Here's an example of how we approached a **complex functionality** in the **Inventory Management System (IMS)** project:

**Feature: Real-Time Cart Management with Persistent Session (Cart Page)**

**Problem Statement:**

Users should be able to:

* Add multiple products to their cart from the product listing.
* See their updated cart in real time.
* Maintain the cart state even if they close and reopen the cart window.

**Approach & Solution:**

1. **Cart Data Structure:**
   * I used a **Python dictionary**:  
     { product\_id: quantity }  
     This made updates (increment/decrement) fast and intuitive.
2. **Persistent Storage:**
   * I created a **simple text file** cart\_session.txt where I saved cart data in the format:  
     { user\_id: { product\_id: quantity, ... } }
   * This ensured the cart was **user-specific** and **persisted between sessions**.
3. **Encapsulation and Separation:**
   * Encapsulated cart-related logic inside a **Cart class**, separating UI and logic for better maintainability.
4. **Live Quantity Management:**
   * On the cart UI, I added:
     + Increase
     + Decrease
     + Auto-remove item when quantity hits 0
     + Live total price recalculation
5. **Product Validation:**
   * During **checkout**, I added logic to:
     + Validate if the product still exists
     + Check if the stock is enough before placing the order
     + Update stock in the database once the order is placed
6. **User Experience:**
   * If the cart is empty, display a friendly message:  
     "🛒 Your cart is empty!"
   * Added color-coded alerts for:
     + Success
     + Errors
     + Warnings

**Techniques Used:**

* File I/O for session persistence
* Dynamic Tkinter (customtkinter) UI updates
* Database joins and updates for order tracking
* Defensive programming (validations, error messages)