

Capstone Project

GROUP3



Roles:

Linxin Zhang: Graphic User Interfaces, Energy and Battery Management

Yanal Al Halabi: Device, logging management and system monitoring

Anjali Bodke: Initial House Configuration

Franklin Viegas: Creation of Unit Testing

Functional Requirements

Application Initialization

- Initialize the system by loading configurations (e.g., devices, batteries, and energy sources)

Device Management

- Add, remove, and list devices.

Battery Management

- List all batteries.
- Start and stop charging batteries.
- Start and stop powering devices using battery.

Energy Source Management

- Add, remove, and list energy sources.
- Toggle the state (active/inactive) of an energy source.

System Monitoring

- Monitor total power consumption and battery charge periodically.
- Log warnings when power consumption exceeds available battery charge.

Logging

- Log events related to devices, batteries, and energy sources (e.g., addition, removal, state changes).
- Categorized by type (e.g., DEVICE, BATTERY, ENERGY, SYSTEM).
- Search logs by name or date.
- Delete and archive logs.

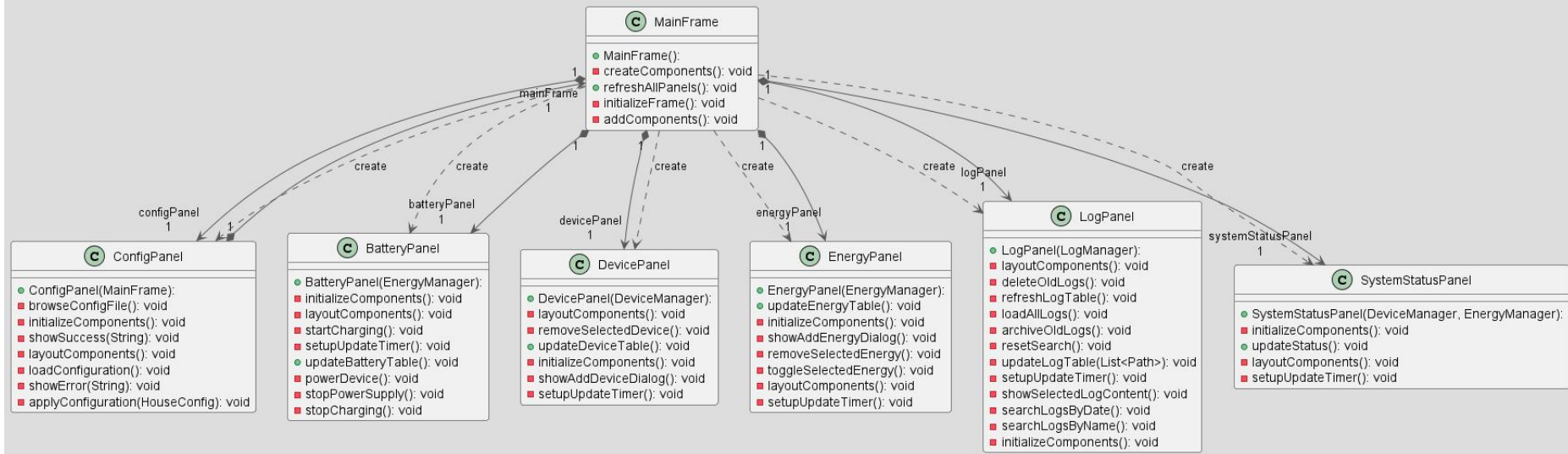
Graphic User Interfaces

- Provide Graphic User Interfaces for operations:
 - Devices, Batteries, Energy sources, Logs, System configuration and system status
- Validate user inputs and provide feedback.

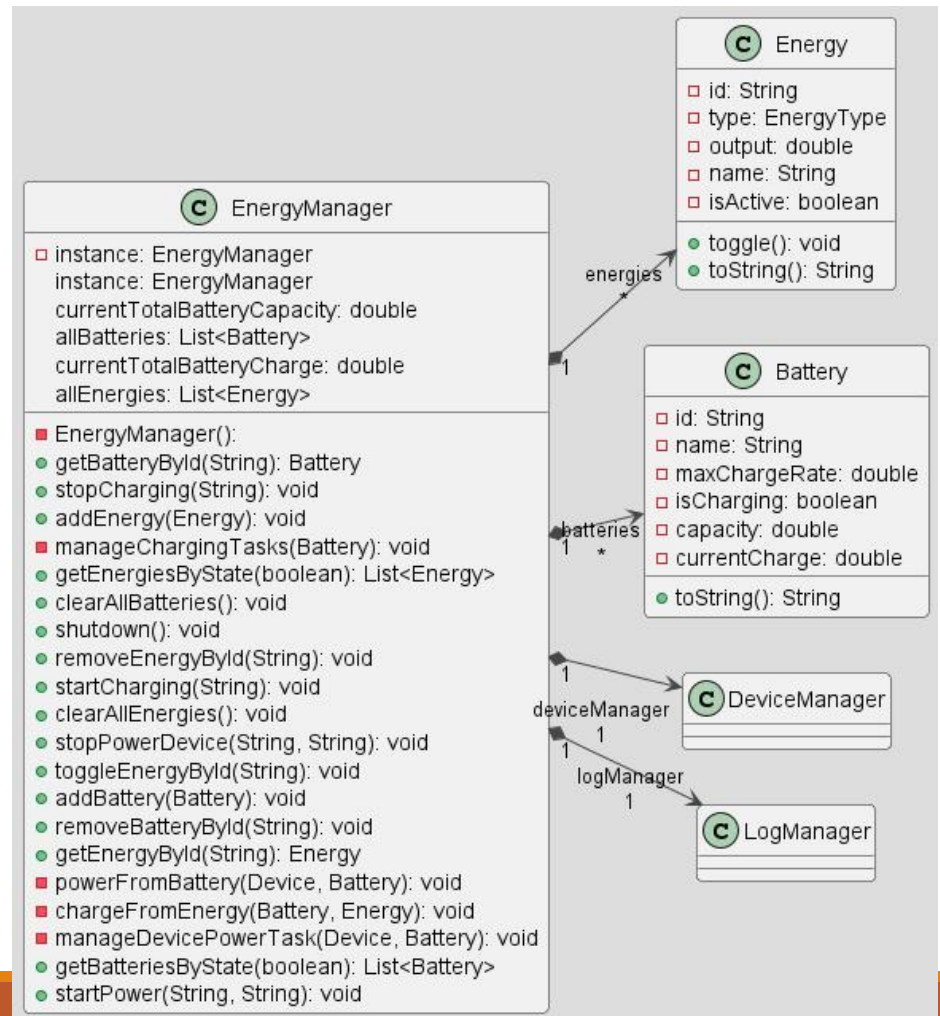
Configuration Loading

- Load a new configuration file.
- Define devices, batteries, and energy sources with attributes like name, type, and capacity.

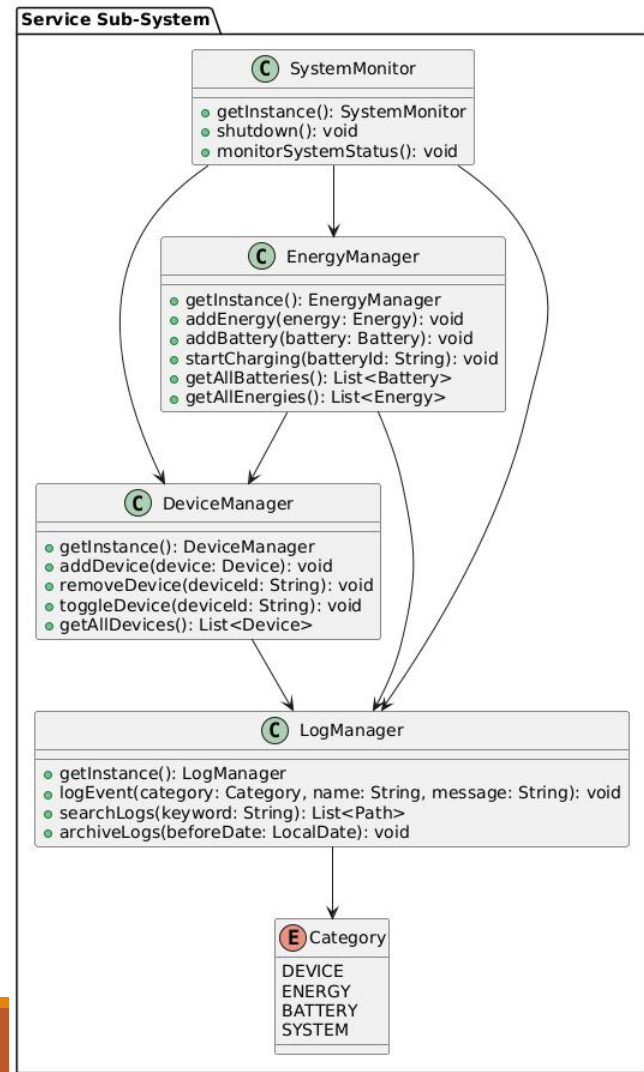
Graphic User Interfaces



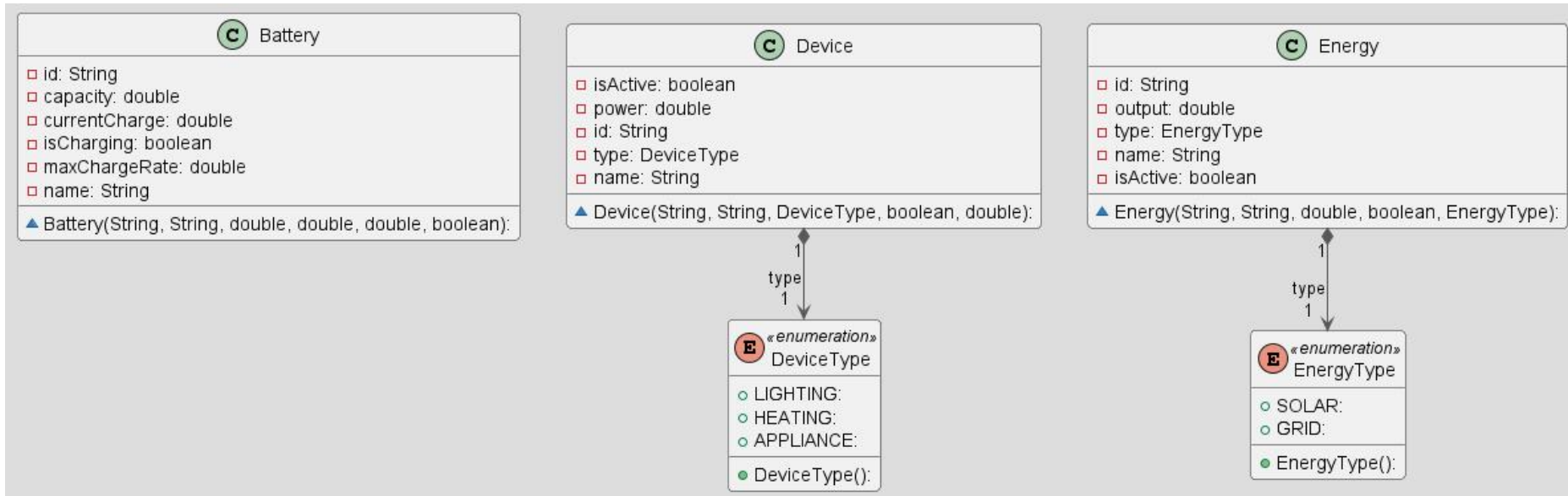
Energy Management



Service



Model



Utility

Utility Sub-System

LoggerHelper

- logEvent(logManager: LogManager, category: LogManager.Category, action: String, name: String, additionalInfo: String): void
- logEnergyEvent(logManager: LogManager, action: String, energyName: String): void
- logBatteryEvent(logManager: LogManager, action: String, batteryName: String): void

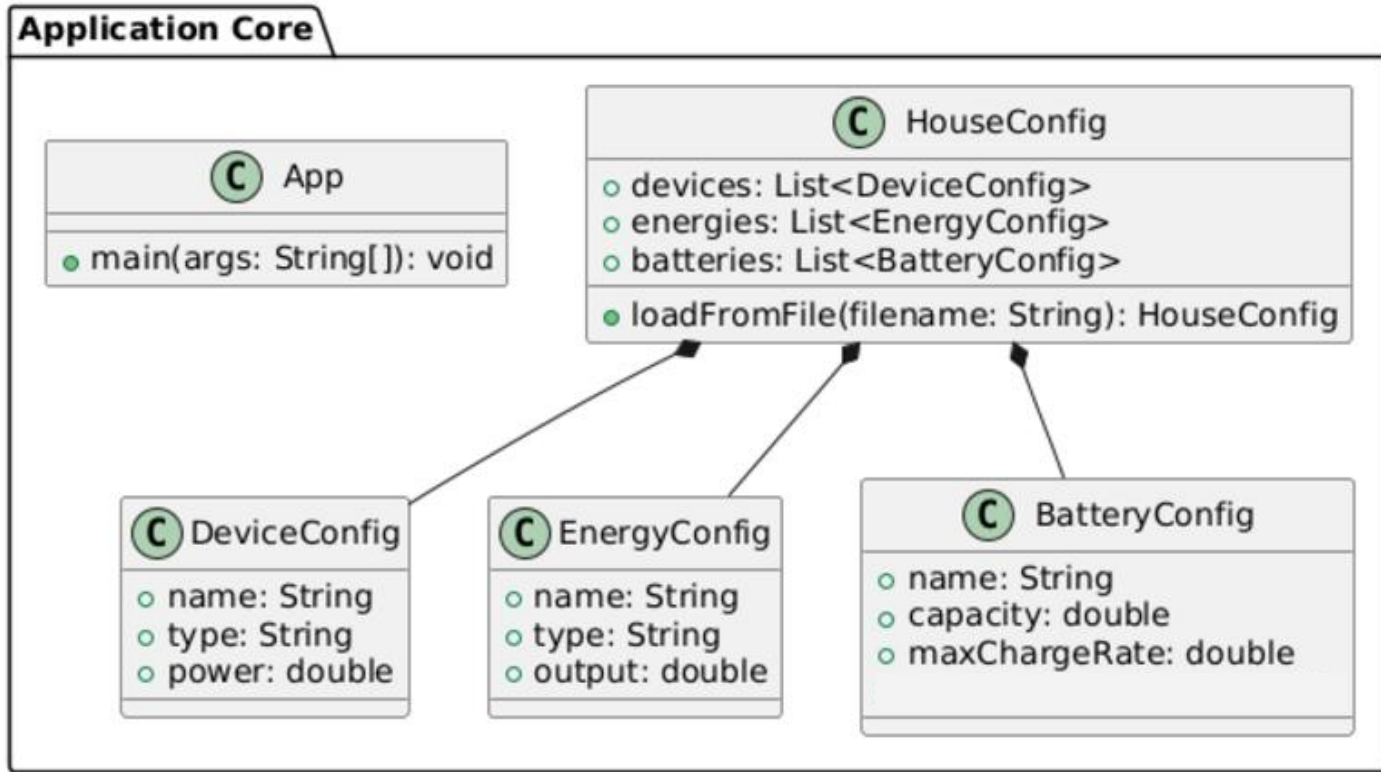
LogManager

- instance: LogManager
- LOG_DIR: Path
- TIME_FORMAT: DateTimeFormatter
- ARCHIVE_DIR: Path
- DATE_FORMAT: DateTimeFormatter
- log: Logger
- isLogFileBeforeDate(Path, LocalDate, DateTimeFormatter): boolean
- getInstance(): LogManager
- deleteLogs(LocalDate): void
- initializeDirectories(): void
- readLogFile(Path): List<String>
- searchLogs(String): List<Path>
- deleteLogFile(Path): void
- logEvent(Category, String, String): void
- writeToLog(Path, LocalDateTime, String): void
- clearAllLogs(): void
- archiveLogFile(Path, ZipOutputStream): void
- archiveLogs(LocalDate): void

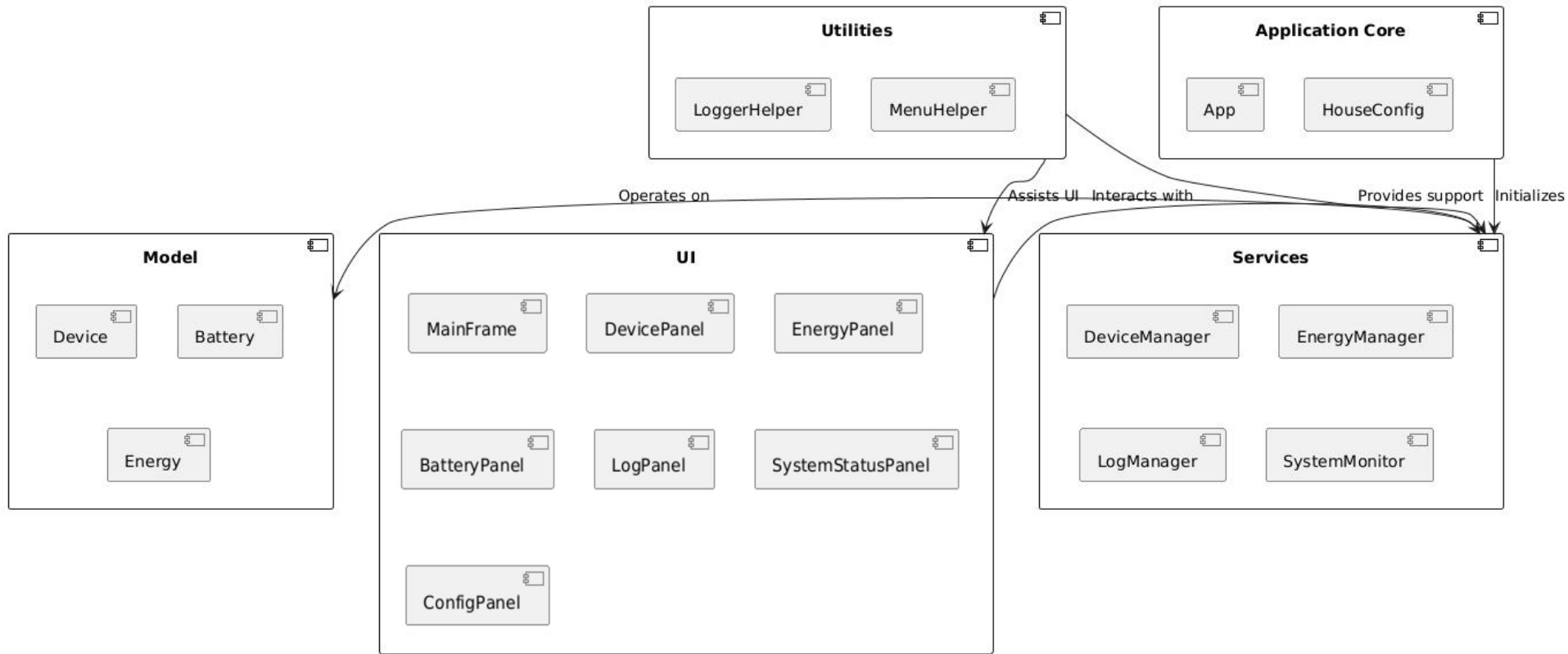
MenuHelper

- getScanner(): Scanner
- clearScreen(): void
- waitForEnter(): void
- getValidChoice(min: int, max: int): int
- getValidDouble(): double

Application Core



Component Diagram



Management I/O in the System

```
public class LogManager {  
    private static volatile LogManager instance;  
  
    private final Path LOG_DIR = Paths.get("logs");  
    private final Path ARCHIVE_DIR = LOG_DIR.resolve("archive");  
    public final DateTimeFormatter DATE_FORMAT = DateTimeFormatter.ofPattern("yyyyMMdd");  
    private final DateTimeFormatter TIME_FORMAT = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");  
  
    @Getter  
    @AllArgsConstructor  
    public enum Category {  
        DEVICE("device"), ENERGY("energy"), BATTERY("battery"), SYSTEM("system");  
  
        private final String value;  
    }  
}
```

```
private void initializeDirectories() {  
    try {  
        Files.createDirectories(LOG_DIR);  
        Files.createDirectories(ARCHIVE_DIR);  
        for (Category category : Category.values()) {  
            Files.createDirectories(LOG_DIR.resolve(category.getValue()));  
        }  
    } catch (IOException e) {  
        Log.error("Failed to initialize log directories", e);  
    }  
}
```

Events of I/O in the System

```
public void logEvent(Category category, String name, String message) {
    LocalDateTime now = LocalDateTime.now();
    String date = now.format(DATE_FORMAT);
    Path logFile = LOG_DIR.resolve(category.getValue()).resolve(String.format("%s_%s.log", name, date));
    Path systemLogFile = LOG_DIR.resolve(Category.SYSTEM.getValue()).resolve("system_" + date + ".log");

    writeToLog(logFile, now, message);
    if (!category.equals(Category.SYSTEM)) {
        writeToLog(systemLogFile, now, String.format("%s: %s", category, message));
    }
}
```

```
public void deleteLogs(LocalDate beforeDate) {
    for (Category category : Category.values()) {
        Path categoryDir = LOG_DIR.resolve(category.getValue());
        if (!Files.exists(categoryDir)) continue;

        try (Stream<Path> paths = Files.list(categoryDir)) {
            paths.filter(path -> isLogFileBeforeDate(path, beforeDate, DATE_FORMAT)).forEach(this::deleteLogFile);
        } catch (IOException e) {
            Log.error("Error deleting logs in category: {}", category, e);
        }
    }
}
```

Managing Files of I/O in the System

```
public void writeToLog(Path logFile, LocalDateTime timestamp, String message) {
    try {
        Files.createDirectories(logFile.getParent());
        try (BufferedWriter writer = Files.newBufferedWriter(logFile, StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {
            writer.write(String.format("[%s] %s%n", timestamp.format(TIME_FORMAT), message));
        }
    } catch (IOException e) {
        Log.error("Failed to write to log file: {}", logFile, e);
    }
}
```

```
public void archiveLogFile(Path logFile, ZipOutputStream zos) {
    try {
        ZipEntry entry = new ZipEntry(logFile.getParent().getFileName() + "/" + logFile.getFileName().toString());
        zos.putNextEntry(entry);
        Files.copy(logFile, zos);
        zos.closeEntry();
        Files.delete(logFile);
    } catch (IOException e) {
        Log.error("Failed to archive log file: {}", logFile, e);
    }
}

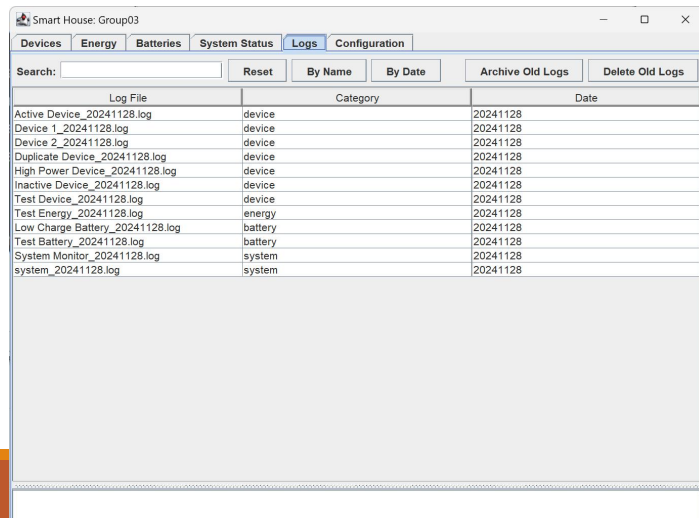
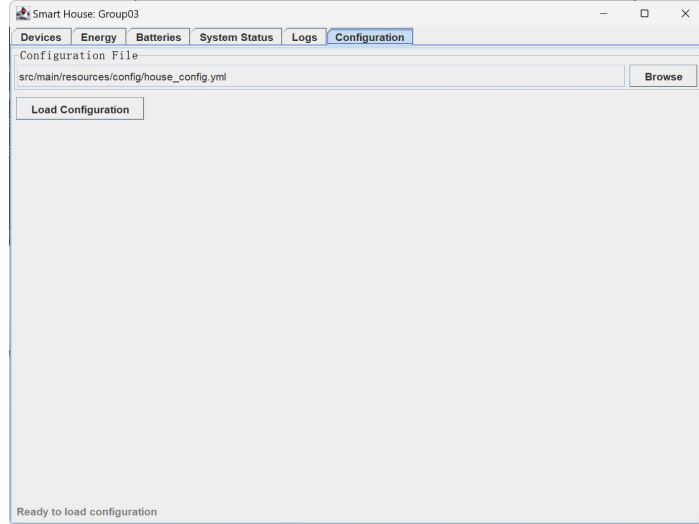
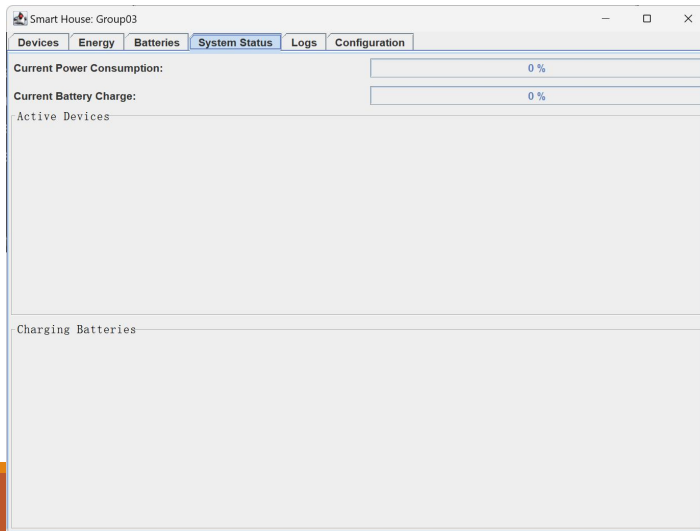
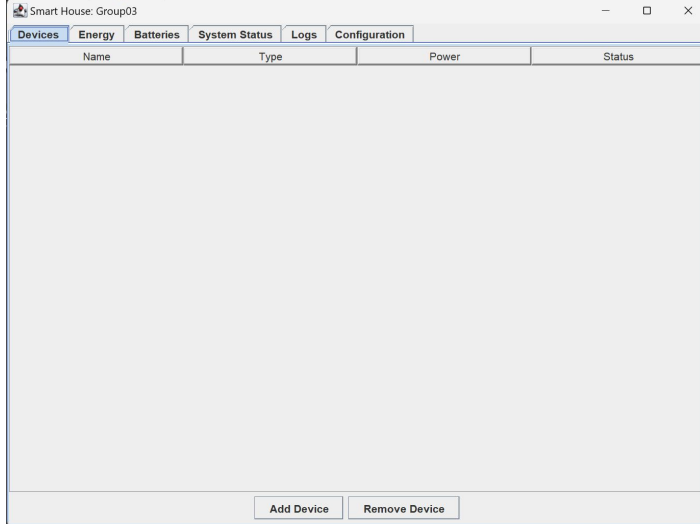
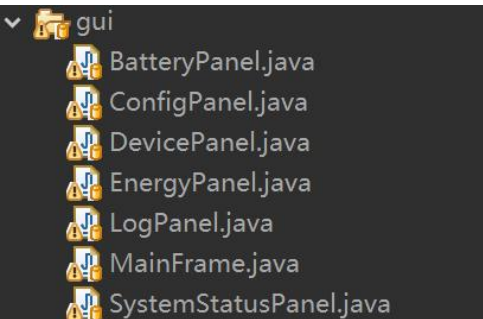
public void deleteLogFile(Path logFile) {
    try {
        Files.delete(logFile);
        Log.info("Deleted log file: {}", logFile);
    } catch (IOException e) {
        Log.error("Failed to delete log file: {}", logFile, e);
    }
}
```

<div> Users > yanal > git > capstone_project > logs > </div>			
<div> <div>Sort</div> <div>View</div> <div>...</div> </div>			
Name	Date modified	Type	Size
archive	24/11/2024 21:00	File folder	
battery	24/11/2024 21:00	File folder	
device	26/11/2024 12:03	File folder	
energy	26/11/2024 12:04	File folder	
system	26/11/2024 12:03	File folder	

<div> Users > yanal > git > capstone_project > logs > device </div>			
<div> <div>Sort</div> <div>View</div> <div>...</div> </div>			
Name	Date modified	Type	Size
temp1_20241124	24/11/2024 22:18	Text Document	1 KB
temp1_20241126	26/11/2024 12:03	Text Document	1 KB
yanal_20241124	24/11/2024 22:06	Text Document	1 KB

<div> Users > yanal > git > capstone_project > logs > energy </div>			
<div> <div>Sort</div> <div>View</div> <div>...</div> </div>			
Name	Date modified	Type	Size
charger_20241124	24/11/2024 22:04	Text Document	1 KB
energy1_20241126	26/11/2024 12:04	Text Document	1 KB
wind_20241124	24/11/2024 22:02	Text Document	1 KB

GUI



GUI

```
public MainFrame() {
    initializeFrame();
    createComponents();
    addComponents();

    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void initializeFrame() {
    setTitle("Smart House: Group03");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(800, 600);
    setLocationRelativeTo(null);
    setMinimumSize(new Dimension(800, 600));
}

private void createComponents() {
    tabbedPane = new JTabbedPane();

    devicePanel = new DevicePanel(deviceManager);
    energyPanel = new EnergyPanel(energyManager);
    batteryPanel = new BatteryPanel(energyManager);
    systemStatusPanel = new SystemStatusPanel(deviceManager, energyManager);
    logPanel = new LogPanel(logManager);
    configPanel = new ConfigPanel(this);

    tabbedPane.addTab("Devices", devicePanel);
    tabbedPane.addTab("Energy", energyPanel);
    tabbedPane.addTab("Batteries", batteryPanel);
    tabbedPane.addTab("System Status", systemStatusPanel);
    tabbedPane.addTab("Logs", logPanel);
    tabbedPane.addTab("Configuration", configPanel);
}

private void addComponents() {
    add(tabbedPane);
}

public void refreshAllPanels() {
    devicePanel.updateDeviceTable();
    energyPanel.updateEnergyTable();
    batteryPanel.updateBatteryTable();
    systemStatusPanel.updateStatus();
}
```


Concurrency(Example: Charging)

```
public void startCharging(String batteryId) {  
    Battery battery = getBatteryById(batteryId);  
  
    if (battery.isCharging()) {  
        log.info("Battery {} is already charging", battery.getName());  
        return;  
    }  
  
    List<Energy> activeEnergies = getEnergiesByState(true);  
  
    if (activeEnergies.isEmpty()) {  
        log.info("No active energy sources found to charge the battery {}", battery.getName());  
        return;  
    }  
  
    battery.setCharging(true);  
    CompletableFuture.runAsync(() -> manageChargingTasks(battery), executorService);  
}
```

```
private void manageChargingTasks(Battery battery) {  
    List<Energy> activeEnergies = getEnergiesByState(true);  
    List<CompletableFuture<Void>> tasks = activeEnergies.stream()  
        .map(energy -> CompletableFuture.runAsync(() -> chargeFromEnergy(battery, energy), executorService))  
        .collect(Collectors.toList());  
  
    try {  
        while (battery.isCharging()) {  
            List<Energy> currentActiveEnergies = getEnergiesByState(true);  
  
            if (!currentActiveEnergies.isEmpty() && currentActiveEnergies.size() != activeEnergies.size()) {  
                Energy newActiveEnergy = currentActiveEnergies.get(currentActiveEnergies.size() - 1);  
                tasks.add(CompletableFuture.runAsync(() -> chargeFromEnergy(battery, newActiveEnergy), executorService));  
            }  
  
            if (tasks.stream().allMatch(CompletableFuture::isDone)) {  
                battery.setCharging(false);  
                break;  
            }  
  
            Thread.sleep(2000);  
        }  
    } catch (InterruptedException e) {  
        Thread.currentThread().interrupt();  
    } finally {  
        tasks.forEach(task -> task.cancel(true));  
        battery.setCharging(false);  
    }  
}
```

Concurrency(Example: Charging)

```
private void chargeFromEnergy(Battery battery, Energy energy) {
    String id = UUID.randomUUID().toString();

    try {
        while (battery.isCharging()) {
            synchronized (battery) {
                double availablePower = energy.getOutput();
                double batteryDeficit = battery.getCapacity() - battery.getCurrentCharge();
                double deviceConsumption = deviceManager.getCurrentTotalConsumption();

                double chargePower = Math.min(battery.getMaxChargeRate(), availablePower);

                if (batteryDeficit <= 0 && deviceConsumption <= 0) {
                    break;
                }

                double netCharge = chargePower - deviceConsumption;

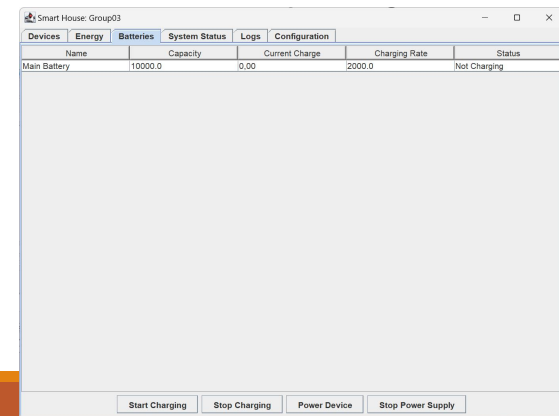
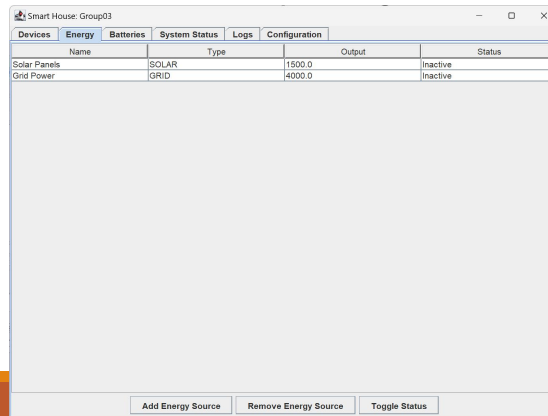
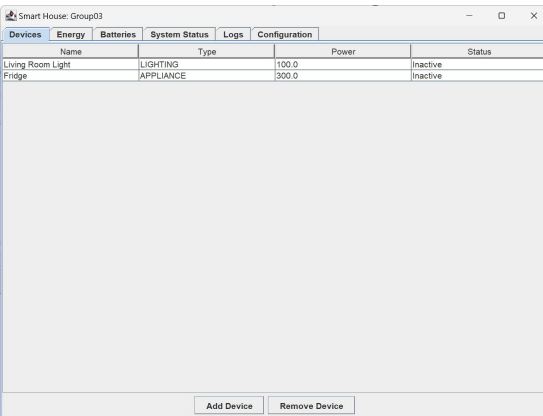
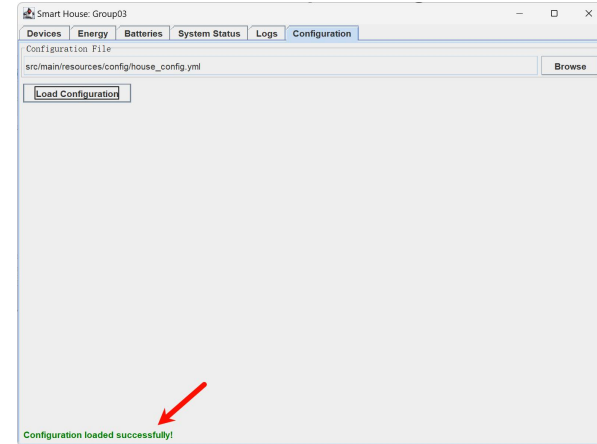
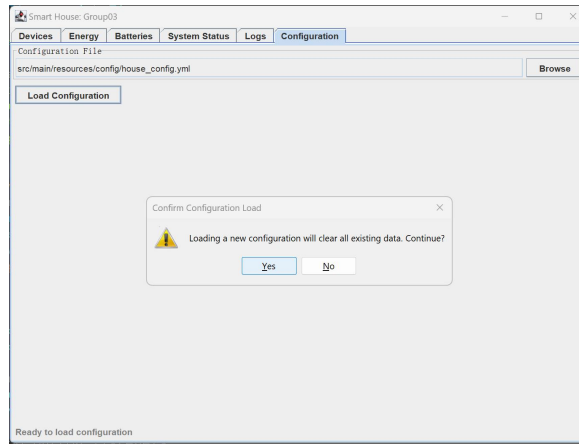
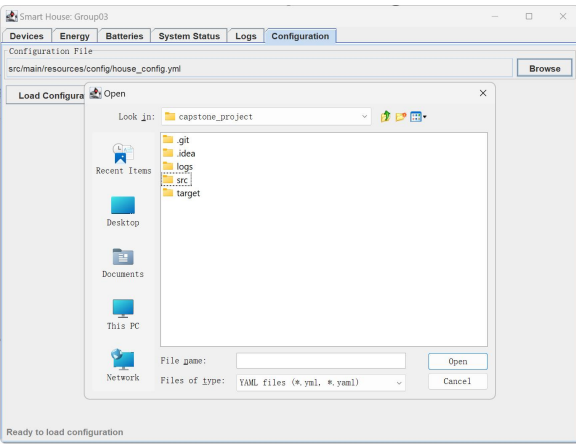
                if (netCharge > 0) {
                    double chargeAmount = Math.min(netCharge, batteryDeficit);
                    battery.setCurrentCharge(battery.getCurrentCharge() + chargeAmount);
                    LoggerHelper.logChargingEvent(logManager, battery.getName(), energy.getName(), chargeAmount);
                } else {
                    battery.setCurrentCharge(Math.max(0, battery.getCurrentCharge() + netCharge));
                    LoggerHelper.logChargingEvent(logManager, battery.getName(), energy.getName() + id, netCharge);
                }
            }

            Thread.sleep(2000);
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}

public void stopCharging(String batteryId) {
    Battery battery = getBatteryById(batteryId);

    battery.setCharging(false);
    LoggerHelper.logBatteryEvent(logManager, "Stopped charging", battery.getName());
}
```

Initial Configuration



Unit Test

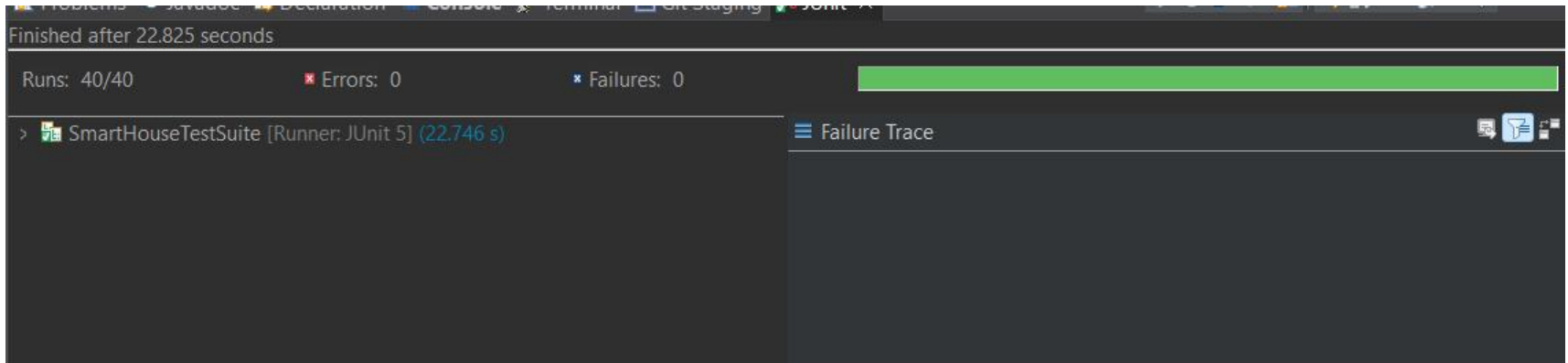
SmartHouseTestSuite.java ×

```
1 package de.fhdo;  
2  
3  
4 import org.junit.platform.suite.api.SelectClasses;  
12  
13 @Suite  
14 @SelectClasses({  
15     HouseConfigTest.class,  
16     LogManagerTest.class,  
17     DeviceManagerTest.class,  
18     EnergyManagerTest.class,  
19     SystemMonitorTest.class  
20 })  
21 public class SmartHouseTestSuite {  
22 }
```

Unit Test of the System

```
HouseConfigTest.java x
7 public class HouseConfigTest {
8
9     @Test
10     void testLoadFromFile() throws IOException {
11         HouseConfig config = HouseConfig.loadFromFile("src/test/resources/house_config.yml");
12
13         assertNotNull(config);
14         assertNotNull(config.getDevices());
15         assertNotNull(config.getEnergies());
16         assertNotNull(config.getBatteries());
17
18         assertEquals(1, config.getDevices().size());
19         HouseConfig.DeviceConfig firstDevice = config.getDevices().get(0);
20         assertEquals("Living Room Lights", firstDevice.getName());
21         assertEquals("LIGHTING", firstDevice.getType());
22         assertEquals(100.0, firstDevice.getPower());
23
24         assertEquals(1, config.getEnergies().size());
25         HouseConfig.EnergyConfig firstEnergy = config.getEnergies().get(0);
26         assertEquals("Solar Panels", firstEnergy.getName());
27         assertEquals("SOLAR", firstEnergy.getType());
28         assertEquals(5000.0, firstEnergy.getOutput());
29
30         assertEquals(1, config.getBatteries().size());
31         HouseConfig.BatteryConfig battery = config.getBatteries().get(0);
32         assertEquals("Main Battery", battery.getName());
33         assertEquals(10000.0, battery.getCapacity());
34         assertEquals(2000.0, battery.getMaxChargeRate());
35         assertEquals(2000.0, battery.getMaxDischargeRate());
36     }
37
38     @Test
39     void testLoadFromNonExistentFile() {
40         assertThrows(IOException.class, () ->
41             HouseConfig.loadFromFile("non_existent_file.yml")
42         );
43     }
44 }
```

Unit Test Results



Thank you