/home/ece448s25/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.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class App {

  public static void main(String[] args) {
    SpringApplication.run(App.class, args);
  }

}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/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; }
}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/HubSpringConfig.java

```
package ece448.iot_hub;

import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;


@Configuration

public class HubSpringConfig {

  /**

   * Create the real MQTTController bean using properties set in Main.

   */

  @Bean

  public MQTTController mqttController(

    @Value("${mqtt.broker}") String broker,

    @Value("${mqtt.clientId}") String clientId,

    @Value("${mqtt.topicPrefix}") String topicPrefix

  ) throws Exception {

    // Connect to the real, system-installed Mosquitto broker

    MQTTController ctl = new MQTTController(broker, clientId, topicPrefix);

    // Subscribe to prefix/update/# to receive simulator messages

    ctl.start();

    return ctl;

  }

}


/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/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,
HubSpringConfig.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);
}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/MockEnvironment.java

```
package ece448.iot_hub;

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

import java.util.ArrayList;
import java.util.List;

/**
 * Simple in-JVM MQTT broker simulator.
 * Supports subscribe(topicFilter, listener) and publish(topic, message).
 * Only supports “#” wildcards at end of filter (e.g. "prefix/update/#").
 */
public class MockEnvironment {
    private static class Subscriber {
        final String filter;
        final IMqttMessageListener listener;
        Subscriber(String filter, IMqttMessageListener listener) {
            this.filter = filter;
            this.listener = listener;
        }
    }

    private final List<Subscriber> subscribers = new ArrayList<>();
```

```
/**
 * Register a subscriber on a topic filter.
 */
public void subscribe(String topicFilter, IMqttMessageListener listener) {
   subscribers.add(new Subscriber(topicFilter, listener));
}

/**
 * Publish a message on a topic; dispatches to all matching subscribers.
 */
public void publish(String topic, MqttMessage message) {
   for (Subscriber sub : subscribers) {
      if (matches(topic, sub.filter)) {
         try {
            sub.listener.messageArrived(topic, message);
         } catch (Exception e) {
            // for simplicity, just log
            e.printStackTrace();
         }
      }
   }
}

/**
 * Matches only exact or “prefix/#” filters.
 */
```

```
  private boolean matches(String topic, String filter) {
    if (filter.endsWith("/#")) {
      String prefix = filter.substring(0, filter.length() - 2);
      return topic.startsWith(prefix + "/");
    }
    return topic.equals(filter);
  }
}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/MQTTController.java

```
package ece448.iot_hub;

import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
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;

public class MQTTController {
  private final String broker;
  private final String clientId;
  private final String topicPrefix;
```

```java
private final MqttClient client;
private final Map<String, String> states = new HashMap<>();
private final Map<String, String> powers = new HashMap<>();
private static final Logger logger = LoggerFactory.getLogger(MQTTController.class);

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());

   // Initialize all plugs to OFF by default
   for (char c = 'a'; c <= 'g'; c++) {
      String plugName = String.valueOf(c);
      states.put(plugName, "off");
      powers.put(plugName, "0.000");
   }
}

public void start() throws Exception {
   MqttConnectOptions mqttopt = new MqttConnectOptions();
   mqttopt.setCleanSession(true);
   client.connect(mqttopt);
   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, e);
   }
}

synchronized public String setState(String plugName, String value) {
   return states.put(plugName, value);
}

synchronized public String setPower(String plugName, String value) {
   return powers.put(plugName, value);
}

synchronized public String getBroker() {
   return broker;
}
```

```
synchronized public String getState(String plugName) {
   String s = states.get(plugName);
   return (s == null) ? "off" : s;
}

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);
   // strip off "<prefix>/" and split -> [ "update", plugName, field ]
   String[] parts = topic.substring(topicPrefix.length() + 1).split("/");
   if (parts.length != 3 || !"update".equals(parts[0])) {
      return;
   }

   // decode actual payload bytes as UTF-8 text
   String payload = new String(msg.getPayload(), StandardCharsets.UTF_8).trim();
   if ("state".equals(parts[2])) {
      // only allow "on" or "off"
```

```
            states.put(parts[1], payload.equals("on") ? "on" : "off");
        }
        else if ("power".equals(parts[2])) {
            // record the numeric power string
            powers.put(parts[1], payload);
        }
    }

    synchronized public String getPower(String plug) {
        String p = powers.get(plug);
        return (p == null) ? "0.000" : p;
    }
}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/PlugsModel.java

```
package ece448.iot_hub;

import java.util.HashMap;
import java.util.Map;

public class PlugsModel extends PlugsResource{
private final Map<String, String> states;
private final Map<String, String> powers;

public PlugsModel(MQTTController mqtt) throws Exception {
super(mqtt);
states = new HashMap<>();

powers = new HashMap<>();


}


public void setPlugState(String plug, String state) {

states.put(plug, state);

}


public void setPlugPower(String plug, String power) {

powers.put(plug, power);

}

}
```

/home/ece448s25/iot_ece448/src/main/java/ece448/iot_hub/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;
```

```
@RestController
public class PlugsResource {
  private final MQTTController mqttcontroller;
  private static final Logger logger = LoggerFactory.getLogger(PlugsResource.class);

  public PlugsResource(MQTTController mqttcontroller) {
    this.mqttcontroller = mqttcontroller;
  }

  synchronized public void pubAction(String plug, String action) throws Exception {
    mqttcontroller.publishAction(plug, action);
  }

  synchronized public String getPlugsState(String plug) throws Exception {
    return mqttcontroller.getState(plug);
  }

  synchronized public String getPlugsPower(String plug) throws Exception {
    return mqttcontroller.getPower(plug);
  }

  synchronized public Object getAllThePlugs() throws Exception {
    List<HashMap<String, Object>> ter = new ArrayList<>();
    for (String plug : mqttcontroller.getStates().keySet()) {
      HashMap<String, Object> hashMap = makePlug(plug);
```

```
        ter.add(hashMap);
      }
      return ter;
    }

    protected HashMap<String, Object> makePlug(String plug) throws Exception {
      HashMap<String, Object> ter = new HashMap<>();
      ter.put("name", plug);
      ter.put("state", getPlugsState(plug));
      ter.put("power", getPlugsPower(plug));
      return ter;
    }

    @GetMapping("/api/plugs")
    public Object getAllPlugs() throws Exception {
      return getAllThePlugs();
    }

    @GetMapping("/api/plugs/{plug:.+}")
    public Object getplug(@PathVariable("plug") String plug,
                @RequestParam(value = "action", required = false) String action) throws
Exception {
      if (action == null) {
        // Just return the plug state
        Object ter = makePlug(plug);
        logger.info("plug {}: {}", plug, ter);
        return ter;
      }

      if (action.equals("on") || action.equals("off") || action.equals("toggle")) {
        // Perform the action
        if (action.equals("on")) {
            mqttcontroller.setState(plug, "on");
        } else if (action.equals("off")) {
            mqttcontroller.setState(plug, "off");
        } else if (action.equals("toggle")) {
            String currentState = mqttcontroller.getState(plug);
            String newState = "on".equals(currentState) ? "off" : "on";
            mqttcontroller.setState(plug, newState);
        }

        // Also publish the action via MQTT
        pubAction(plug, action);

        // Return updated state
        Object ter = makePlug(plug);
        logger.info("plug {}: {} after action {}", plug, ter, action);
        return ter;
      }

      logger.info("wrong action: {}", action);
      return null;
```

```
    }

    // Simple handler for simulator requests
    @GetMapping("/{plug}")
    public String handleSimulator(@PathVariable("plug") String plug,
                                  @RequestParam(value = "action", required = false) String action) {
        logger.info("Direct simulator request: plug={}, action={}", plug, action);

        if (action != null) {
            if (action.equals("on")) {
                mqttcontroller.setState(plug, "on");
            } else if (action.equals("off")) {
                mqttcontroller.setState(plug, "off");
            } else if (action.equals("toggle")) {
                String currentState = mqttcontroller.getState(plug);
                String newState = "on".equals(currentState) ? "off" : "on";
                mqttcontroller.setState(plug, newState);
            }
        }

        return "OK";
    }
}
```