Module 1: The AI-Native Smells Detector, Lesson 3: Immobility | Assignment: The Trapped Discount

Overview

A discount calculation has been requested by three different teams (web checkout, mobile app, nightly invoice job). Read the existing code and explain why none of them can reuse it without copying.

Estimated Time

10 minutes

Before You Start

- Read the code block below.
- Have a blank text file or notepad ready for your answers.
- You do not need to write a full refactor - a sketch is enough.

> **REMEMBER**
> Immobility is not about how big the code is. It is about what the code imports and depends on. Ask: 'what would I have to drag along to reuse this in a different context?'

Your Task

Read the discount function below. Answer the questions in the Submit section.

```
# checkout_view.py - lives inside the web app
from myapp.db import db_session
from myapp.web import current_user, render_template
from myapp.logging import audit_logger

def apply_discount_view(order_id):
    user = current_user()
    order = db_session.query(Order).get(order_id)

    # ===== the actual discount logic =====
    if user.is_loyalty_member:
        discount = order.subtotal * 0.10
    elif order.subtotal > 100:
        discount = order.subtotal * 0.05
    else:
```

```
        discount = 0
    order.discount = discount
    order.total = order.subtotal - discount
    # ====================================

    db_session.commit()
    audit_logger.info("discount applied", order=order_id, amount=discount)
    return render_template("receipt.html", order=order)
```

What to Submit

1. Name the smell and explain in one sentence why the discount logic cannot be reused from the mobile API or the nightly invoice job as it stands.
2. List every concern OTHER than the discount calculation that this function performs. (You should find at least four.)
3. Sketch a refactor: write the signature (and one-line body description) of a pure function that contains ONLY the discount rules.
4. Bonus: name one piece of test infrastructure (a fixture, a mock, a setup step) that would become unnecessary after your refactor.

Solution Sketch (Instructor)

SOLUTION

Smell: immobility. The discount rules are correct but are buried inside a function that also handles HTTP, DB, auth, audit logging, and template rendering. Other concerns: (1) user authentication, (2) database query, (3) database commit, (4) audit logging, (5) HTML template rendering. Refactor signature: 'def calculate_discount(subtotal: float, is_loyalty_member: bool) -> float:' - returns the discount amount, no side effects. Bonus: the database fixture, the mock current_user, the audit logger mock, and the template context fixture all become unnecessary - you just call the function with two arguments and assert.

Source File Reference

This assignment is part of Lesson 3 of Module 1: The AI-Native Smells Detector. The code snippet above is a teaching example. The full hands-on for this lesson uses related source files in the source_code/ directory.