource;

    @Test
    public void testGetPlugsResource() throws Exception {
        String plug1 = "plug1";
        String plug2 = "plug2";
        String state1 = "on";
        String state2 = "off";
        String power1 = "100";
        String power2 = "200";
        Map<String, Object> expectedPlug1 = new HashMap<>();
        expectedPlug1.put("name", plug1);
        expectedPlug1.put("state", state1);
        expectedPlug1.put("power", power1);
        Map<String, Object> expectedPlug2 = new HashMap<>();
        expectedPlug2.put("name", plug2);
        expectedPlug2.put("state", state2);
        expectedPlug2.put("power", power2);
        when(mqttController.getPlugs()).thenReturn(Arrays.asList(plug1,
plug2));
        when(mqttController.getState(plug1)).thenReturn(state1);
        when(mqttController.getState(plug2)).thenReturn(state2);
        when(mqttController.getPower(plug1)).thenReturn(power1);
        when(mqttController.getPower(plug2)).thenReturn(power2);
        Collection<Object> result = plugsResource.getPlugs();
        assertEquals(2, result.size());
        assertTrue(result.contains(expectedPlug1));
        assertTrue(result.contains(expectedPlug2));
    }

    private final ByteArrayOutputStream outContent= new
ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;
    private final String[] testArgs = {"testConfig.json"};
    @Before
    public void setUp() {
        System.setOut(new PrintStream(outContent));
    }

    @After
    public void tearDown() {
```

```
        System.setOut(originalOut);
    }
    @Test
    public void testMain() throws Exception {
        String testConfigContent = "{\"httpPort\": 8080, \"mqttBroker\":
\"tcp://localhost:1883\", \"mqttClientId\": \"iot_hub_test\",
\"mqttTopicPrefix\": \"iot_hub_test\", \"sleepIntervalMillis\": 100}";
        Files.write(Paths.get("testConfig.json"),
testConfigContent.getBytes());
        ExecutorService executor = Executors.newSingleThreadExecutor();
        Future<?> future = executor.submit(() -> {
            try {
                Main.main(testArgs);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        Thread.sleep(1000);
        future.cancel(true);
        executor.shutdown();
        executor.awaitTermination(5, TimeUnit.SECONDS);
        String expectedOutput = "testConfig.json:
{\"httpPort\":8080,\"mqttBroker\":\"tcp://localhost:1883\",\"mqttClientId\
":\"iot_hub_test\",\"mqttTopicPrefix\":\"iot_hub_test\",\"sleepIntervalMil
lis\":100}";
        assertTrue(outContent.toString().contains(expectedOutput));
        Files.deleteIfExists(Paths.get("testConfig.json"));
    }
}
```

MqttControllerTest.java

```
package ece448.iot_sim;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;
import org.springframework.core.env.Environment;

import ece448.iot_hub.MqttController;

import java.util.List;
import java.util.Map;
```

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertTrue;
import static org.mockito.Mockito.*;
import org.mockito.ArgumentCaptor;
import org.mockito.Captor;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class MqttControllerTest {

    private MqttClient client;
    private Environment environment;
    private MqttController mqttController;

    @Captor
    private ArgumentCaptor<MqttMessage> mqttMessageCaptor;


    @Before
    public void setUp() throws Exception {
        client = Mockito.mock(MqttClient.class);
        environment = Mockito.mock(Environment.class);

when(environment.getProperty("mqtt.clientId")).thenReturn("testClientId");

when(environment.getProperty("mqtt.topicPrefix")).thenReturn("testPrefix")
;

        mqttController = new MqttController(environment);
        mqttController.setMqttClient(client);
        mqttMessageCaptor = ArgumentCaptor.forClass(MqttMessage.class);
    }

    @Test
    public void testMainMethod() throws IOException, InterruptedException
{
        String configContent = "{ \"httpPort\": 8080, \"mqttBroker\":
\"tcp://localhost:1883\", \"mqttClientId\": \"testClient\",
\"mqttTopicPrefix\": \"test/\", \"plugNames\": [\"plug1\"] }";
        Path tempConfigPath = Files.createTempFile("simConfig", ".json");
        Files.write(tempConfigPath, configContent.getBytes());
        Thread mainThread = new Thread(() -> {
            try {
```

```
                String[] args = {tempConfigPath.toString()};
                Main.main(args);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        mainThread.start();
        Thread.sleep(60000);
        mainThread.interrupt();
        mainThread.join();
        Files.deleteIfExists(tempConfigPath);
    }

    @Test
    public void testPublishAction() throws Exception {
        mqttController.publishAction("plug1", "on");
        verify(client).publish(eq("testPrefix/action/plug1/on"),
mqttMessageCaptor.capture());

        MqttMessage sentMessage = mqttMessageCaptor.getValue();
        assertArrayEquals(new byte[0], sentMessage.getPayload());
    }

    @Test
    public void testGetPlugs() {
        mqttController.handleUpdate("testPrefix/update/plug1/state", new
MqttMessage("on".getBytes()));
        mqttController.handleUpdate("testPrefix/update/plug2/state", new
MqttMessage("off".getBytes()));
        List<String> plugs = mqttController.getPlugs();
        assertEquals(2, plugs.size());
        assertTrue(plugs.contains("plug1"));
        assertTrue(plugs.contains("plug2"));
        Map<String, String> states = mqttController.getStates();
        assertEquals(2, states.size());
        assertEquals("on", states.get("plug1"));
        assertEquals("off", states.get("plug2"));
        mqttController.handleUpdate("testPrefix/update/plug1/power", new
MqttMessage("100".getBytes()));
        mqttController.handleUpdate("testPrefix/update/plug2/power", new
MqttMessage("50".getBytes()));
        Map<String, String> powers = mqttController.getPowers();
        assertEquals(2, powers.size());
        assertEquals("100", powers.get("plug1"));
        assertEquals("50", powers.get("plug2"));
```

```
    }

}
```

PlugsResourceTests.java

```
package ece448.iot_sim;

import static org.junit.Assert.*;

import java.util.Arrays;

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

import ece448.grading.GradeP3.MqttController;
import ece448.iot_hub.HubConfig;

public class PlugsResourceTests {

    private Object[] runSimAndHub() throws Exception {

        String broker = "tcp://127.0.0.1";
        String topicPrefix =
System.currentTimeMillis()+"/test/iot_ece448";

        SimConfig config = new SimConfig(
            8080, Arrays.asList("xx", "yy", "zz.666"),
            broker, "testee/iot_sim", topicPrefix);
        HubConfig hubConfig = new HubConfig(
            8088, broker, "testee/iot_hub", topicPrefix);

        Thread simThread = new Thread() {
            public void run() {
                try (ece448.iot_sim.Main m = new
ece448.iot_sim.Main(config))
                {
                    // loop forever
                    for (;;)
                    {
                        Thread.sleep(60000);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
```

```
        };
        simThread.start();

        Thread hubThread = new Thread() {
            public void run() {
                try (ece448.iot_hub.Main m = new
ece448.iot_hub.Main(hubConfig, new String[0]))
                {
                    // loop forever
                    for (;;)
                    {
                        Thread.sleep(60000);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
        hubThread.start();

        MqttController mqtt = new MqttController(broker, "grader/iot_hub",
topicPrefix);
        mqtt.start();

        Thread.sleep(3000);

        return new Object[]{mqtt, simThread, hubThread};
    }

    private void close(Object[] materials) throws Exception {
        ((Thread)materials[1]).interrupt();
        ((Thread)materials[2]).interrupt();
        ((MqttController)materials[0]).close();
    }

    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();
    }

    @Test
    public void testPlugOn() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];
```

```
        getHub("/api/plugs/yy?action=on");
        Thread.sleep(1000);
        assertTrue("on".equals(mqtt.getState("yy")));
    }

    @Test
    public void testPlugOff() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        mqtt.publishAction("xx", "on");
        Thread.sleep(1000);
        assertTrue("on".equals(mqtt.getState("xx")));

        getHub("/api/plugs/xx?action=off");
        Thread.sleep(1000);
        assertTrue("off".equals(mqtt.getState("xx")));

        close(materials);
    }

    @Test
    public void testPlugsTwo() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        getHub("/api/plugs/xx?action=on");
        getHub("/api/plugs/yy?action=toggle");
        Thread.sleep(1000);

        assertTrue("on".equals(mqtt.getState("xx")));
        assertTrue("on".equals(mqtt.getState("yy")));
        assertTrue("off".equals(mqtt.getState("zz.666")));

        close(materials);
    }

    @Test
    public void testPlugToggle() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        getHub("/api/plugs/yy?action=toggle");
        Thread.sleep(1000);
```

```
        assertTrue("on".equals(mqtt.getState("yy")));

        close(materials);
    }

    @Test
    public void testState() throws Exception {
        Object[] materials = runSimAndHub();

        String rsp = getHub("/api/plugs/xx");
        assertTrue(rsp.contains("off") && !rsp.contains("on"));

        close(materials);
    }

    @Test
    public void testOn() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        mqtt.publishAction("xx", "on");
        Thread.sleep(1000);
        assertTrue("on".equals(mqtt.getState("xx")));

        String rsp = getHub("/api/plugs/xx");
        assertTrue(rsp.contains("on") && !rsp.contains("off"));

        close(materials);
    }

    @Test
    public void testOnCheck() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        mqtt.publishAction("zz.666", "on");
        Thread.sleep(1000);
        assertTrue("on".equals(mqtt.getState("zz.666")));

        String rsp = getHub("/api/plugs/zz.666");
        assertTrue(rsp.contains("on") && !rsp.contains("off")
            && rsp.contains("666.000") && !rsp.contains("0.000"));

        close(materials);
    }
```

```
    @Test
    public void testOnGet() throws Exception {
        Object[] materials = runSimAndHub();

        getHub("/api/plugs/xx?action=on");
        Thread.sleep(1000);
        String rsp = getHub("/api/plugs/xx");
        assertTrue(rsp.contains("on") && !rsp.contains("off"));

        close(materials);
    }

    @Test
    public void testToggleOffUpdate() throws Exception {
        Object[] materials = runSimAndHub();
        MqttController mqtt = (MqttController)materials[0];

        getHub("/api/plugs/yy?action=toggle");
        Thread.sleep(1000);

        String rsp = getHub("/api/plugs/yy");
        assertTrue("on".equals(mqtt.getState("yy")));
        assertTrue(rsp.contains("on") && !rsp.contains("off"));

        getHub("/api/plugs/yy?action=off");
        Thread.sleep(1000);

        rsp = getHub("/api/plugs/yy");
        assertTrue("off".equals(mqtt.getState("yy")));
        assertTrue(rsp.contains("off") && !rsp.contains("on"));

        close(materials);
    }

    @Test
    public void testToggleOffCheck() throws Exception {
        Object[] materials = runSimAndHub();

        String rsp = getHub("/api/plugs/zz.666");
        assertTrue(rsp.contains("off") && !rsp.contains("on")
            && rsp.contains("0.000") && !rsp.contains("666.000"));

        getHub("/api/plugs/zz.666?action=on");
        Thread.sleep(1000);
```

```
        rsp = getHub("/api/plugs/zz.666");
        assertTrue(rsp.contains("on") && !rsp.contains("off")
            && rsp.contains("666.000") && !rsp.contains("0.000"));

        getHub("/api/plugs/zz.666?action=toggle");
        Thread.sleep(1000);

        rsp = getHub("/api/plugs/zz.666");
        assertTrue(rsp.contains("off") && !rsp.contains("on")
            && rsp.contains("0.000") && !rsp.contains("666.000"));

        close(materials);
    }

}
```