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.

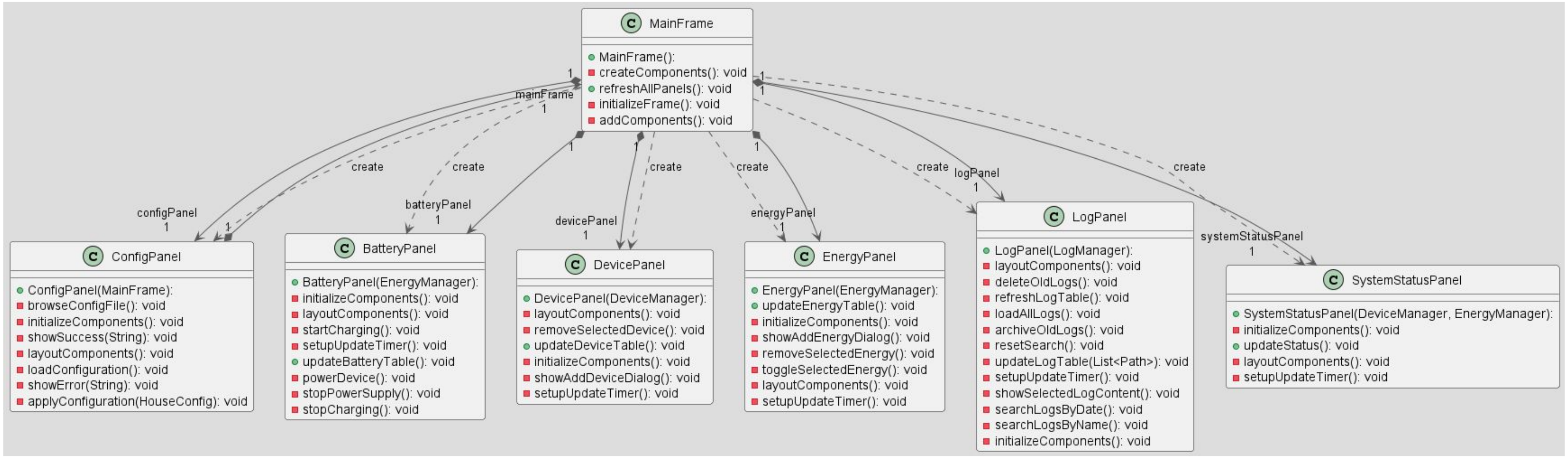
Graphic User Interfaces

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

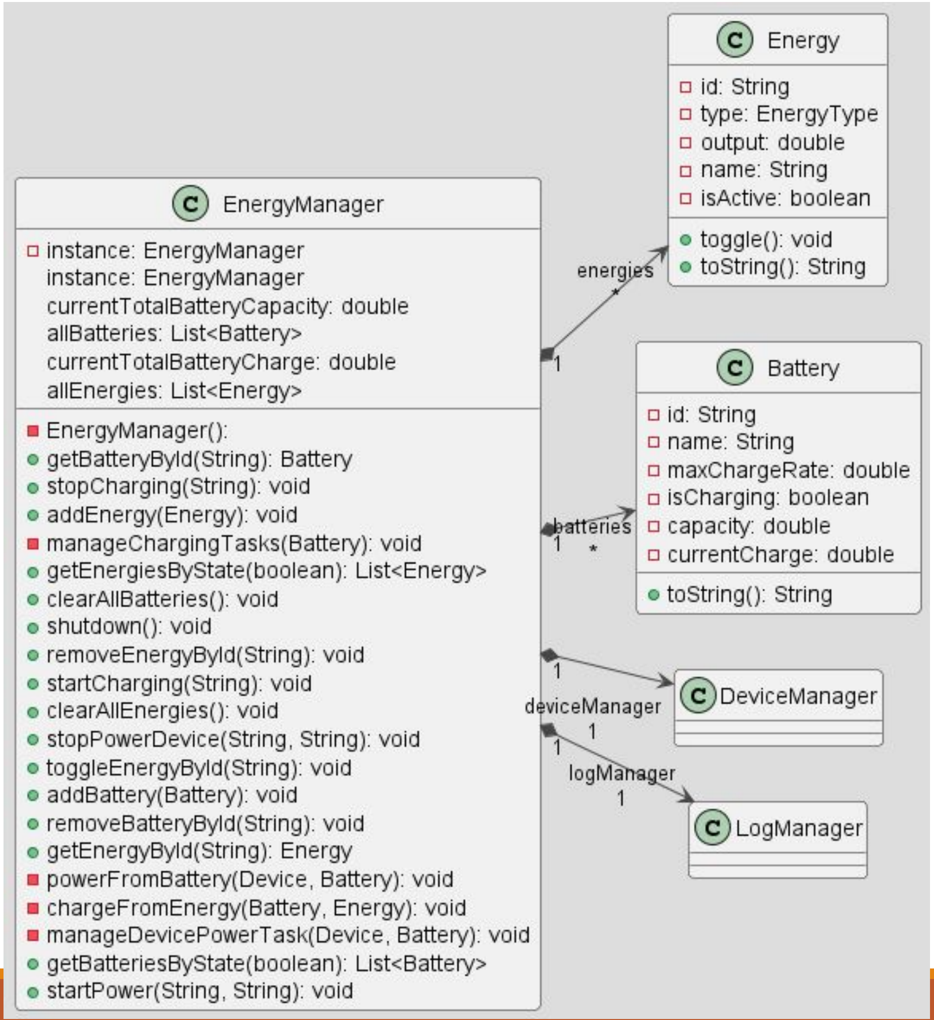
Configuration Loading

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

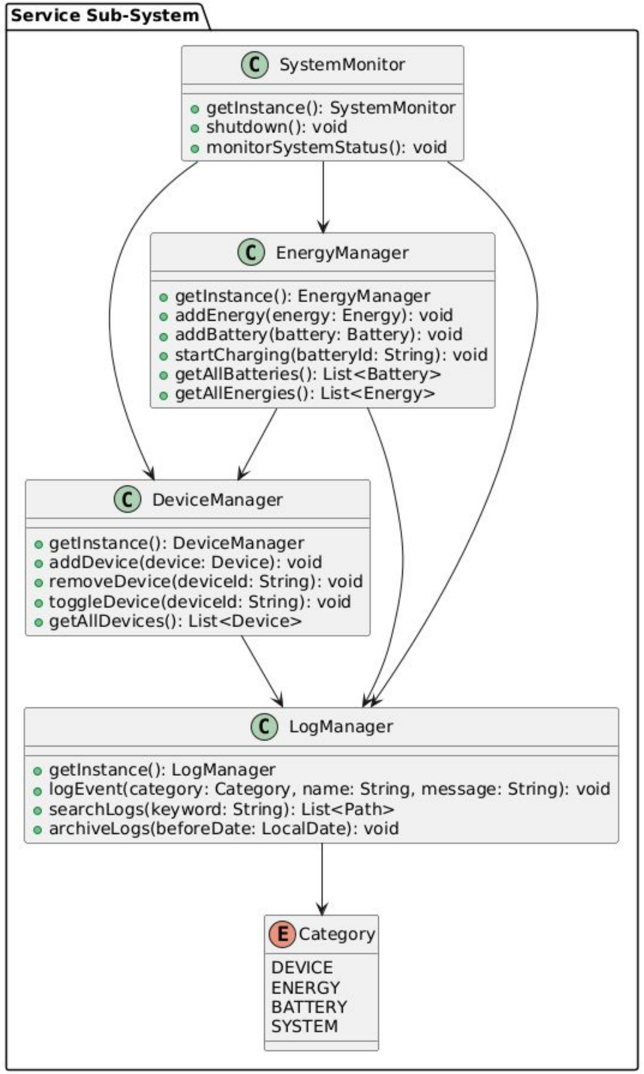
Graphic User Interfaces


MainFrame
+ MainFrame():
- createComponents(): void
+ refreshAllPanels(): void
- initializeFrame(): void
- addComponents(): void
ConfigPanel
+ ConfigPanel(MainFrame):
- browseConfigFile(): void
- initializeComponents(): void
- showSuccess(String): void
- layoutComponents(): void
- loadConfiguration(): void
- showError(String): void
- applyConfiguration(HouseConfig): void
BatteryPanel
+ BatteryPanel(EnergyManager):
- initializeComponents(): void
- layoutComponents(): void
- startCharging(): void
- setupUpdateTimer(): void
+ updateBatteryTable(): void
- powerDevice(): void
- stopPowerSupply(): void
- stopCharging(): void
DevicePanel
+ DevicePanel(DeviceManager):
- layoutComponents(): void
- removeSelectedDevice(): void
+ updateDeviceTable(): void
- initializeComponents(): void
- showAddDeviceDialog(): void
- setupUpdateTimer(): void
EnergyPanel
+ EnergyPanel(EnergyManager):
+ updateEnergyTable(): void
- initializeComponents(): void
- showAddEnergyDialog(): void
- removeSelectedEnergy(): void
- toggleSelectedEnergy(): void
- layoutComponents(): void
- setupUpdateTimer(): void
LogPanel
+ LogPanel(LogManager):
- layoutComponents(): void
- deleteOldLogs(): void
- refreshLogTable(): void
- loadAllLogs(): void
- archiveOldLogs(): void
- resetSearch(): void
- updateLogTable(List<Path>): void
- setupUpdateTimer(): void
- showSelectedLogContent(): void
- searchLogsByDate(): void
- searchLogsByName(): void
- initializeComponents(): void
SystemStatusPanel
+ SystemStatusPanel(DeviceManager, EnergyManager):
- initializeComponents(): void
+ updateStatus(): void
- layoutComponents(): void
- setupUpdateTimer(): void
mainFrame 1
configPanel 1
batteryPanel 1
devicePanel 1
energyPanel 1
logPanel 1
systemStatusPanel 1
create
create
create
create
create
create
1
1
1
1
1
1

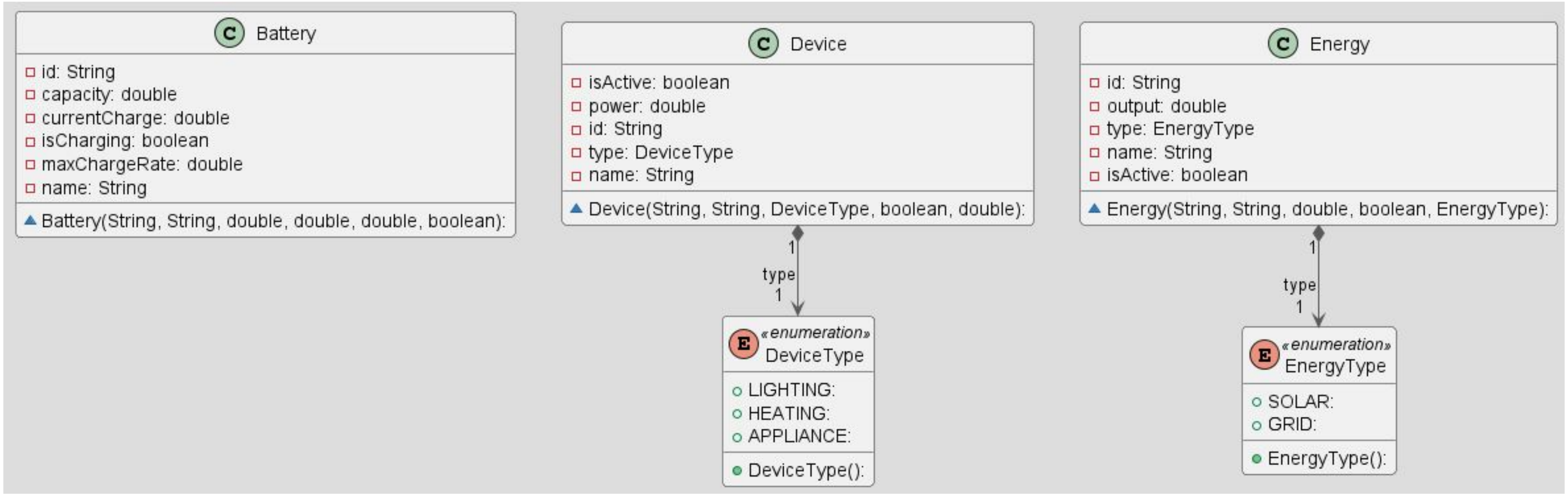
Energy Management


Energy
id: String
type: EnergyType
output: double
name: String
isActive: boolean
toggle(): void
toString(): String
EnergyManager
instance: EnergyManager
instance: EnergyManager
currentTotalBatteryCapacity: double
allBatteries: List<Battery>
currentTotalBatteryCharge: double
allEnergies: List<Energy>
EnergyManager():
getBatteryById(String): Battery
stopCharging(String): void
addEnergy(Energy): void
manageChargingTasks(Battery): void
getEnergiesByState(boolean): List<Energy>
clearAllBatteries(): void
shutdown(): void
removeEnergyById(String): void
startCharging(String): void
clearAllEnergies(): void
stopPowerDevice(String, String): void
toggleEnergyById(String): void
addBattery(Battery): void
removeBatteryById(String): void
getEnergyById(String): Energy
powerFromBattery(Device, Battery): void
chargeFromEnergy(Battery, Energy): void
manageDevicePowerTask(Device, Battery): void
getBatteriesByState(boolean): List<Battery>
startPower(String, String): void
energies
*
1
Battery
id: String
name: String
maxChargeRate: double
isCharging: boolean
capacity: double
currentCharge: double
toString(): String
batteries
1
*
1
deviceManager
1
DeviceManager
1
logManager
1
LogManager

Service

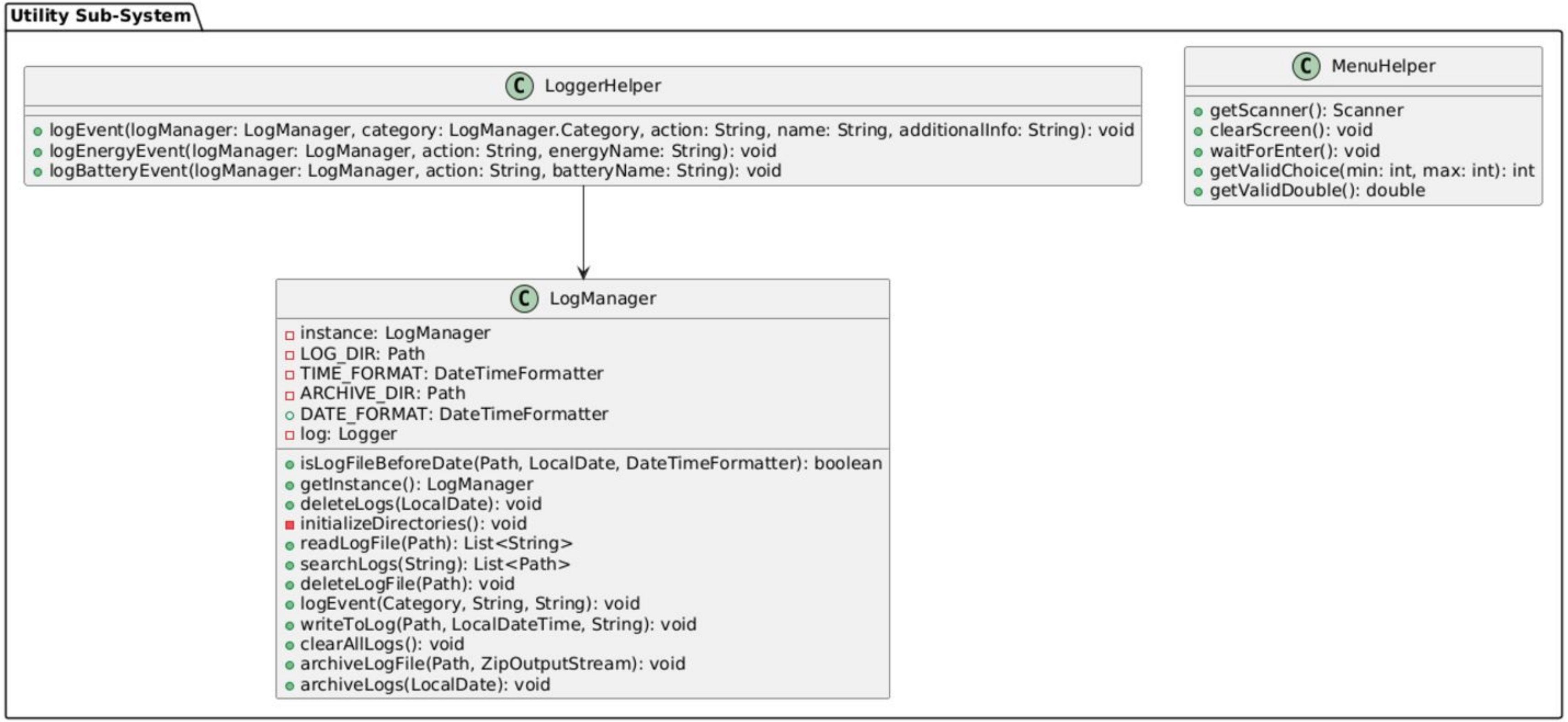

Service Sub-System
SystemMonitor
+ getInstance(): SystemMonitor
+ shutdown(): void
+ monitorSystemStatus(): void
EnergyManager
+ getInstance(): EnergyManager
+ addEnergy(energy: Energy): void
+ addBattery(battery: Battery): void
+ startCharging(batteryId: String): void
+ getAllBatteries(): List<Battery>
+ getAllEnergies(): List<Energy>
DeviceManager
+ getInstance(): DeviceManager
+ addDevice(device: Device): void
+ removeDevice(deviceId: String): void
+ toggleDevice(deviceId: String): void
+ getAllDevices(): List<Device>
LogManager
+ getInstance(): LogManager
+ logEvent(category: Category, name: String, message: String): void
+ searchLogs(keyword: String): List<Path>
+ archiveLogs(beforeDate: LocalDate): void
Category
DEVICE
ENERGY
BATTERY
SYSTEM

Model



Battery
id: String
capacity: double
currentCharge: double
isCharging: boolean
maxChargeRate: double
name: String
Battery(String, String, double, double, double, boolean):
Device
isActive: boolean
power: double
id: String
type: DeviceType
name: String
Device(String, String, DeviceType, boolean, double):
Energy
id: String
output: double
type: EnergyType
name: String
isActive: boolean
Energy(String, String, double, boolean, EnergyType):
1
type
1
«enumeration»
DeviceType
LIGHTING:
HEATING:
APPLIANCE:
DeviceType():
1
type
1
«enumeration»
EnergyType
SOLAR:
GRID:
EnergyType():

Utility



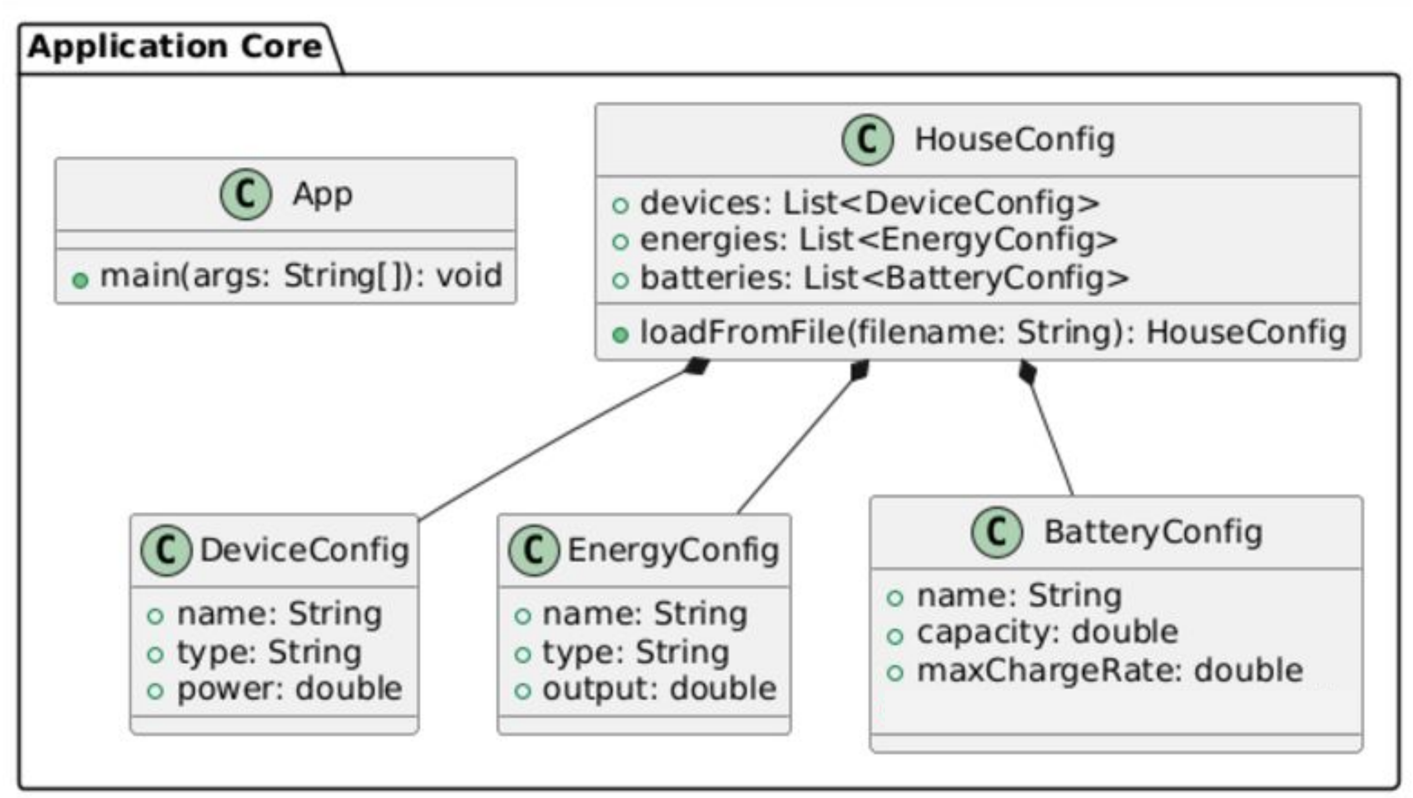
LoggerHelper
- logEvent(logManager: LogManager, category: LogManager.Category, action: String, name: String, additionalInfo: String): void
- logEnergyEvent(logManager: LogManager, action: String, energyName: String): void
- logBatteryEvent(logManager: LogManager, action: String, batteryName: String):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

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

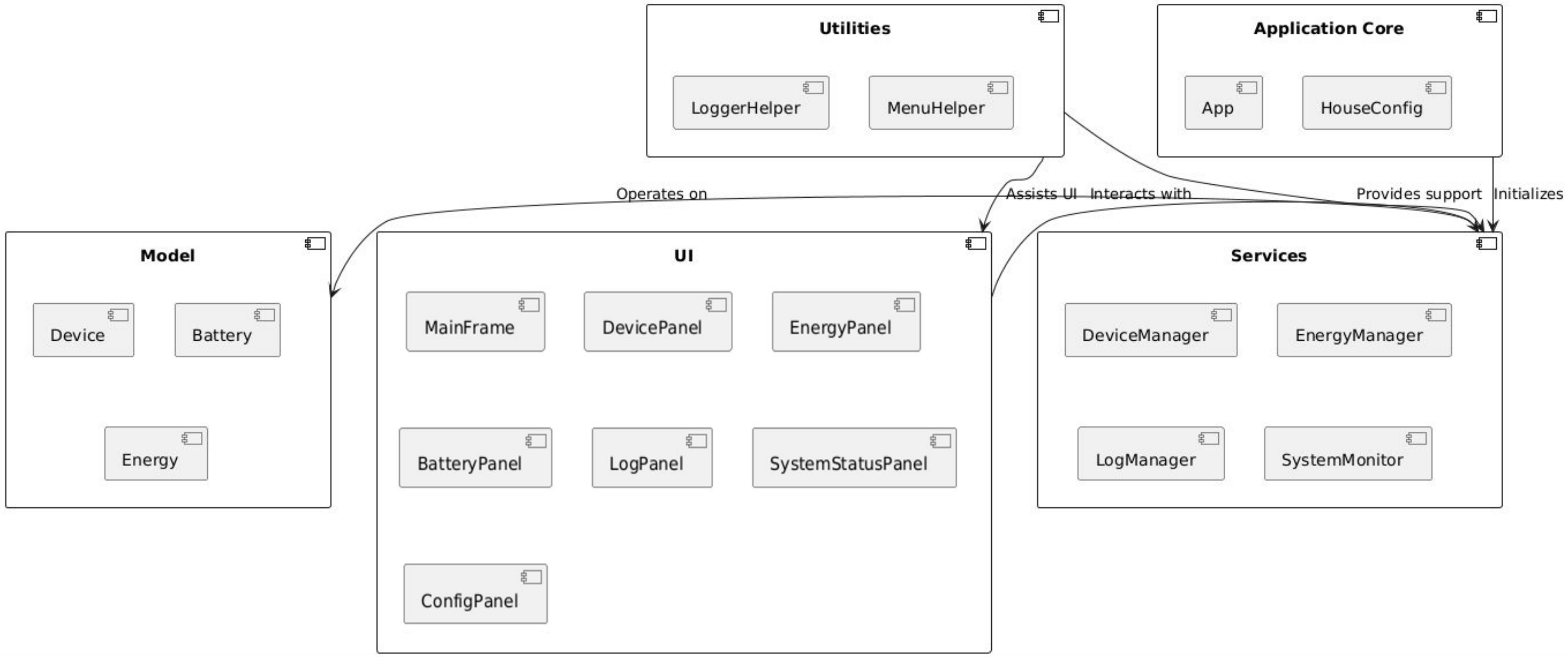
LogManager
- instance: LogManager
- LOG_DIR: Path
- TIME_FORMAT: DateTimeFormatter
- ARCHIVE_DIR: Path
- DATE_FORMAT: DateTimeFormatter
- log: Logger

- 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

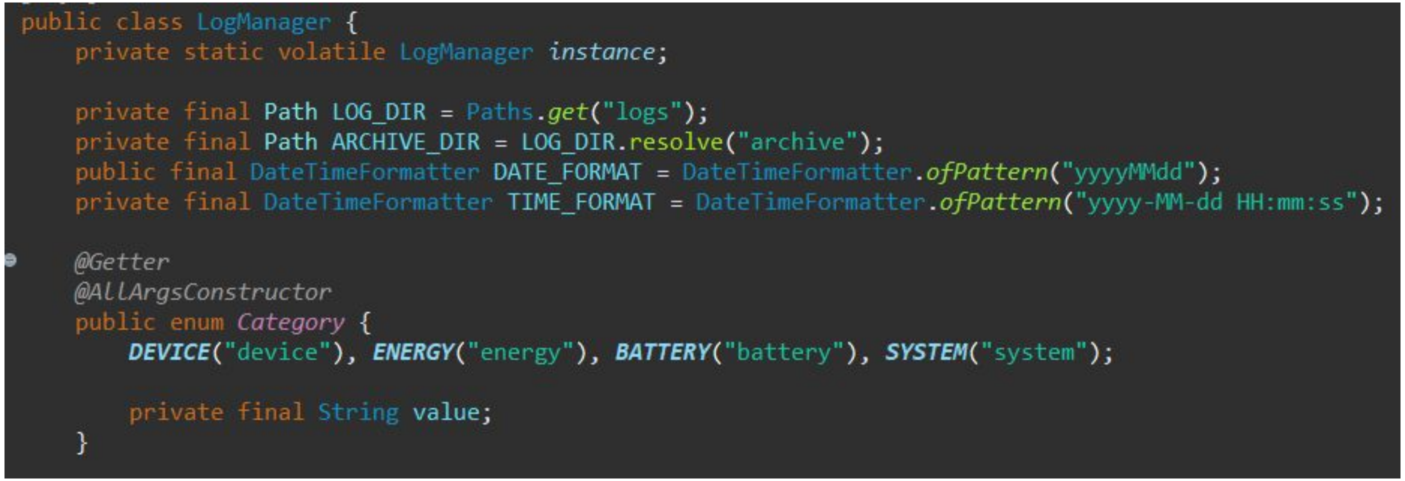
Application Core



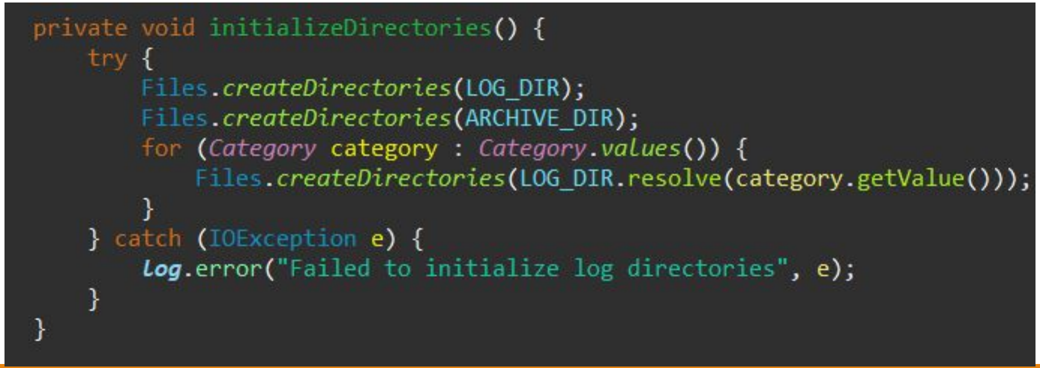
Component Diagram



Management I/O in the System

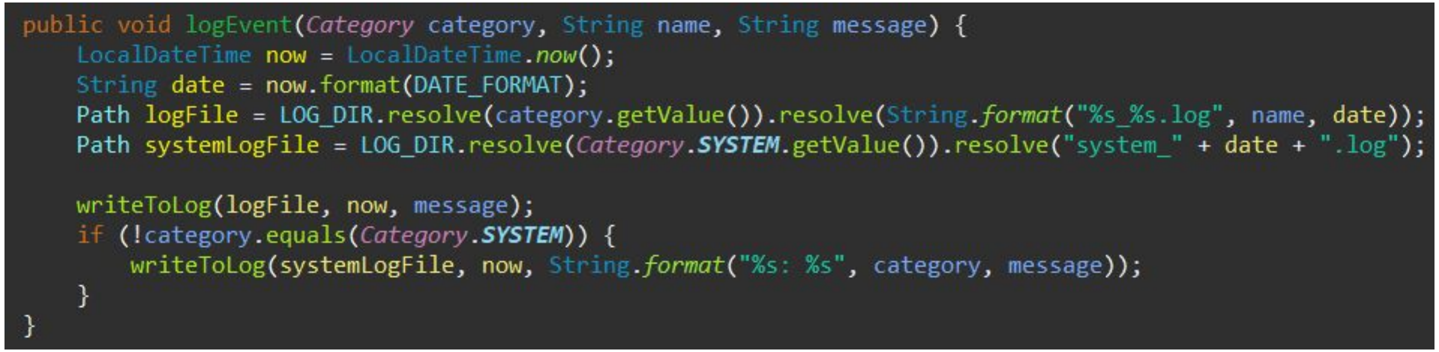


```
public class LogManager {
    private static volatile LogManager instance;

    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;
    }
```

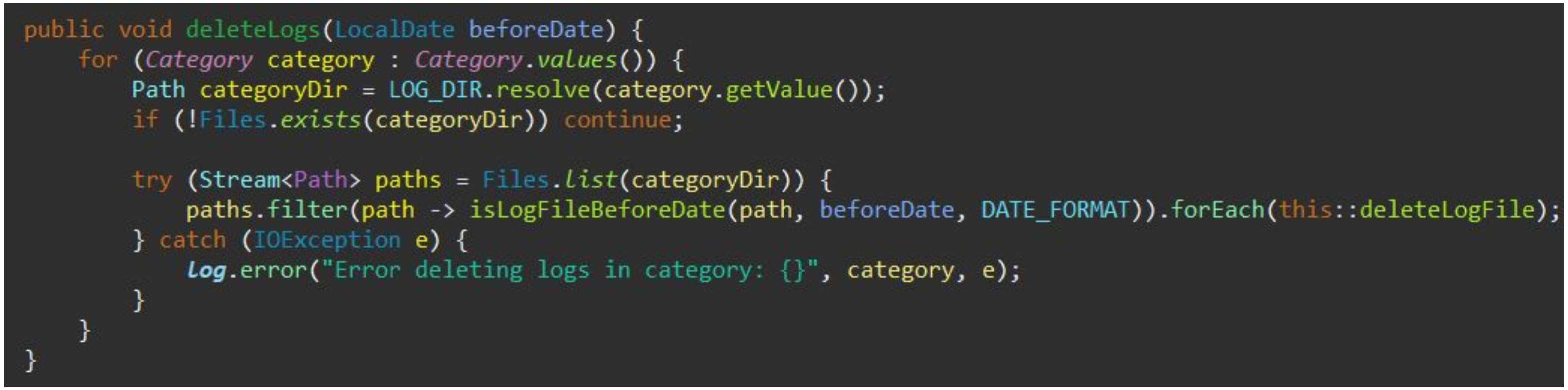


```
private void initializeDirectories() {
    try {
        Files.createDirectories(LOG_DIR);
        Files.createDirectories(ARCHIVE_DIR);
        for (Category category : Category.values()) {
            Files.createDirectories(LOG_DIR.resolve(category.getValue()));
        }
    } catch (IOException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

Events of I/O in the System

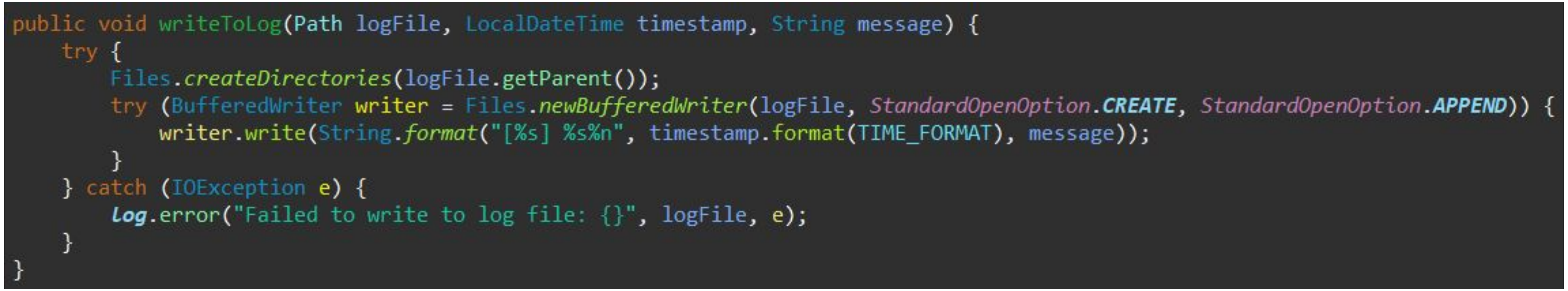


```java
public void logEvent(Category category, String name, String message) {
    LocalDateTime now = LocalDateTime.now();
    String date = now.format(DATE_FORMAT);
    Path logFile = LOG_DIR.resolve(category.getValue()).resolve(String.format("%s_%s.log", name, date));
    Path systemLogFile = LOG_DIR.resolve(Category.SYSTEM.getValue()).resolve("system_"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

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

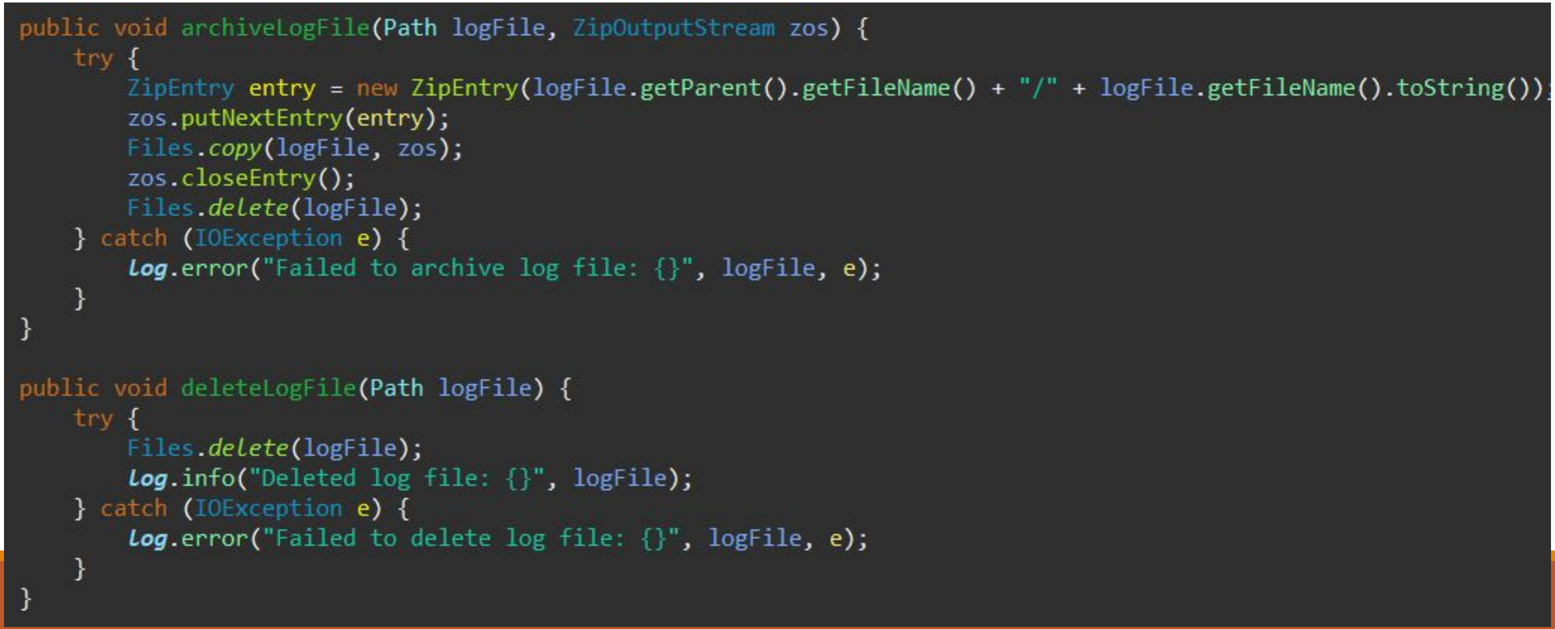


```java
public void deleteLogs(LocalDate beforeDate) {
    for (Category category : Category.values()) {
        Path categoryDir = LOG_DIR.resolve(category.getValue());
        if (!Files.exists(categoryDir)) continue;

        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);
        }
    }
}
```

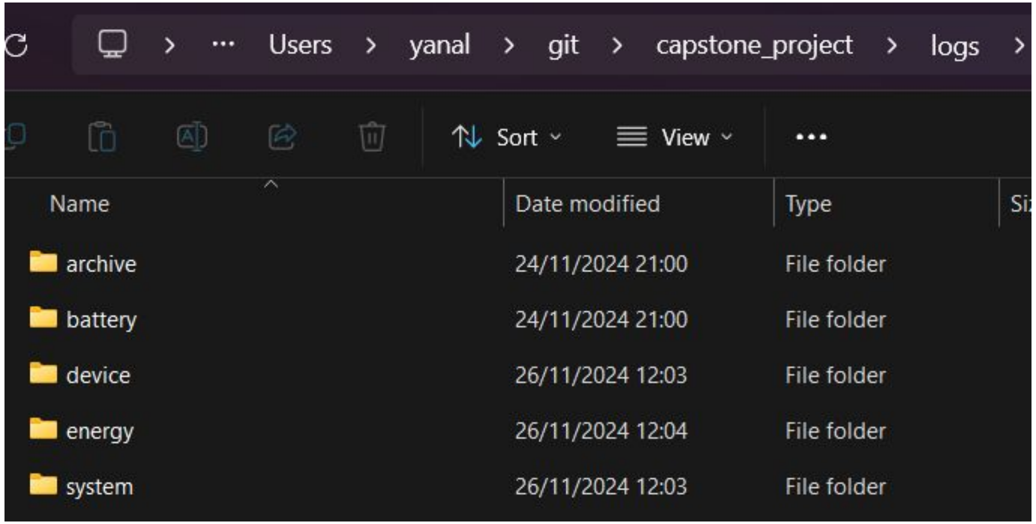
Managing Files of I/O in the System



```java
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);
    }
}
```

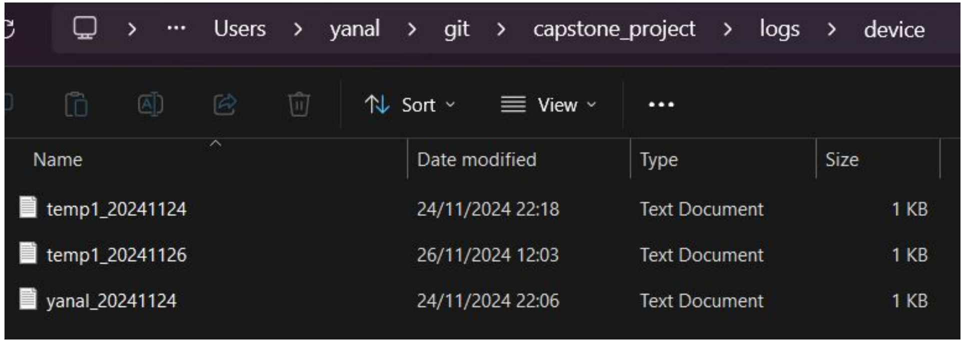


```java
public void archiveLogFile(Path logFile, ZipOutputStream zos) {
    try {
        ZipEntry entry = new ZipEntry(logFile.getParent().getFileName() + "/" + logFile.getFileName().toString());
        zos.putNextEntry(entry);
        Files.copy(logFile, zos);
        zos.closeEntry();
        Files.delete(logFile);
    } 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);
    }
}
```



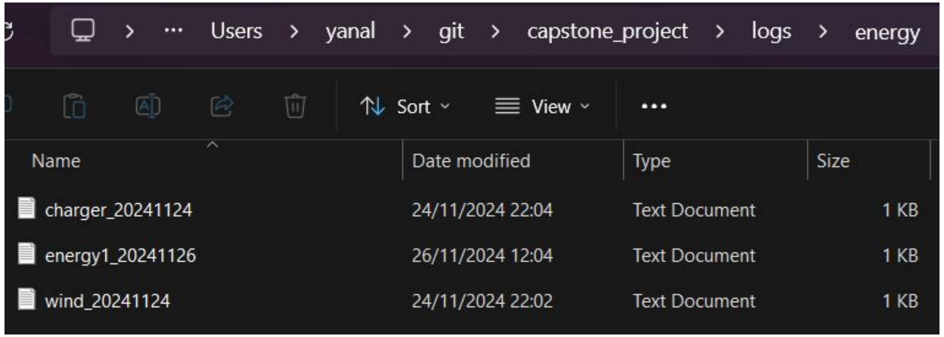
Users › yanal › git › capstone_project › logs ›

Name	Date modified	Type	Si
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	



Users › yanal › git › capstone_project › logs › device

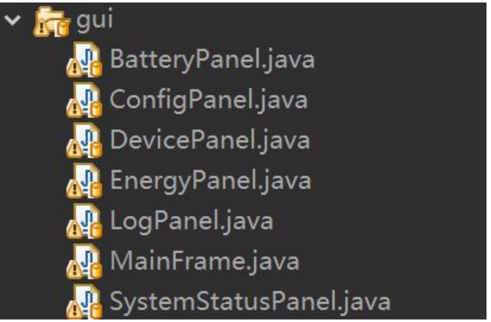
Name	Date modified	Type	Size
temp1_20241124	24/11/2024 22:18	Text Document	1 KB
temp1_20241126	26/11/2024 12:03	Text Document	1 KB
yanal_20241124	24/11/2024 22:06	Text Document	1 KB



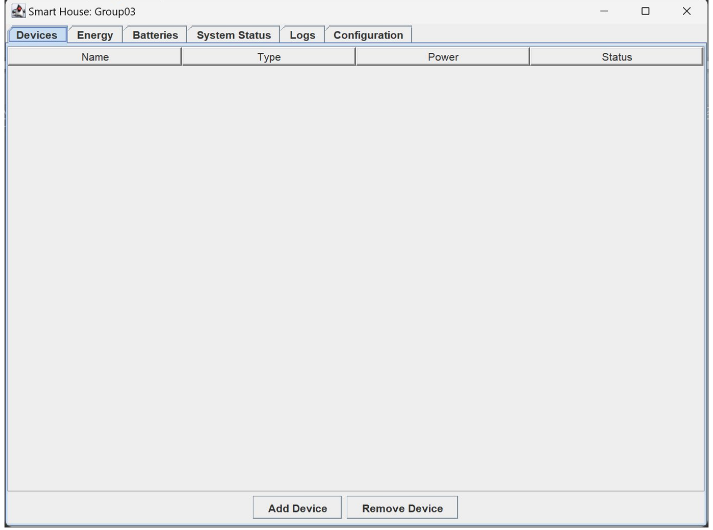
Users › yanal › git › capstone_project › logs › energy

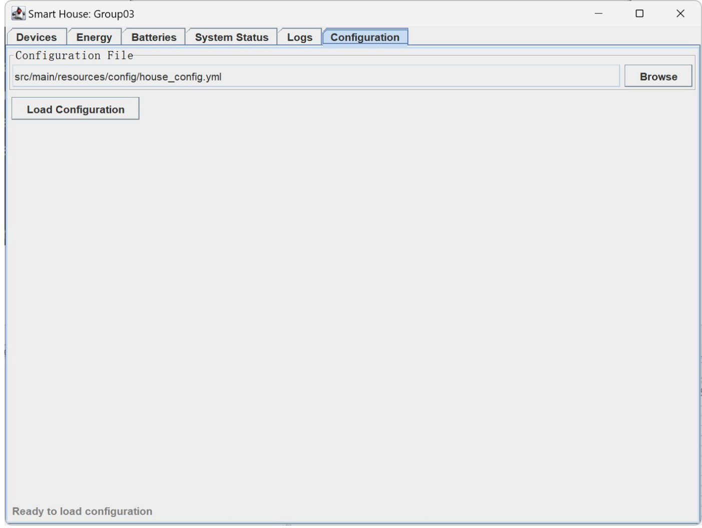
Name	Date modified	Type	Size
charger_20241124	24/11/2024 22:04	Text Document	1 KB
energy1_20241126	26/11/2024 12:04	Text Document	1 KB
wind_20241124	24/11/2024 22:02	Text Document	1 KB

GUI



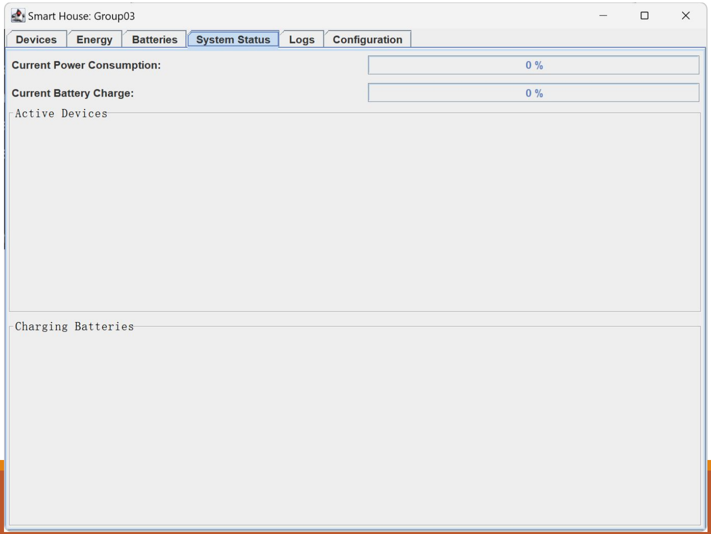
- gui
  - BatteryPanel.java
  - ConfigPanel.java
  - DevicePanel.java
  - EnergyPanel.java
  - LogPanel.java
  - MainFrame.java
  - SystemStatusPanel.java





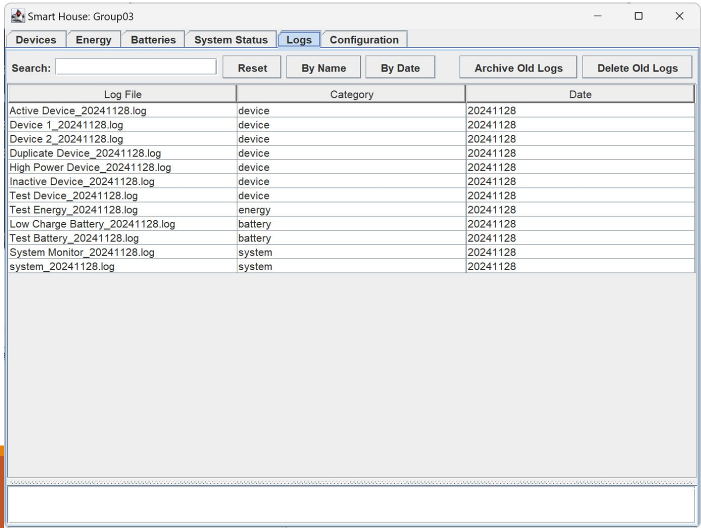
Configuration File: src/main/resources/config/house_config.yml

Ready to load configuration



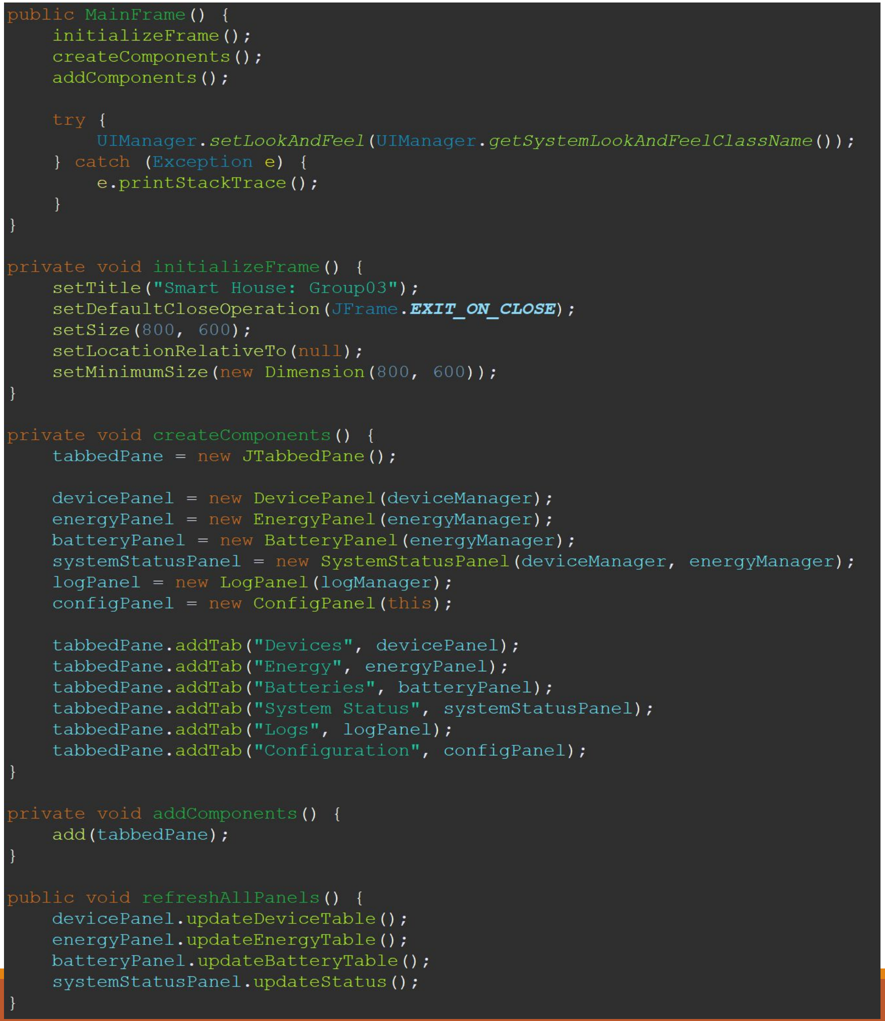
Current Power Consumption: 0 %

Current Battery Charge: 0 %



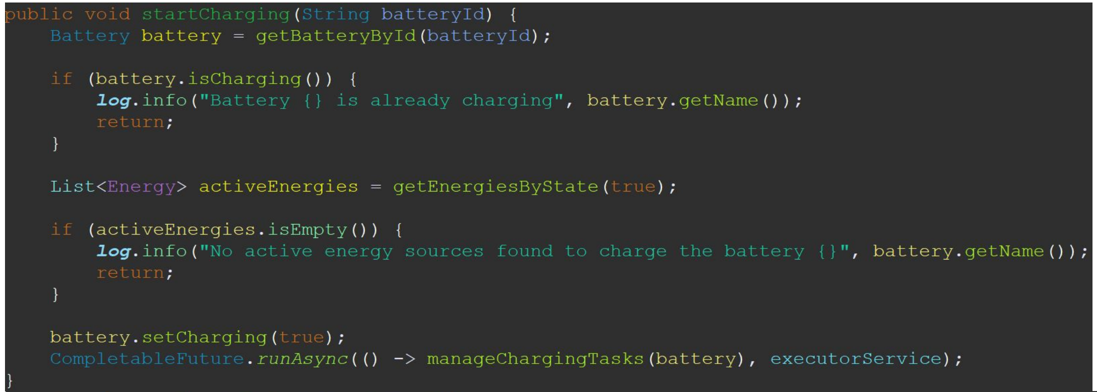
Log File	Category	Date
Active Device_20241128.log	device	20241128
Device 1_20241128.log	device	20241128
Device 2_20241128.log	device	20241128
Duplicate Device_20241128.log	device	20241128
High Power Device_20241128.log	device	20241128
Inactive Device_20241128.log	device	20241128
Test Device_20241128.log	device	20241128
Test Energy_20241128.log	energy	20241128
Low Charge Battery_20241128.log	battery	20241128
Test Battery_20241128.log	battery	20241128
System Monitor_20241128.log	system	20241128
system_20241128.log	system	20241128

GUI

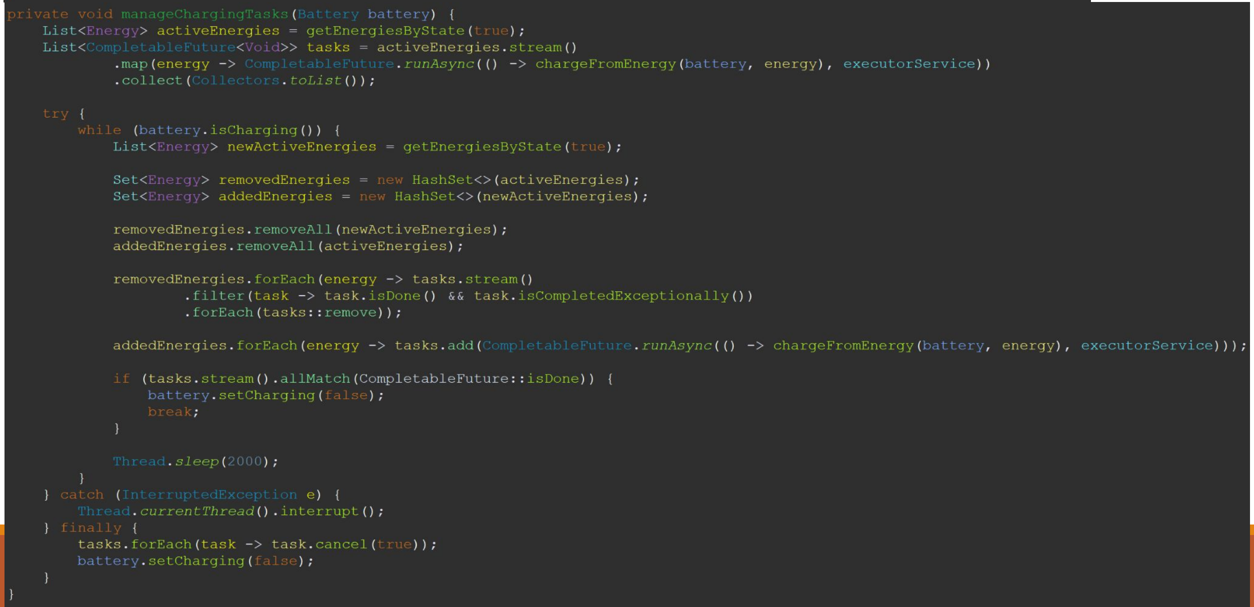


```java
public MainFrame() {
    initializeFrame();
    createComponents();
    addComponents();

    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)

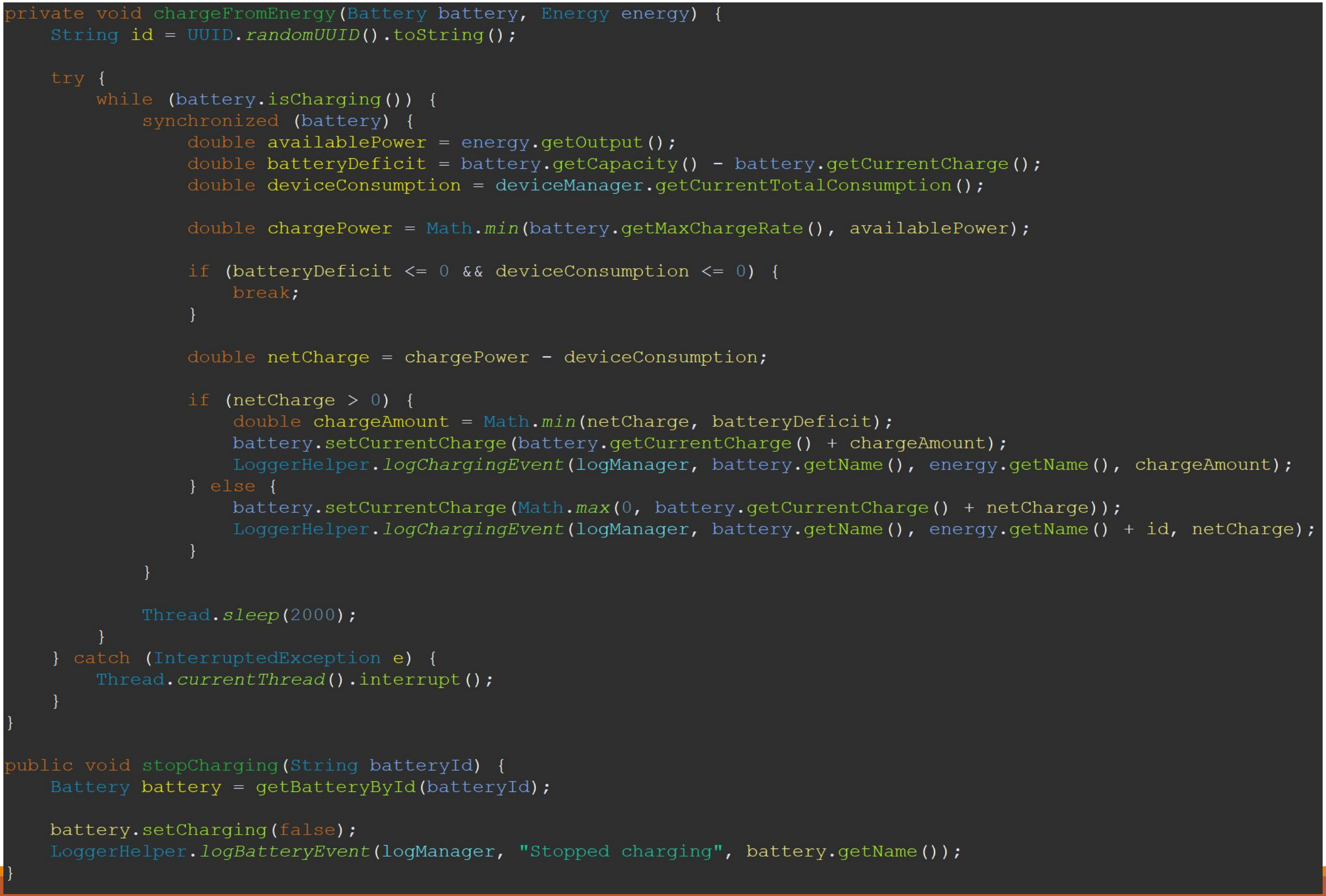


```java
public void startCharging(String batteryId) {
    Battery battery = getBatteryById(batteryId);

    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);
}
```



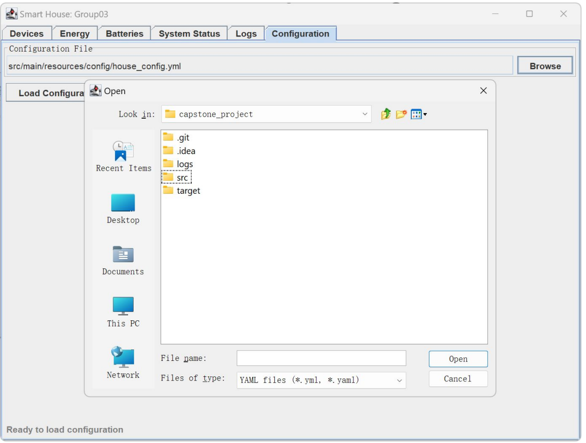
```java
private void manageChargingTasks(Battery battery) {
    List<Energy> activeEnergies = getEnergiesByState(true);
    List<CompletableFuture<Void>> tasks = activeEnergies.stream()
            .map(energy -> CompletableFuture.runAsync(() -> chargeFromEnergy(battery, energy), executorService))
            .collect(Collectors.toList());

    try {
        while (battery.isCharging()) {
            List<Energy> newActiveEnergies = getEnergiesByState(true);

            Set<Energy> removedEnergies = new HashSet<>(activeEnergies);
            Set<Energy> addedEnergies = new HashSet<>(newActiveEnergies);

            removedEnergies.removeAll(newActiveEnergies);
            addedEnergies.removeAll(activeEnergies);

            removedEnergies.forEach(energy -> tasks.stream()
                    .filter(task -> task.isDone() && task.isCompletedExceptionally())
                    .forEach(tasks::remove));

            addedEnergies.forEach(energy -> tasks.add(CompletableFuture.runAsync(() -> chargeFromEnergy(battery, energy), 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)



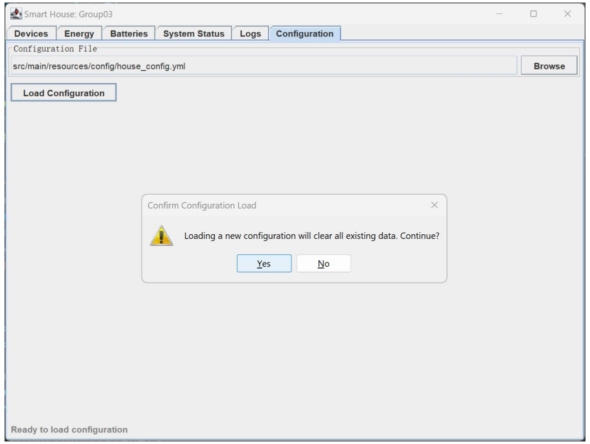
```java
private void chargeFromEnergy(Battery battery, Energy energy) {
    String id = UUID.randomUUID().toString();

    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



src/main/resources/config/house_config.yml

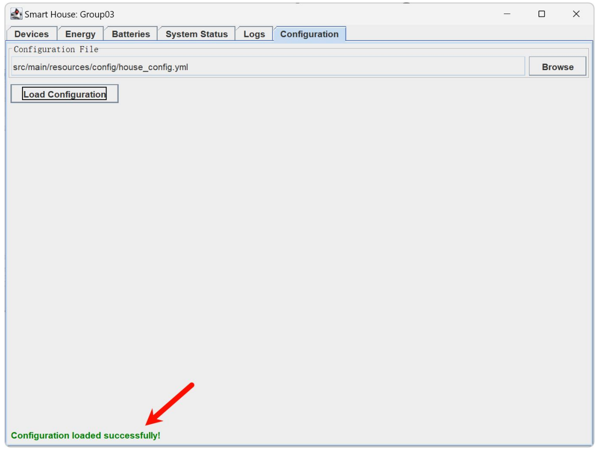
Ready to load configuration



src/main/resources/config/house_config.yml

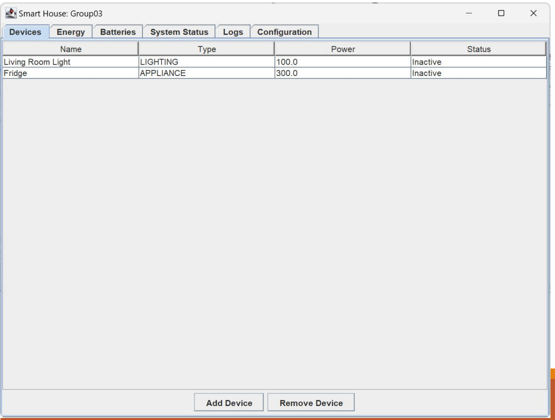
Loading a new configuration will clear all existing data. Continue?

Ready to load configuration

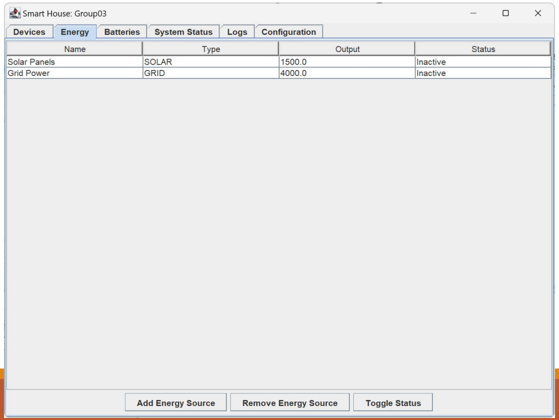


src/main/resources/config/house_config.yml

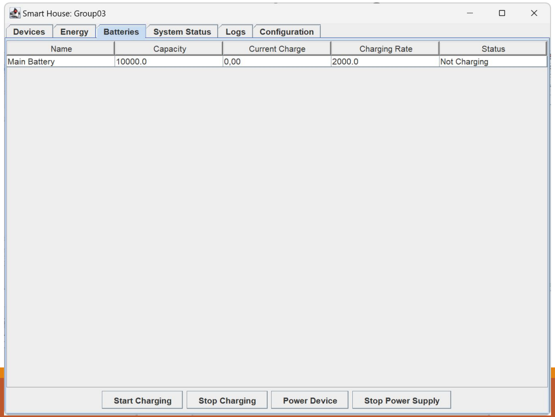
Configuration loaded successfully!



Name	Type	Power	Status
Living Room Light	LIGHTING	100.0	Inactive
Fridge	APPLIANCE	300.0	Inactive

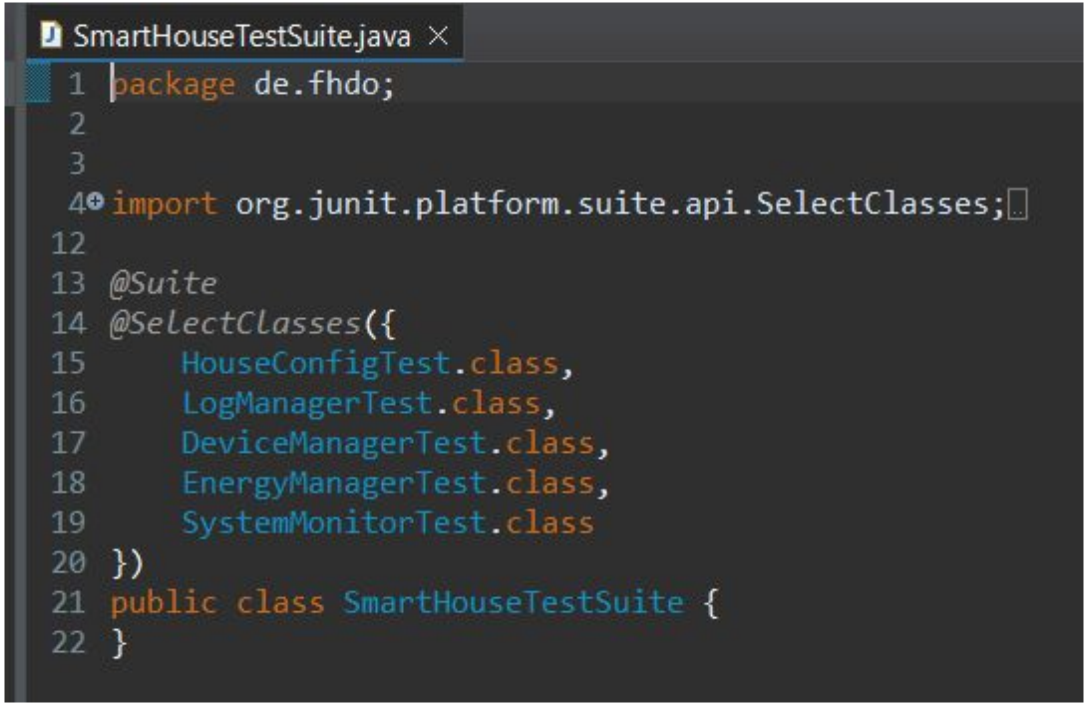


Name	Type	Output	Status
Solar Panels	SOLAR	1500.0	Inactive
Grid Power	GRID	4000.0	Inactive



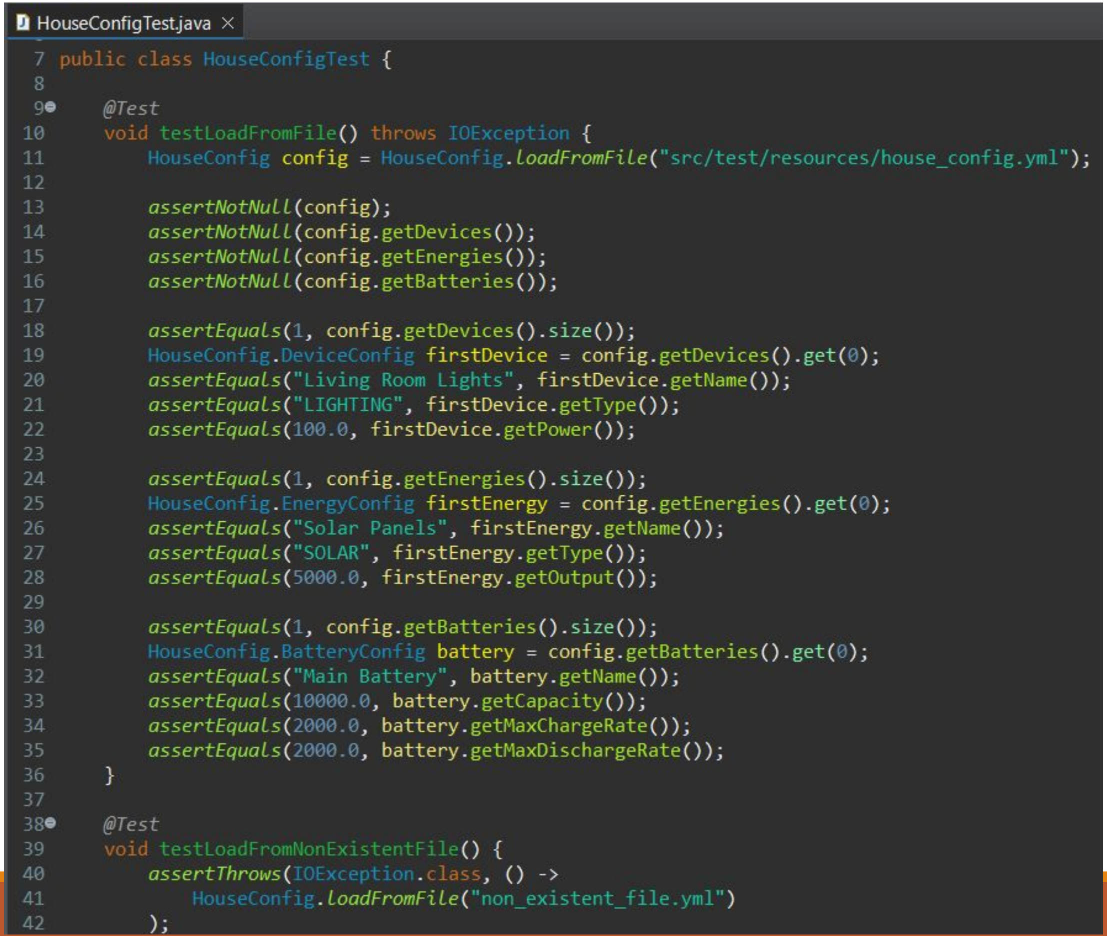
Name	Capacity	Current Charge	Charging Rate	Status
Main Battery	10000.0	0,00	2000.0	Not Charging

Unit Test


SmartHouseTestSuite.java

```java
package de.fhdo;


import org.junit.platform.suite.api.SelectClasses;

@Suite
@SelectClasses({
    HouseConfigTest.class,
    LogManagerTest.class,
    DeviceManagerTest.class,
    EnergyManagerTest.class,
    SystemMonitorTest.class
})
public class SmartHouseTestSuite {
}
```

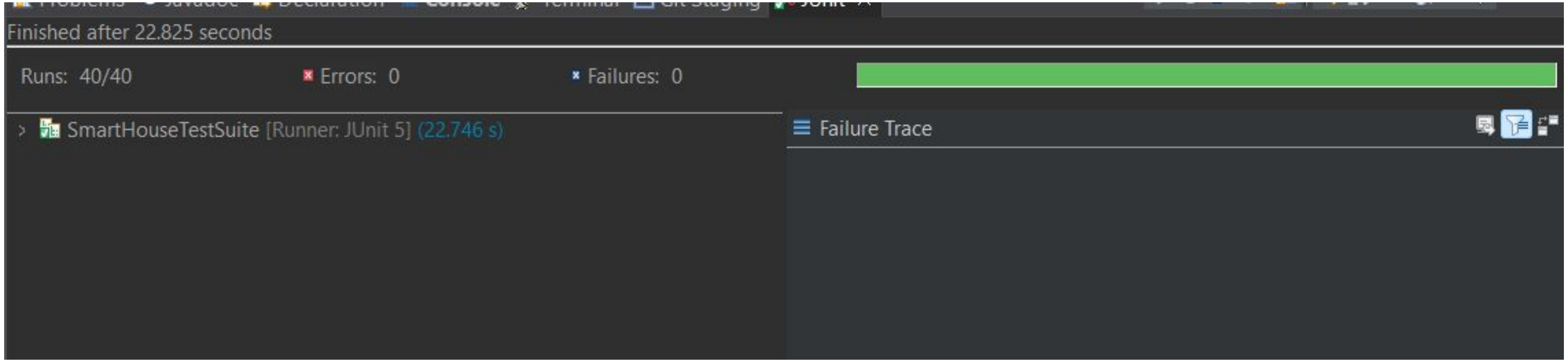
Unit Test of the System



HouseConfigTest.java

```java
public class HouseConfigTest {

    @Test
    void testLoadFromFile() throws IOException {
        HouseConfig config = HouseConfig.loadFromFile("src/test/resources/house_config.yml");

        assertNotNull(config);
        assertNotNull(config.getDevices());
        assertNotNull(config.getEnergies());
        assertNotNull(config.getBatteries());

        assertEquals(1, config.getDevices().size());
        HouseConfig.DeviceConfig firstDevice = config.getDevices().get(0);
        assertEquals("Living Room Lights", firstDevice.getName());
        assertEquals("LIGHTING", firstDevice.getType());
        assertEquals(100.0, firstDevice.getPower());

        assertEquals(1, config.getEnergies().size());
        HouseConfig.EnergyConfig firstEnergy = config.getEnergies().get(0);
        assertEquals("Solar Panels", firstEnergy.getName());
        assertEquals("SOLAR", firstEnergy.getType());
        assertEquals(5000.0, firstEnergy.getOutput());

        assertEquals(1, config.getBatteries().size());
        HouseConfig.BatteryConfig battery = config.getBatteries().get(0);
        assertEquals("Main Battery", battery.getName());
        assertEquals(10000.0, battery.getCapacity());
        assertEquals(2000.0, battery.getMaxChargeRate());
        assertEquals(2000.0, battery.getMaxDischargeRate());
    }

    @Test
    void testLoadFromNonExistentFile() {
        assertThrows(IOException.class, () ->
            HouseConfig.loadFromFile("non_existent_file.yml")
        );
```

Unit Test Results



```
Finished after 22.825 seconds

Runs:  40/40          Errors:  0          Failures:  0

> SmartHouseTestSuite [Runner: JUnit 5] (22.746 s)
```

Thank you