# Main program structure description

Here is a structured prompt you can paste directly into your Agent manager (Antigravity). It summarizes the architectural decisions, the "3-Lane" logic, and the specific refactoring requirements for the files.

---

**Project Specification: Laser Spectroscopy Control System (PySide6)**

**Objective**

Build a high-performance GUI application for laser line locking using **Python 3** and **PySide6**. The system must handle real-time hardware control, data visualization, and external remote control (ZeroRPC) without freezing the user interface.

**Architecture: The "3-Lane" Concurrency Model**

The application must strictly follow a multi-threaded architecture separated into three distinct "lanes" to ensure thread safety and responsiveness.

- **Lane 1: The GUI (Main Thread)**
    - **Role:** Visualization and User Interaction only.
    - **Components:** MainWindow (PySide6).
    - **Constraints:** Must never perform blocking operations (no time.sleep, no hardware I/O). It only listens for Signals from other lanes to update labels/plots.
- **Lane 2: The Hardware Worker (High-Priority Thread)**
    - **Role:** Performs the laser locking PID loop, signal analysis, and hardware I/O.
    - **Components:** LaserLockController (refactored).
    - **Constraints:**
        - Runs in a dedicated QThread.
        - **Refactoring Requirement:** The existing LaserLockingController_new.py must be stripped of all matplotlib, print(), and input() calls.
        - **Communication:** Instead of plotting, it must emit() data via PySide6 Signal.
        - **Lifecycle:** Must include a _stop_requested flag to break infinite loops cleanly.
- **Lane 3: The Services/Comms Worker (Low-Priority Thread)**
    - **Role:** Handles high-latency or blocking external communications.
    - **Components:** ServiceManager.
    - **Capabilities:**
        - **Grafana:** Receives data signals from Lane 2 and uploads them via HTTP.
        - **ZeroRPC:** Hosts a blocking RPC server (running in a daemon sub-thread) to accept remote commands (e.g., remote_lock()).
        - **Bridging:** Converts incoming RPC commands into PySide6 Signals to safely trigger actions in Lane 2.

**Coordinator: The GeneralManager**

A central class GeneralManager is responsible for:

1. **Dependency Injection:** Instantiating Interface, DataHandler, GrafanaManager.
2. **Thread Setup:** Creating the LaserLockController and ServiceManager, creating QThread instances, and using .moveToThread().
3. **Signal Wiring:** Connecting the "Data Ready" signals from Lane 2 to the "Upload" slots in Lane 3 and "Update Plot" slots in Lane 1.