# Project requirements

## 1. Introduction

- To develop a concurrent Energy Management System that efficiently handles charging and usage of batteries through multiple threads.

## 2. Prerequisite Reading

- Review and understand the basics of concurrency as covered in the tutorials:

## 3. Functional Requirements

- **Simulate Multithreaded Charging of Reserved Batteries:**
  - Implement concurrent charging operations from multiple energy sources on a set of reserved batteries.
  - Each energy source will represent a thread that charges the battery independently.
- **Simulate Multithreaded Usage of Batteries:**
  - Implement concurrent usage of batteries by several energy-consuming objects.
  - Each consuming object will represent a thread that drains energy from the battery.
- **Control Overload of the System:**
  - Monitor the battery's state to avoid overloading. For example, prevent charging beyond maximum capacity and shut off usage when the battery reaches a critical low level.
  - Implement logic to handle the battery charge level thresholds, such as stopping usage threads when the battery is depleted or halting charging when full.
- **Evidence of Functionality:**
  - Provide code of each part of the functionality working as intended.
  - Videos of presentation

## 4. Analysis and Questions

- **Comparison of Concurrency Models (Pros & Cons):**
  - **Thread-Based Model:**
    - **Pros:** Simpler to implement, suitable for I/O-bound tasks, and leverages shared memory.
    - **Cons:** Risk of thread contention, potential for deadlocks, and high memory usage with many threads.
  - **Actor-Based Model:** (Optional if applicable)
    - **Pros:** Easier to manage state without locks, fault isolation.

        - **Cons:** Requires message-passing infrastructure, can be slower for some tasks.

- **Concurrency vs. Parallelism:**
    - **Concurrency**: Refers to managing multiple tasks at once, making progress on each, often through interleaving execution. Tasks may not run simultaneously but switch between each other quickly.
    - **Parallelism**: Involves executing multiple tasks simultaneously, requiring multiple processors or cores. This speeds up task completion but requires independent tasks.
- **Blocking vs. Non-blocking Concurrency Algorithms:**
    - **Blocking Algorithms**: Rely on thread suspension and resumption, using locks to ensure resource safety. They ensure predictability but can lead to deadlocks.
    - **Non-blocking Algorithms**: Use atomic operations to handle concurrency without suspending threads, allowing for higher performance in high-concurrency situations.

## 5. Implementation Plan

- List the classes and methods you plan to implement for the above functionality, such as:
    - Battery class: to manage battery charge state.
    - EnegySourceSimulator class: to represent the multithreaded charging of batteries.
    - BatteryUsageSimulator class: to represent the multithreaded usage by energy-consuming objects.
    - ControlSystem class: to monitor and control overload situations.