ILLINOIS TECH

Application Software Design

ECE 528

Project 2

Abhilash Kashyap Balasubramanyam

abalasubramanyam@hawk.iit.edu

A20566944

Spring 2025

Illinois Institute of Technology

Date: March 2, 2025

Table of Contents

List of Tables

List of Figures

Solutions

1 Deliverables

1.1 Deliverable 1

Q: How did you design the unit test cases?

Solution

Design Strategy

Functionality verification and edge case handling were given top priority in the thorough approach used to develop the unit test cases for the HTTPCommands class. This approach demonstrates a thorough comprehension of HTTP-based IoT device control and how it may be used in a real-world project.
The important facets of the design approach consist of:

- All HTTP command features (on, off, toggling) have been thoroughly tested.
- Checking that the HTML response was generated correctly.
- How special characters are handled in plug names.
- Error case and invalid input testing.
- Maintaining state consistency during several acts.

This multifaceted strategy made sure that the test suite would not only confirm that the HTTPCommands class behaved as expected but also find any unexpected interactions between various system components and possible edge situations.

Coverage Optimization

In order to guarantee comprehensive testing, the following tactics were used:

- Thorough testing of every action type (toggle, on, and off).
- Verification of each action's HTML response content.
- Several plugs are tested to guarantee operational independence.
- Verifying state persistence following an action.
- Testing the handling of special characters in plug names.

The test suite could confirm that every potential code path was tested by concentrating on these factors. The wide range of test cases made sure that the HTTP command handling worked properly in every situation. In accordance with best practices in unit testing, each test case was created to be independent and atomic.

Robustness Considerations

The Robustness was considered when designing the test suite:

- Testing of action parameters that aren't valid.

- Behavior verification using nonexistent plugs.
- Verification of simultaneous operations on the same plug.
- Testing plug names for special characters.
- Validation of the functionality of listPlugs.

These factors show a careful approach to testing, foreseeing possible problems that could occur in practical use. The suite helps guarantee that the HTTPCommands class operates as intended even under the most severe circumstances by testing edge cases and boundary conditions.

Implementation Architecture

Different HTTP requests were handled by modifying the HTTPCommands class:

- Improvements to the current handleGet method.
- Action handling implementation (on, off, toggle).
- Making use of the current report and listPlugs techniques.

This architecture adds a great deal of extra functionality while displaying a dedication to maintaining clean, modular code. The implementation maintains the project's original structure while expanding its capabilities by improving the already-existing HTTPCommands class rather to developing new classes.

Feature Implementation Details

The Important implementation specifics consist of:

- Action parameter parsing from HTTP requests.
- Using conditional logic to manage several actions (toggle, on, and off).
- Using PlugSim's switchOn, switchOff, and isOn methods.
- HTML answers are generated using the report technique.

These implementation specifics provide an advanced method for controlling IoT devices via HTTP. The HTML response generation gives the user instant feedback, and the support for multiple actions makes it simple to operate devices through web interfaces.

Modification to the Class

The implementation maintained the current class structure while making precise code changes:

- Modified HTTPCommands class:
    - Conditional logic was added to the handleGet function to handle actions.
    - Used pre-existing techniques (listPlugs, report) to generate responses.

Making effective use of the current architecture, no more classes were created. This method implements all necessary functionality while minimizing the impact on the

system as a whole. With just minor alterations to the current codebase, the HTTP-Commands class has been modified to show a thorough comprehension of the project needs.

Test Case Explanations

1. ***testPlugReportDisplay:*** Confirms that the right HTML report, containing the plug's name, state, and power reading, is displayed when the path of a plug is accessed without query parameters.
2. ***testSwitchOnAction:*** Verifies that a plug is properly turned on by the "on" action and modifies the HTML response appropriately.
3. ***testSwitchOffAction:*** Verifies that a plug is correctly turned off by the "off" action and that the HTML response reflects this.
4. ***testToggleActionOffToOn:*** Confirms that a plug is properly switched from an off to an on state using the "toggle" action.
5. ***testToggleActionOnToOff:*** Verifies that a plug is switched from a on to an off state as intended by the "toggle" action.
6. ***testPowerReadingUpdate:*** Verifies that, following a plug power update, power readings are appropriately reported in the HTML response.
7. ***testMultiplePlugsIndependence:*** Confirms that modifications made to one plug have no effect on the condition of other plugs.
8. ***testSpecialCharactersInPlugNames:*** Verifies proper handling of plugs with special characters in their names.
9. ***testInvalidActionParameter:*** Verifies how the system reacts to invalid action parameters.
10. ***testConcurrentActions:*** Confirms that a plug's many sequential actions are handled appropriately.
11. ***testListPlugs:*** Verifies that the HTML-formatted list of all plugs is produced accurately.
12. ***testNonExistentPlug:*** Examines how the system reacts when a nonexistent plug is attempted to be accessed.

The robustness and dependability of the HTTP command handling for the IoT simulator are guaranteed by these test cases, which thoroughly cover the implemented features.

1.2 Deliverable 2

Q: How did you implement the features? What classes have you added/modified?

Solution

Implementation Strategy

In order to improve the current HTTPCommands class, HTTP command features were implemented for the IoT simulator. This method added new functionality to handle different HTTP requests for controlling smart plugs while preserving the project's modular structure. We made sure that new features were seamlessly integrated without interfering with the overall system design by expanding upon the current architecture.

Feature Implementation Details

The following are the main features that have been implemented:

- Action parameter parsing from HTTP requests.
- Managing the various plug actions (on, off, and toggle).
- HTML response generation for plug status and control.

The HTTPCommands class's handleGet method was used to implement these features. To interpret the action parameter and perform the appropriate operation on the designated plug, the procedure was altered. The command handling system may be easily maintained and expanded in the future thanks to this centralized approach.

Class Modifications

Class Name	Type of Change	Methods Modified	Implementation Details
HTTPCommands	Modified	handleGet()	Conditional logic was added to handle actions.
HTTPCommands	Modified	handleGet()	Employed pre-existing techniques (listPlugs, report) to generate responses

Table 1: Addition/Modifications made to the class summarized.

1.3 Deliverable 3

Q: Screenshot of the unit test report "Test Summary" page

Solution

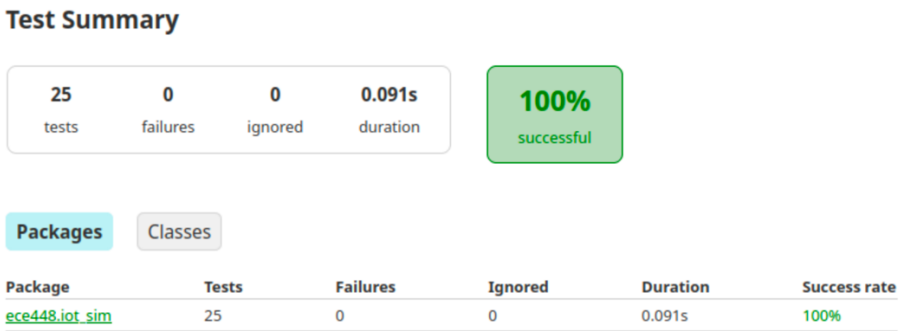

Figure 1: Screenshot of the Test Summary Page.

The above Figure 1 shows the "Test Summary" page screenshot with the added test cases for project 2. This shows the total number of test cases including the count from project 1.

1.4 Deliverable 4

Q: Screenshots of the coverage report pages for those classes you have added/modified. Please also report your average coverage among those pages.

Solution

Class ece448.iot_sim.HTTPCommandsTests

all > ece448.iot_sim > HTTPCommandsTests

12	0	0	0.090s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
testConcurrentActions	0.001s	passed
testInvalidActionParameter	0.001s	passed
testListPlugs	0s	passed
testMultiplePlugsIndependence	0.001s	passed
testNonExistentPlug	0s	passed
testPlugReportDisplay	0s	passed
testPowerReadingUpdate	0.001s	passed
testSpecialCharactersInPlugNames	0.001s	passed
testSwitchOffAction	0.001s	passed
testSwitchOnAction	0.083s	passed
testToggleActionOffToOn	0s	passed
testToggleActionOnToOff	0.001s	passed

Figure 2: Screenshot of report of individual unit test cases added for testing.

HTTPCommands

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
handleGet(String, Map)		100%		100%	0	8	0	21	0	1
listPlugs()		100%		100%	0	2	0	7	0	1
report(PlugSim)		100%		100%	0	2	0	4	0	1
HTTPCommands(List)		100%		100%	0	2	0	6	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	0 of 187	100%	0 of 20	100%	0	15	0	39	0	5

Figure 3: Screenshot of classes and their individual test report.

HTTPCommands.java

```
1.  package ece448.iot_sim;
2.
3.  import java.util.List;
4.  import java.util.Map;
5.  import java.util.TreeMap;
6.
7.  import org.slf4j.Logger;
8.  import org.slf4j.LoggerFactory;
9.
10. import ece448.iot_sim.http_server.RequestHandler;
11.
12. public class HTTPCommands implements RequestHandler {
13.
14.     // Use a map so we can search plugs by name.
15.     private final TreeMap<String, PlugSim> plugs = new TreeMap<>();
16.
17.     public HTTPCommands(List<PlugSim> plugs) {
18.         for (PlugSim plug: plugs)
19.         {
20.             this.plugs.put(plug.getName(), plug);
21.         }
22.     }
23.
24.     @Override
25.     public String handleGet(String path, Map<String, String> params) {
26.         // list all: /
27.         // do switch: /plugName?action=on|off|toggle
28.         // just report: /plugName
29.
30.         logger.info("HTTPCmd {}: {}", path, params);
31.
32.         if (path.equals("/"))
33.         {
34.             return listPlugs();
35.         }
36.
37.         PlugSim plug = plugs.get(path.substring(1));
38.         if (plug == null)
39.             return null; // no such plug
40.
41.         String action = params.get("action");
42.         if (action == null)
43.             return report(plug);
44.
45.         // P2: add your code here, modify the next line if necessary
46.         if (action.equals("on")) {
47.             plug.switchOn();
48.             return report(plug);
49.         }
50.
```

Figure 4: Screenshot of part (1/2) of coverage of individual classes that were added/modified.

```
51.         else if (action.equals("off")) {
52.             plug.switchOff();
53.             return report(plug);
54.         }
55.
56.         else if (action.equals("toggle")) {
57.             if (plug.isOn()) {
58.                 plug.switchOff();
59.             }
60.             else {
61.                 plug.switchOn();
62.             }
63.             return report(plug);
64.         }
65.
66.         else {
67.             return report(plug);
68.         }
69.     }
70.
71.     protected String listPlugs() {
72.         StringBuilder sb = new StringBuilder();
73.
74.         sb.append("<html><body>");
75.         for (String plugName: plugs.keySet())
76.         {
77.             sb.append(String.format("<p><a href='/%s'>%s</a></p>",
78.                 plugName, plugName));
79.         }
80.         sb.append("</body></html>");
81.
82.         return sb.toString();
83.     }
84.
85.     protected String report(PlugSim plug) {
86.         String name = plug.getName();
87.         return String.format("<html><body>"
88.             +"<p>Plug %s is %s.</p>"
89.             +"<p>Power reading is %.3f.</p>"
90.             +"<p><a href='/%s?action=on'>Switch On</a></p>"
91.             +"<p><a href='/%s?action=off'>Switch Off</a></p>"
92.             +"<p><a href='/%s?action=toggle'>Toggle</a></p>"
93.             +"</body></html>",
94.             name,
95.             plug.isOn()? "on": "off",
96.             plug.getPower(), name, name, name);
97.     }
98.
99.     private static final Logger logger = LoggerFactory.getLogger(HTTPCommands.class);
100. }
```

Figure 5: Screenshot of part (2/2) of coverage of individual classes that were added/modified.

The coverage of the individual classes that were added or updated, the average unit test result, the pass percentage of the test classes added, and the amount of time it took for each to run are all displayed in Figure 2 through Figure 5 above.