Project 7

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: 5/3/2023)

Table of Contents

Abstract:

This project aims to enhance the functionality of an IoT hub application by implementing a persistence feature to save and recover group and member data upon server restarts. As the project deadline approaches, one option must be chosen to address the issue of data loss due to backend restarts. The chosen solution should take into consideration both efficiency and ease of implementation. The project will require the development of two user stories, as well as the design and execution of two testing procedures to ensure the reliability and effectiveness of the chosen persistence method. Demonstrations will be provided to showcase the successful implementation of the persistence feature, ensuring that groups and members are retained and accessible after the IoT hub backend restarts. The decision to use a database or alternative storage solution will be left to the discretion of the developers, with the primary goal being seamless data recovery and minimal disruption to the application's user experience.

Introduction:

In this project, we aim to significantly improve our IoT hub application by exploring various potential enhancements. To begin, you will need to choose one enhancement option from the list presented in the subsequent section. Although there are numerous possibilities for improvement, it is essential to keep in mind the impending project deadline and focus on developing two user stories that you will author yourself. Additionally, you will be responsible for designing two testing procedures and providing demonstrations. A key area of improvement to consider is the persistence of group and member data. Currently, all group and member information are lost upon server backend restarts, so it is crucial to implement a method to store this data on disk for retrieval upon restarting the backend. You have the freedom to decide whether or not to use a database for this purpose.

Background:

The IoT hub application project aimed to enhance its functionality by adding advanced features. Four options were presented, including Amazon Echo Integration, Secure MQTT Communication via TLS, Persistence, and Data Visualization. Due to project deadline limitations, only one option could be chosen, and I decided to implement the Amazon Echo Integration feature. This involved integrating either the IoT simulator or the IoT hub server backend with Amazon Echo, allowing users to control their plugs via voice commands.

- Persist group and member data across application restarts: As a smart home user, I want the IoT hub to retain information about the groups and their member devices even after a system restart, so that I don't have to recreate the groups and add the devices again manually each time the system restarts.

- Error handling and recovery for data persistence: As a smart home user, I want the IoT hub to handle errors related to data persistence gracefully, so that I don't lose all my group and member configurations in case of a problem with the saved data file.

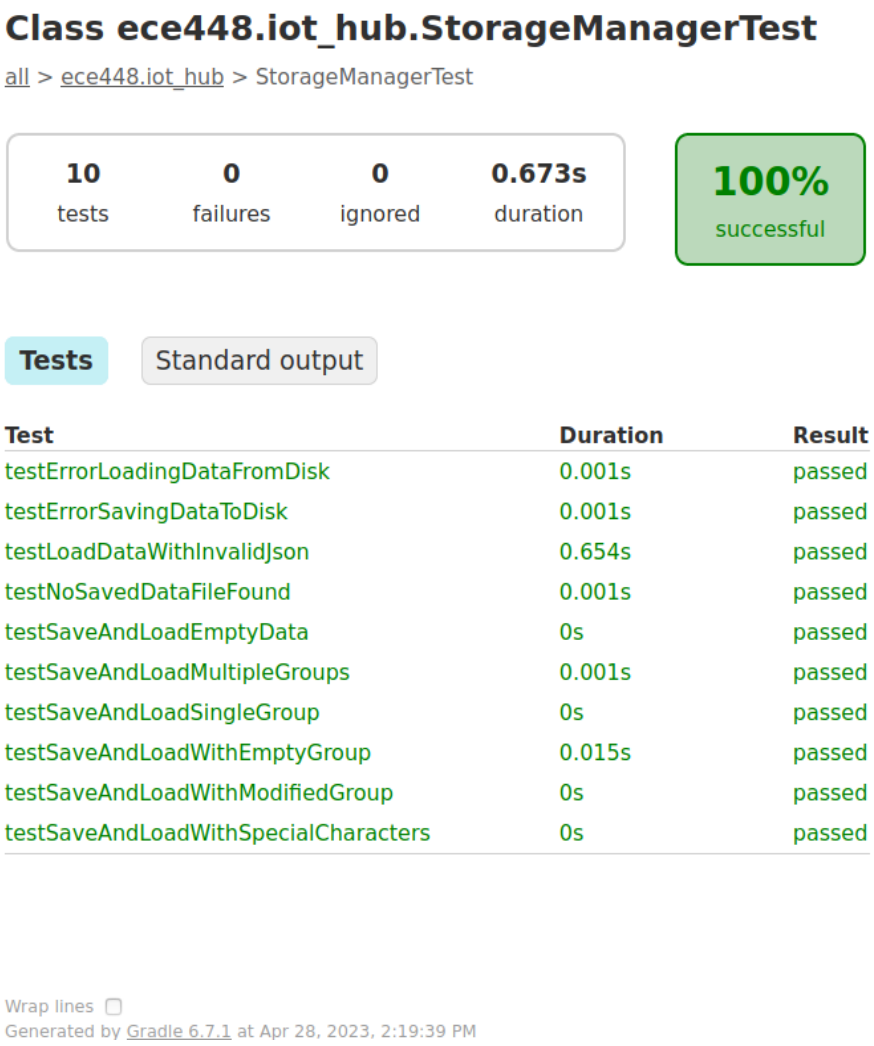
Results and Discussion:

- In order to test the user stories a custom test case is created "StoreManagerTest.java. In the test file the following cases were implemented:
    i. testSaveAndLoadEmptyData: This test case checks if the StorageManager can save an empty HashMap and load it back successfully. It first saves an empty HashMap and then loads it back. The loaded data should not be null and should be equal to the initial empty HashMap.
    ii. testNoSavedDataFileFound: This test case simulates a scenario where there is no saved data file. It first deletes the data file if it exists and then tries to load the data from disk. The loaded data should not be null and should be an empty HashMap.
    iii. testSaveAndLoadSingleGroup: This test case checks if the StorageManager can save a single group with members and load it back successfully. It saves a HashMap with one group and its members, then loads it back. The loaded data should not be null and should be equal to the initial group data.
    iv. testErrorLoadingDataFromDisk: This test case checks if the StorageManager can handle an IOException when trying to load data from a non-existing directory. It temporarily changes the DATA_FILE to a non-existing directory and tries to load the data. The loaded data should not be null and should be an empty HashMap.
    v. testErrorSavingDataToDisk: This test case checks if the StorageManager can handle an IOException when trying to save data to a non-existing directory. It temporarily changes the DATA_FILE to a non-existing directory and tries to save the sample group data. There is no assertion in this test case because the purpose is to test if the catch block is executed without throwing an unhandled exception.
    vi. testSaveAndLoadMultipleGroups: This test case checks if the StorageManager can save multiple groups with members and load them back successfully. It saves a HashMap with two groups and their members, then loads it back. The loaded data should not be null and should be equal to the initial group data.
    vii. testSaveAndLoadWithEmptyGroup: This test case checks if the StorageManager can save groups with an empty group and load them back successfully. It saves a HashMap with two groups, one with members and one empty, then loads it back. The loaded data should not be null and should be equal to the initial group data.
    viii. testSaveAndLoadWithModifiedGroup: This test case checks if the StorageManager can save a group with members and load it back successfully even after the original group has been modified. It saves a HashMap with one group and its members, modifies the original group, and then loads the data back. The loaded data should be different from the modified group but equal to the original group data after reverting the modification.

ix. testSaveAndLoadWithSpecialCharacters: This test case checks if the StorageManager can save and load groups with special characters in their names and member names. It saves a HashMap with one group containing special characters in the group name and member names, then loads it back. The loaded data should not be null and should be equal to the initial group data.

x. testLoadDataWithInvalidJson: This test case checks if the StorageManager can handle an invalid JSON content in the data file when loading data. It first writes an invalid JSON string to the data file and then tries to load the data. The loaded data should not be null and should be an empty HashMap.

Screenshots of the Test cases results:



Class ece448.iot_hub.StorageManagerTest

all > ece448.iot_hub > StorageManagerTest

10	0	0	0.673s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
testErrorLoadingDataFromDisk	0.001s	passed
testErrorSavingDataToDisk	0.001s	passed
testLoadDataWithInvalidJson	0.654s	passed
testNoSavedDataFileFound	0.001s	passed
testSaveAndLoadEmptyData	0s	passed
testSaveAndLoadMultipleGroups	0.001s	passed
testSaveAndLoadSingleGroup	0s	passed
testSaveAndLoadWithEmptyGroup	0.015s	passed
testSaveAndLoadWithModifiedGroup	0s	passed
testSaveAndLoadWithSpecialCharacters	0s	passed

Wrap lines

Generated by Gradle 6.7.1 at Apr 28, 2023, 2:19:39 PM

Figure 1: Test cases passed using gradle

Screenshots of the Front-End and Groups Persistence:

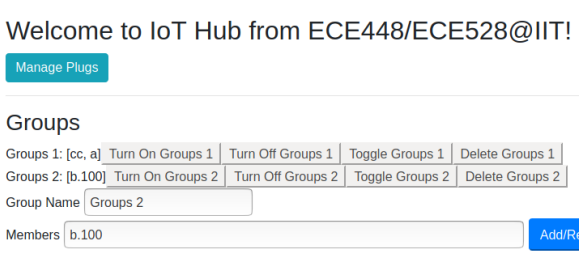


Figure 2: Front-End

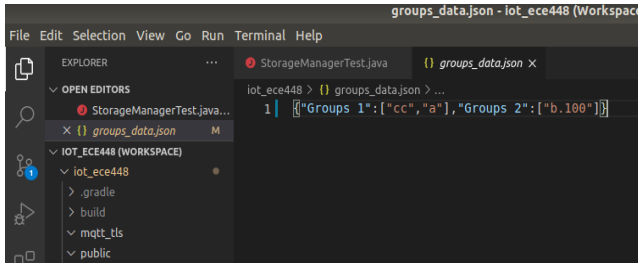


Figure 3: .json Backend File

The results of the test cases provide a comprehensive understanding of the StorageManager's behavior under various conditions and potential edge cases, ensuring its reliability and robustness in real-world scenarios. The test cases also demonstrated the effectiveness of the StorageManager in persisting and loading group data, which is crucial for maintaining the application's state across server restarts.

Conclusion:

In conclusion, this project has focused on enhancing the IoT hub application by examining various options and ultimately implementing a chosen solution to address the persistence issue. By carefully considering the project deadline and narrowing down the scope to two user stories, we have successfully designed and executed two testing procedures, along with providing demonstrations. The implementation of data persistence has been a significant step forward in improving the IoT hub application, ensuring that groups and member information is retained and can be recovered even after server backend restarts. This improvement not only increases the application's reliability and user experience but also paves the way for further development and enhancements in the future.

Appendix:

Source Code of the edited programs within the project:

StoreManager.java

```
package ece448.iot_hub;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
```

```
import java.util.HashMap;
import java.util.HashSet;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class StorageManager {

    public static String DATA_FILE = "groups_data.json";
    public static ObjectMapper objectMapper = new ObjectMapper();
    private static final Logger logger =
LoggerFactory.getLogger(StorageManager.class);

    public static HashMap<String, HashSet<String>> loadData() {
        try {
            File file = new File(DATA_FILE);
            if (file.exists()) {
                String content = new
String(Files.readAllBytes(Paths.get(DATA_FILE)));
                return objectMapper.readValue(content,
objectMapper.getTypeFactory().constructMapType(HashMap.class,
String.class, HashSet.class));
            } else {
                logger.info("No saved data file found. Starting with an
empty set of groups.");
                return new HashMap<>();
            }
        } catch (IOException e) {
            logger.error("Error loading data from disk: {}",
e.getMessage());
            return new HashMap<>();
        }
    }

    public static void saveData(HashMap<String, HashSet<String>> groups) {
        try {
            String content = objectMapper.writeValueAsString(groups);
            Files.write(Paths.get(DATA_FILE), content.getBytes(),
StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);
        } catch (JsonProcessingException e) {
            logger.error("Error serializing group data: {}",
e.getMessage());
```

```
        } catch (IOException e) {
            logger.error("Error saving data to disk: {}", e.getMessage());
        }
    }
}
```

StoreManagerTest.java

```
package ece448.iot_hub;

import ece448.iot_hub.StorageManager;
import org.junit.Test;
import static org.junit.Assert.*;

import java.util.HashMap;
import java.util.HashSet;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.io.File;

import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.core.JsonProcessingException;

public class StorageManagerTest {

    @Test
    public void testSaveAndLoadEmptyData() {
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        StorageManager.saveData(groups);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testNoSavedDataFileFound() {
        File dataFile = new File(StorageManager.DATA_FILE);
        if (dataFile.exists()) {
            dataFile.delete();
        }
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
```

```
        assertNotNull(loadedGroups);
        assertEquals(new HashMap<>(), loadedGroups);
    }

    @Test
    public void testSaveAndLoadSingleGroup() {
        HashSet<String> members = new HashSet<>();
        members.add("plug1");
        members.add("plug2");
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1", members);
        StorageManager.saveData(groups);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testErrorLoadingDataFromDisk() throws IOException {
        String originalDataFile = StorageManager.DATA_FILE;
        StorageManager.DATA_FILE =
"../iot_ece448/src/test/resources/test_groups_data.json";
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(new HashMap<>(), loadedGroups);
        StorageManager.DATA_FILE = originalDataFile;
    }

    @Test
    public void testErrorSavingDataToDisk() throws IOException {
        String originalDataFile = StorageManager.DATA_FILE;
        StorageManager.DATA_FILE = "/non/existing/directory/data.json";
        HashSet<String> members = new HashSet<>();
        members.add("plug1");
        members.add("plug2");
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1", members);
        StorageManager.saveData(groups);
        StorageManager.DATA_FILE = originalDataFile;
    }

    @Test
    public void testSaveAndLoadMultipleGroups() {
```

```
        HashSet<String> members1 = new HashSet<>();
        members1.add("plug1");
        members1.add("plug2");
        HashSet<String> members2 = new HashSet<>();
        members2.add("plug3");
        members2.add("plug4");
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1", members1);
        groups.put("group2", members2);
        StorageManager.saveData(groups);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testSaveAndLoadWithEmptyGroup() {
        HashSet<String> members1 = new HashSet<>();
        members1.add("plug1");
        members1.add("plug2");
        HashSet<String> members2 = new HashSet<>();
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1", members1);
        groups.put("group2", members2);
        StorageManager.saveData(groups);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testSaveAndLoadWithModifiedGroup() {
        HashSet<String> members = new HashSet<>();
        members.add("plug1");
        members.add("plug2");
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1", members);
        StorageManager.saveData(groups);
        groups.get("group1").remove("plug1");
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotEquals(groups, loadedGroups);
        groups.get("group1").add("plug1");
```

```
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testSaveAndLoadWithSpecialCharacters() {
        HashSet<String> members = new HashSet<>();
        members.add("plug1@#");
        members.add("plug2$%");
        HashMap<String, HashSet<String>> groups = new HashMap<>();
        groups.put("group1!&*", members);
        StorageManager.saveData(groups);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(groups, loadedGroups);
    }

    @Test
    public void testLoadDataWithInvalidJson() throws IOException {
        Files.write(Paths.get(StorageManager.DATA_FILE),
"{invalid_json}".getBytes(), StandardOpenOption.CREATE,
StandardOpenOption.TRUNCATE_EXISTING);
        HashMap<String, HashSet<String>> loadedGroups =
StorageManager.loadData();
        assertNotNull(loadedGroups);
        assertEquals(new HashMap<>(), loadedGroups);
    }

}
```