Deliverables:

Project 1:

I.

Overview

While it is still at the beginning of the semester and we haven't learned much of the Java ecosystem, it is a good time for us to utilize Project 1 to warm up your Java skills, incorporate new knowledge of logging and testing, and make sure the tools and the Git repository properly for a productive semester.

You probably don't need to write many lines of code for this project. Nevertheless, we would still like to practice Agile software development by going through a complete cycle. In particular, we'll talk about the requirements analysis and the testing in this document.

II.

Project Requirements

Requirement analysis addresses the needs of the end-users of your software products. Many tools and procedures are developed to help analyze the requirements better such as by using user stories. Nevertheless, our focus is to warm up your Java skills in this project, we won't face end users and will just describe the requirements in plain English.

A.

State of a Plug

One should be able to get the state of a plug by calling PlugSim.isOn(), which either returns true if the plug is on, or returns false if the plug is off.

B.

Switch On a Plug

One should be able to switch on a plug by calling PlugSim.switchOn().

C.

Switch Off a Plug

One should be able to switch off a plug by calling PlugSim.switchOff().

D.

Toggle a Plug

One should be able to toggle a plug by calling PlugSim.toggle().

E.

Measure Power Consumption

One should be able to request to measure the power consumption of a plug by calling PlugSim.measurePower() and to read the last power measurement by PlugSim.getPower()

F.

Power Reading of Plugs in Off State

The power reading should be 0 if the plug is off.

G.

Power Reading of Specific Plugs

If the plug has a name of “name.P” and the plug is on, the power reading should be P. This will help us to test the power measurement functionality.

III.

Testing Procedures

Testing for Project 1 is separated into two parts: unit testing and acceptance testing.

A.

Unit testing is applied to individual classes to make sure they work as designed. The only class we work with for Project 1 is PlugSim, so you will need to write your own unit tests to make sure that PlugSim works. Two examples of unit tests are provided in src/test/java/ece448/iot_sim/PlugSimTests.java.

B.

Acceptance testing is used to make sure all project requirements are met. Our acceptance testing procedures are implemented in ece448.grading.GradeP1 and you can use gradle grade_p1 to execute all of them. It should be fairly straightforward to verify all requirements are covered.

Note that in Project 1, we only work with a single class, unit testing, and acceptance testing may look pretty much the same. In upcoming projects, when we need to work with multiple classes, they should be quite different.

Project 2:

I.

Overview

In this project, we will work on features that allow us to control the IoT simulator via web pages using the HTTP protocol. For actual smart plugs, these features make it possible to turn a light on or off just by using a browser on your smartphone without the need to install any apps. You may access the web page following Section III.C of the Project Instructions but please keep in mind the web page may or may not work depending on your progress of this project.

User story is a widely used tool for requirements analysis in Agile software development. Each user story uses a short and informal piece of natural language to describe one or more desired features for both end users and developers. Usually, one organizes a user story using the template "As a ... (who), I want ... (what), so that ... (why)." Using such a template will help to meet the so-called SMART criteria for the user story to be specific, measurable, achievable, relevant, and time-bound. End users prefer to communicate using natural language but too much informal description makes developers difficult to evaluate what should be done – user stories achieve a balance between the two.

You will find the user stories describing the requirements in the next section. You will implement the features in the HTTPCommands class so please first read its source code. Pay attention to the method report as you could call this method to generate web pages. Locate the comment starting with // P2: and your code should be implemented there.

Once you have some rough ideas on how to implement the features, you should start to create unit tests in a file like src/test/java/ece448/iot_sim/HTTPCommandsTests.java. Follow the red-green cycle to add unit tests and to implement desired features.

II.

User Stories

1.

Plug Report

As an end-user, I want to access the web page containing a report of the plug with the name "plugName" at the path /plugName without any query string, so that I can view the report using a browser. The report should include whether the switch is on and its power reading.

2.

Toggle or Switch a Plug On/Off

As an end-user, I want to control the plug with the name “plugName” at the path /plugName with a query string, so that I can control the plug using a browser. To toggle a plug, the query string is action=toggle. To switch a plug to be on, the query string is action=on. To switch a plug to be off, the query string is action=off.

3.

Control Feedback

As an end-user, I want to receive the up-to-date report as the response to the path /plugName with a query string, so that I can verify that the plug acts properly.

III.

Testing Procedures

The testing procedures are implemented in ece448.grading.GradeP2. It should be straightforward to verify all user stories are covered.

Project 3:

I.

Overview

In this project, we are going to work with the MQTT protocol that allows us to control the IoT simulator and report its status. For your convenience, the Eclipse MosquittoTM MQTT broker is installed in our course virtual machine.

You should be able to find the user stories describing the requirements in the next section. While we are going to discuss possible class designs and implementations in the lectures, you are free to choose designs and implementations you are comfortable with. Don’t forget to follow the red-green cycle to add unit tests.

II.

User Stories

1.

Plug On/Off Updates

As an end-user, I want to receive MQTT messages when a plug is turned on or off, so that I can monitor the plug on/off events using an MQTT client. For a plug with the name "plugName" and a configuration string "prefix", the topic should be prefix/update/plugName/state, and the message should be either "on" or "off". Note that the prefix can have the character / multiple times, e.g. illinoistech/ece448/unit_test.

2.

Plug Power Updates

As an end-user, I want to receive MQTT messages when the power consumption of a plug is measured, so that I can monitor the plug power consumption using an MQTT client. For a plug with the name "plugName" and a configuration string "prefix", the topic should be prefix/update/plugName/power, and the message should be the power consumption in plain text.

3.

Toggle or Switch a Plug On/Off

As an end-user, I want to send MQTT messages to toggle or switch on/off a plug, so that I can control the plug using an MQTT client. For a plug with the name "plugName" and a configuration string "prefix", the topic should be prefix/action/plugName/actionString, where the actionString is one of "toggle", "on", or "off".

III.

Testing Procedures

The testing procedures are implemented in ece448.grading.GradeP3. It should be fairly straightforward to verify all user stories are covered.

Project 4:

I.

Overview

In this project, we will build the server backend for our IoT hub as a Spring Boot application. The server backend will provide RESTful services to the frontend web application that we will build in Project 6. To control smart plugs and obtain their state updates, the server backend will communicate with one or more IoT simulators via an MQTT broker.

User stories are described in the next section. While we are going to discuss the possible class design and implementations in the lecture, you are free to choose designs and implementations that you are comfortable with.

You should be able to follow the red-green cycle to add unit tests and implement your classes by now. For convenience, you may need to utilize MQTT and HTTP communications to test some of your classes. Please refer to GradeP3.java and GradeP4.java for useful codes.

II.

User Stories

1.

State of a Single Plug

As an end-user, I want to query the state of the plug “plugName” via a GET request to /api/plugs/plugName, so that I can obtain the state of individual plugs in a web application. The response should be a JSON object in the format, e.g.

{"name":"plugName", "state":"on", "power":100}.

The value for “state” could also be “off”.

2.

States of All Plugs

As an end-user, I want to query the states of all plugs via a GET request to /api/plugs, so that I can obtain all of them at once in a web application. The response should be a JSON array of objects, where each represents the state of a single plug.

3.

Control a Single Plug

As an end-user, I want to switch on/off or toggle the plug “plugName” via a GET request to /api/plugs/plugName with a query string, so that I can control it in a web application. To toggle the plug, the query string is action=toggle. To switch the plug on, the query string is action=on. To switch the plug off, the query string is action=off.

III.

Testing Procedures

The testing procedures are implemented in ece448.grading.GradeP4. It should be fairly straightforward to verify all user stories are covered. Note that we also check the IoT simulators and the MQTT broker to make sure everything works properly.

Codes:

1. ~/iot_ece448/src/main/java/ece448/grading/GradeP4.java

Grade P4.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);
        }
}
```

2. ~/iot_ece448/src/main/java/ece448/iot_hub/HubConfig.java

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;
        }
}
```

3. ~/iot_ece448/src/main/java/ece448/iot_hub/App.java

App.java

```
package ece448.iot_hub;
```

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

@SpringBootApplication
public class App {
  @Autowired
  Environment env;

  @Bean
  public HubMqttController hubMqttController() throws Exception {
    return new HubMqttController(
      env.getProperty("mqtt.broker"),
      env.getProperty("mqtt.clientId"),
      env.getProperty("mqtt.topicPrefix"));
  }
}
```

4. ~/iot_ece448/src/main/java/ece448/iot_hub/Main.java

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);
}
```

5. ~/iot_ece448/src/main/java/ece448/iot_hub/PlugController.java

PlugController.java

```
package ece448.iot_hub;

import java.util.List;
```

```
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 PlugController {
  private final HubMqttController mqtt;

  public PlugController(HubMqttController mqtt) {
    this.mqtt = mqtt;
  }

  @GetMapping("/api/plugs/{plugName}")
  public Plug getPlug(@PathVariable("plugName") String plugName) {
    return new Plug(
      plugName,
      mqtt.getState(plugName),
      mqtt.getPower(plugName));
  }

  @GetMapping("/api/plugs")
  public List<Plug> getPlugs() {
    return mqtt.getPlugs();
  }
```

```
    @GetMapping("/api/plugs/{plugName}/control")
    public void controlPlug(
        @PathVariable("plugName") String plugName,
        @RequestParam("action") String action) {
        if (action.equals("on") || action.equals("off") || action.equals("toggle")) {
            mqtt.publishAction(plugName, action);
        }
    }
}
```

6. ~/iot_ece448/src/main/java/ece448/iot_hub/HubMqttController.java

HubMqttController.java

```
package ece448.iot_hub;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.eclipse.paho.client.mqttv3.IMqttMessageListener;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HubMqttController implements AutoCloseable {
```

```
    private final MqttClient client;
    private final String topicPrefix;
    private final Map<String, String> states = new ConcurrentHashMap<>();
    private final Map<String, Double> powers = new ConcurrentHashMap<>();
    private final Logger logger = LoggerFactory.getLogger(HubMqttController.class);

    public HubMqttController(String broker, String clientId, String topicPrefix) throws
Exception {
        this.topicPrefix = topicPrefix;

        // Initialize MQTT client with memory persistence
        this.client = new MqttClient(broker, clientId, null);

        MqttConnectOptions options = new MqttConnectOptions();
        options.setAutomaticReconnect(true);
        options.setCleanSession(true);
        client.connect(options);

        // Subscribe to state updates
        client.subscribe(topicPrefix + "/update/+/state", (topic, msg) -> {
            String plugName = topic.split("/")[3];
            String state = new String(msg.getPayload());
            states.put(plugName, state);
            logger.debug("State update: {} {}", plugName, state);
        });
```

```
    // Subscribe to power updates
    client.subscribe(topicPrefix + "/update/+/power", (topic, msg) -> {
        String plugName = topic.split("/")[3];
        double power = Double.parseDouble(new String(msg.getPayload()));
        powers.put(plugName, power);
        logger.debug("Power update: {} {}", plugName, power);
    });
}

public void publishAction(String plugName, String action) {
    try {
        if (!client.isConnected()) {
            client.reconnect();
        }
        String topic = String.format("%s/action/%s/%s", topicPrefix, plugName, action);
        client.publish(topic, new MqttMessage(new byte[0]));
    } catch (Exception e) {
        logger.error("Failed to publish action", e);
    }
}

public String getState(String plugName) {
    return states.getOrDefault(plugName, "off");
}

public double getPower(String plugName) {
```

```
        return powers.getOrDefault(plugName, 0.0);
    }

    public List<Plug> getPlugs() {
        // Return all plugs that have state information
        List<Plug> plugs = new ArrayList<>();
        for (Map.Entry<String, String> entry : states.entrySet()) {
            plugs.add(new Plug(
                entry.getKey(),
                entry.getValue(),
                powers.getOrDefault(entry.getKey(), 0.0)));
        }
        return plugs;
    }

    @Override
    public void close() throws Exception {
        if (client.isConnected()) {
            client.disconnect();
        }
        client.close();
    }
}
```

7. ~/iot_ece448/src/main/java/ece448/iot_hub/Plug.java

Plug.java

```
package ece448.iot_hub;
```

```
public class Plug {
    private final String name;
    private final String state;
    private final double power;

    public Plug(String name, String state, double power) {
        this.name = name;
        this.state = state;
        this.power = power;
    }

    // Getters
    public String getName() { return name; }
    public String getState() { return state; }
    public double getPower() { return power; }
}
```