Task 1: E-Commerce App Services

The five listed requirements for the e-commerce app are as follows:

1. **User Authentication and Validation Service:**
    - **Functionality:** Ensures that users can register, log in, and access their accounts. It also handles account modifications such as email, password, country, name, and phone number.
    - **Test Cases:**
        - **Validate inputs:** Register or log in with the correct credentials.
        - **Redundant input:** Attempt to register with an already registered email address.
        - **Security constraints:** Implement two-factor authentication and email notifications for suspicious activity or account lockout after multiple failed login attempts, similar to an Apple account.
2. **Product Service:**
    - **Functionality:** Allows users to view product information such as name, price, and availability.
    - **Test Cases:**
        - **Search and filter:** Ensure that filters for name, price, and range are accessible and functional for users.
3. **Order Service:**
    - **Functionality:** Enables users to manage their orders, including placing, canceling, viewing order details, and tracking shipments.
    - **Test Cases:**
        - **Validate order:** Ensure the number of items is at least one and prevent negative values.
        - **Availability:** Check stock availability in the user's country and confirm that the product can be delivered.
4. **Payment Integration Service:**
    - **Functionality:** Handles different payment methods, including cash on delivery and digital payment services such as PayPal, credit cards, and mobile wallets.
    - **Test Cases:**

        - **Secure and valid payment:** Ensure successful payment with a valid (non-expired) payment method and send an email notification with a receipt.
        - **Refund policy:** Ensure the refund policy is clear (e.g., "Refunds will be processed within 15 to 20 business days" after order cancellation) and that notifications are sent.

5. **Recommendation and Discount Service:**
    - **Functionality:** Tracks user activity and displays related products.
    - **Test Cases:**
        - **Related products:** Ensure that recommended products are genuinely relevant to the user’s activity.

Task 2: Clean Code Principles + Code Refactoring

1. **Meaningful Names**
    - **Principle:** Use informative and self-explanatory variable name.
    - **Before Refactoring:**

```
x = 'Agile'
```

    - **After Refactoring:**

```
course_name = 'Agile'
```

2. **Single Responsibility Principle (SRP)**
    - **Principle:** A function or class should have only one reason to change.
    - **Before Refactoring (Violates SRP):**

```
def create_user(name, email):
    print(f"Saving {name} to the ITI database...")
    print(f"Sending welcome email to {email}...")
```

    - **After Refactoring (Follows SRP):**

```
def save_user(name):
    print(f"Saving {name} to the ITI database...")

def send_welcome_email(email):
    print(f"Sending welcome email to {email}...")

# Usage
save_user("Omar")
send_welcome_email("oelghareeb@gmail.com")
```

Now each function has a single responsibility which makes it easier to modify and more reusability.

3. **DRY (Don't Repeat Yourself)**
    - **Principle:** Avoid duplication and redundancy by reusing functions.
    - **Before Refactoring:**

```python
def save_user(name):
    print(f"Saving user {name} to the ITI database...")

def send_email(email):
    print(f"Sending welcome email to {email}...")

def log_activity(activity):
    print(f"Logging activity: {activity}")
```

    - **After Refactoring:**

```python
def print_message(action, value):
    print(f"{action}: {value}")

def save_user(name):
    print_message("Saving user", name)

def send_email(email):
    print_message("Sending welcome email", email)

def log_activity(activity):
    print_message("Logging activity", activity)
```

We avoid multiple print logic and using just a simple function that offers more readability and maintainability.

4. **Avoid Magic Number**
    - **Principle:** Use named constants and avoid using hardcoded values.
    - **Before Refactoring:**

```
def calculate_tax(amount):
    return amount * 0.20
```

    - **After Refactoring:**

```
TAX_RATE = 0.20

def calculate_tax(amount):
    return amount * TAX_RATE
```

This will offer more control, for example if we want to change the tax we will go to the variable here and change it rather than trying to change every single hardcoded value.

5. **Keep Functions Small & Focused**
    - **Principle:** Function should do one thing and can do it well.
    - **Before Refactoring:**

```python
def create_user(name, email):
    print(f"Saving {name} to the ITI database...")
    print(f"Sending welcome email to {email}...")
```

    - **After Refactoring:**

```python
def save_user(name):
    print(f"Saving {name} to the ITI database...")

def send_welcome_email(email):
    print(f"Sending welcome email to {email}...")

# Usage
save_user("Omar")
send_welcome_email("oelghareeb@gmail.com")
```

As we can see here the function is divided into single focus functions and if you notice this also apply **SRP** + **Small & Focused** together