# Mock Project: Device Fleet Management Service

go/mock-project-device-management

**Project:** Device Fleet Management Service

**Timeframe**: 1 week

**Objective:**

Your task is to design and implement a backend service that simulates the management of a fleet of devices. This service will expose an API to register devices, track their status, and trigger mock long-running operations like software updates. A simple CLI client should also be developed to interact with the service.

**The key of this assignment is to deliver a prototype meeting below core requirements, instead of a perfect product.**

**Core Requirements:**

1. Backend Service (C++):
    - Develop a backend service using C++. You can use open-source libraries for HTTP/gRPC serving (e.g., `grpc++`, `httplib`, `pistache`).
    - The service should maintain an in-memory state of the devices.
2. API Design (gRPC & Protocol Buffers):
    - Define a gRPC service using Protocol Buffers (`.proto` files).
    - The API should include methods for:
        - `RegisterDevice`: Adds a new device to the fleet. Input: Device ID, initial state.
        - `SetDeviceStatus`: Manually sets a device's status. Input: Device ID, Status (e.g., IDLE, BUSY, OFFLINE, MAINTENANCE).
        - `GetDeviceInfo`: Retrieves the current state and info for a Device ID.
        - `InitiateDeviceAction`: Triggers a mock long-running action on a device. Input: Device ID, ActionType (e.g., SOFTWARE_UPDATE), ActionParams (e.g., software version). This should:
            - Return a unique Action ID.
            - Change the device state to reflect the ongoing action (e.g., UPDATING, RECOVERING).

            - Simulate the action asynchronously (e.g., using a thread). The device state should transition back to IDLE or an ERROR state after a duration.
        - `GetDeviceActionStatus`: Checks the status of a previously initiated action using the Action ID. Output: Action Status (e.g., PENDING, RUNNING, COMPLETED, FAILED).
3. State Management & Simulation:
    - Device states and ongoing actions should be managed in memory.
    - Simulate time-consuming actions. For example, a SOFTWARE_UPDATE action might take 10-30 seconds, during which the device is in an UPDATING state.
4. CLI Client (Python):
    - Develop a command-line interface (CLI) tool in Python that uses the gRPC API to interact with the backend service.
    - The CLI should allow users to:
        - Register new devices.
        - List devices and their statuses.
        - Get the status of a specific device.
        - Trigger a SOFTWARE_UPDATE action on a device.
        - Poll the status of an ongoing action.

**Useful Technologies & Libraries:**

- C++: Standard library, gRPC (`grpc++`), Protocol Buffers compiler (`protoc`).
- Python: `grpcio`, `grpcio-tools`, `protobuf`, `argparse`.
- Protocol Buffers: For defining the API service and messages.

**Deliverables**: Please provide:

1. A link to a Git repository containing all source code for both the backend and frontend.
2. A comprehensive `README.md` file in the repository that includes:
    - Clear instructions on how to set up, build, and run both the C++ backend service and the Python CLI application on a Linux environment.
    - The `.proto` service definition file.
    - An overview of the architecture, explaining how the backend and CLI interact.
    - Detailed examples of how to use the CLI to interact with the service for all API functions.
    - Any assumptions made, simplifications, and potential next steps or improvements if more time were available.