**🔍 Giải Thích Chi Tiết**

**1.**core/**(Lõi ứng dụng)**

* app.h/cpp:
  + Điều khiển luồng chính của ứng dụng (khởi tạo, shutdown).
  + Kết nối các module (ui, data, command).
* data.h/cpp:
  + Xử lý kết nối database (SQLite/MySQL).
  + Cung cấp API để product, command truy vấn dữ liệu.

**2.**modules/**(Các tính năng độc lập)**

* product/:
  + product.h/cpp: Lớp cơ sở chứa thông tin hàng hóa (ID, tên, số lượng).
  + electronic.h/cpp, clothing.h/cpp: Class con mở rộng (thêm thuộc tính riêng).
  + productFactory.h/cpp: Tạo đối tượng Product dựa trên loại hàng (Factory Pattern).
* command/:
  + command.h/cpp: Lớp trừu tượng cho các lệnh (Command Pattern).
  + addCommand, deleteCommand, editCommand: Cụ thể hóa từng thao tác.
* user/:
  + Xử lý đăng nhập, phân quyền truy cập.

**3.**ui/**(Giao diện)**

* mainwindow.h/cpp:
  + Xử lý sự kiện từ người dùng (nhập liệu, nút bấm).
  + Hiển thị dữ liệu từ product và data.
* mainwindow.ui:
  + File XML thiết kế giao diện bằng Qt Designer.

**4.**utils/**(Tiện ích)**

* logger.h/cpp:
  + Ghi log vào file hoặc console khi user thêm/xóa/sửa sản phẩm.
* notify.h/cpp:
  + Kiểm tra số lượng tồn kho và gửi cảnh báo (qua UI hoặc email).

**5.**main.cpp

* Khởi tạo QApplication (nếu dùng Qt) và chạy App.

**⚙️ Cách Các Thành Phần Tương Tác**

1. **Người dùng** tương tác với mainwindow → Gửi lệnh (VD: "Thêm sản phẩm").
2. mainwindow gọi command (VD: addCommand.execute()).
3. command thao tác với data để cập nhật database.
4. data thông báo kết quả cho command → command ghi log qua logger.
5. notify kiểm tra số lượng tồn kho định kỳ và cảnh báo nếu cần.

**📌 Lưu Ý Quan Trọng**

1. **Tách biệt UI và Logic**:
   * mainwindow chỉ xử lý giao diện, không chứa logic nghiệp vụ.
2. **Dependency Injection**:
   * app nên khởi tạo và kết nối các module (data, command, user).
3. **Mở rộng sau này**:
   * Thêm module report/ để xuất báo cáo.
   * Thêm network/ để đồng bộ đa thiết bị.

**🎯 Kết Luận**

* core: Quản lý lifecycle app và database.
* modules: Mỗi thư mục là một tính năng độc lập (product, command, user).
* ui: Giao diện tách biệt với logic.
* utils: Tiện ích dùng chung (log, notify).

**1. "MainWindow chỉ xử lý giao diện, không chứa logic nghiệp vụ" - Kiểm tra giá trị đầu vào có phải là logic nghiệp vụ không?**

**Khái niệm**:

* **Logic nghiệp vụ (Business Logic)**: Là các quy tắc xử lý dữ liệu đặc thù của ứng dụng (ví dụ: tính toán tồn kho, validate phức tạp, áp dụng giảm giá).
* **Logic giao diện (UI Logic)**: Xử lý tương tác người dùng đơn giản (hiển thị thông báo, đổi màu nút bấm).

**Ví dụ kiểm tra giá trị đầu vào**:

cpp

// ❌ Cách sai: Đặt logic nghiệp vụ trong MainWindow

void MainWindow::onAddButtonClicked() {

QString name = ui->nameEdit->text();

int quantity = ui->quantitySpinBox->value();

// Logic nghiệp vụ (kiểm tra số lượng âm) --> KHÔNG NÊN đặt ở đây

if (quantity < 0) {

QMessageBox::warning(this, "Lỗi", "Số lượng không thể âm!");

return;

}

// Gọi Command để thêm sản phẩm

auto command = new AddProductCommand(name, quantity);

command->execute();

}

// ✅ Cách đúng: MainWindow chỉ validate UI, logic nghiệp vụ đưa vào Command

void MainWindow::onAddButtonClicked() {

QString name = ui->nameEdit->text();

int quantity = ui->quantitySpinBox->value();

// Chỉ kiểm tra UI cơ bản (ví dụ: ô trống)

if (name.isEmpty()) {

QMessageBox::warning(this, "Lỗi", "Tên sản phẩm không được trống!");

return;

}

// Logic nghiệp vụ (kiểm tra số lượng âm) đưa vào Command

auto command = new AddProductCommand(name, quantity);

command->execute(); // Command tự validate quantity < 0

}

**Tóm tắt**:

* **Kiểm tra UI đơn giản** (ô trống, định dạng email) → Có thể đặt trong MainWindow.
* **Kiểm tra nghiệp vụ** (số lượng âm, trùng mã sản phẩm) → Nên đặt trong Command hoặc Product class.

**2. "MainWindow gọi Command trực tiếp hay thông qua App?"**

**Phân tích**:

* **Cách 1: MainWindow → Command** (Direct Call):

cpp

// Trong MainWindow.cpp

void MainWindow::onDeleteButtonClicked() {

int productId = ui->idSpinBox->value();

auto command = new DeleteProductCommand(productId);

command->execute();

}

* + **Ưu điểm**: Đơn giản, phù hợp với ứng dụng nhỏ.
  + **Nhược điểm**: Khó kiểm soát tổng thể (undo/redo, logging).
* **Cách 2: MainWindow → App → Command** (Thông qua lớp trung gian):

cpp

// Trong App.cpp

void App::executeCommand(Command\* command) {

command->execute();

\_commandHistory.push(command); // Lưu lịch sử để undo/redo

Logger::log("Command executed: " + command->getName());

}

// Trong MainWindow.cpp

void MainWindow::onDeleteButtonClicked() {

int productId = ui->idSpinBox->value();

auto command = new DeleteProductCommand(productId);

\_app->executeCommand(command); // Gọi thông qua App

}

* + **Ưu điểm**: Dễ mở rộng (thêm undo/redo, logging tập trung).
  + **Nhược điểm**: Code phức tạp hơn.

**Khuyến nghị**:

* Nếu ứng dụng có **undo/redo** hoặc cần **quản lý tập trung** → Dùng cách 2 (gọi qua App).
* Nếu ứng dụng nhỏ, không cần undo/redo → Dùng cách 1 (gọi trực tiếp).

**3. "App.h/cpp có quản lý undo/redo không?"**

**Thiết kế tốt**:

* App nên đóng vai trò **điều phối** (orchestration), bao gồm cả undo/redo:

cpp

// Trong App.h

class App {

private:

std::stack<Command\*> \_undoStack;

std::stack<Command\*> \_redoStack;

public:

void executeCommand(Command\* command) {

command->execute();

\_undoStack.push(command);

}

void undo() {

if (!\_undoStack.empty()) {

Command\* cmd = \_undoStack.top();

cmd->undo(); // Mỗi Command phải implement undo()

\_redoStack.push(cmd);

\_undoStack.pop();

}

}

};

**Ví dụ workflow**:

1. **MainWindow** gọi App::executeCommand(new AddProductCommand(...)).
2. **App** thực thi command và lưu vào \_undoStack.
3. Khi user nhấn "Undo", **App** gọi undo() trên command gần nhất.

**Tại sao không đặt undo/redo trong MainWindow?**

* **Vi phạm Single Responsibility Principle**: MainWindow chỉ nên xử lý UI.
* **Khó tái sử dụng**: Nếu sau này có thêm CLI (Command Line Interface), undo/redo sẽ bị trùng code

**Kết Luận**

1. **Logic nghiệp vụ** nên tách riêng khỏi MainWindow → Dễ bảo trì, kiểm thử.
2. **Gọi Command**:
   * Nhỏ & đơn giản → Gọi trực tiếp từ MainWindow.
   * Cần undo/redo → Gọi qua App.
3. App nên quản lý undo/redo để tập trung hóa control flow.