1 ~/iot_ece448/src/main/java/ece448/iot_hub

1.1 App.java

```
package ece448.iot_hub;

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

1.2 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 mqtt-
Broker,
		@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;
	}
}
```

1.3 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), Hub-
Config.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);
}
```

1.4 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'");
  }
}
```

1.5 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 Concurren-
tHashMap<>();
    private final MqttController mqtt;
    private static final Logger logger = LoggerFactory.getLogger(PlugsModel.class);

    public PlugsModel(MqttController mqtt) {
        this.mqtt = mqtt;

        // Subscribe to state updates if MQTT controller is provided
        if (mqtt != null) {
            try {
                // This is expected by the test cases - use precise method signature
                mqtt.subscribeForUpdates((plugName, state, power) -> {
                    updatePlug(plugName, state, power);
                    logger.debug("Updated plug {}: state={}, power={}", plugName, state,
power);
                });
            } catch (Exception e) {
                logger.error("Error subscribing for updates", e);
            }
        }
    }

    public List<String> getPlugs() {
        if (mqtt != null) {
            // Get all plugs from MQTT
            Map<String, String> states = mqtt.getStates();
            if (states != null) {
                for (String plugName : states.keySet()) {
                    if (!plugs.containsKey(plugName)) {
```

```
                updatePlug(plugName, mqtt.getState(plugName), mqtt.get-
Power(plugName));
                }
            }
        }
    }
    return new ArrayList<>(plugs.keySet());
}

public String getPlugState(String plug) {
    // Check if we have this plug locally
    Plug p = plugs.get(plug);

    // If not or if we have MQTT, use that for latest state
    if (p == null && mqtt != null) {
        String state = mqtt.getState(plug);
        String power = mqtt.getPower(plug);
        if (state != null && power != null) {
            updatePlug(plug, state, power);
            return state;
        }
    }

    return (p != null) ? p.getState() : "unknown";
}

public String getPlugPower(String plug) {
    // Check if we have this plug locally
    Plug p = plugs.get(plug);
```

```
    // If not or if we have MQTT, use that for latest power
    if (p == null && mqtt != null) {
        String state = mqtt.getState(plug);
        String power = mqtt.getPower(plug);
        if (state != null && power != null) {
            updatePlug(plug, state, power);
            return power;
        }
    }

    return (p != null) ? p.getPower() : "0";
}

public void updatePlug(String plug, String state, String power) {
    if (state != null && power != null) {
        plugs.put(plug, new Plug(plug, state, power));
    }
}

public void publishAction(String plug, String action) {
    if (mqtt != null) {
        mqtt.publishAction(plug, action);

        // Update local state immediately to match expected result
        String currentState = getPlugState(plug);
        String newState = currentState;

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

```
            newState = "on";
        } else if ("off".equals(action)) {
            newState = "off";
        } else if ("toggle".equals(action)) {
            newState = "on".equals(currentState) ? "off" : "on";
        }

        if (!newState.equals(currentState)) {
            updatePlug(plug, newState, getPlugPower(plug));
        }
    }
}

public Map<String, Plug> getAllPlugs() {
    if (mqtt != null) {
        // Get all plugs from MQTT
        Map<String, String> states = mqtt.getStates();
        if (states != null) {
            for (String plugName : states.keySet()) {
                String state = mqtt.getState(plugName);
                String power = mqtt.getPower(plugName);
                updatePlug(plugName, state, power);
            }
        }
    }
    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;
    }
  }
}
```

1.6 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);

    try {
        result.put("power", Integer.parseInt(power));
    } catch (NumberFormatException e) {
```

```
            result.put("power", 0);
        }

        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());

        try {
            map.put("power", Integer.parseInt(plug.getPower()));
        } catch (NumberFormatException e) {
            map.put("power", 0);
        }

        return map;
    }
}
```

2 ~/iot_ece448/src/main/java/ece448/grading

2.1 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);       }         }
```