```python

class Product:

def \_\_init\_\_(self, product\_id, name, description, category, price, stock):

self.product\_id = product\_id

self.name = name

self.description = description

self.category = category

self.price = price

self.stock = stock

def update\_stock(self, quantity):

self.stock += quantity

def get\_details(self):

return {

'product\_id': self.product\_id,

'name': self.name,

'description': self.description,

'category': self.category,

'price': self.price,

'stock': self.stock

}

def \_\_str\_\_(self):

return f"{self.name} (ID: {self.product\_id}), Price: ${self.price}, Stock: {self.stock}"

class Category:

def \_\_init\_\_(self, category\_id, name, description):

self.category\_id = category\_id

self.name = name

self.description = description

def get\_details(self):

return {

'category\_id': self.category\_id,

'name': self.name,

'description': self.description

}

def \_\_str\_\_(self):

return self.name

class Inventory:

def \_\_init\_\_(self):

self.products = {}

def add\_product(self, product):

self.products[product.product\_id] = product

def remove\_product(self, product\_id):

if product\_id in self.products:

del self.products[product\_id]

def get\_product(self, product\_id):

return self.products.get(product\_id)

def update\_stock(self, product\_id, quantity):

product = self.get\_product(product\_id)

if product:

product.update\_stock(quantity)

def list\_products(self):

return list(self.products.values())

class Sales:

def \_\_init\_\_(self):

self.transactions = []

def process\_sale(self, cart, customer\_id):

total\_amount = 0

for item in cart:

product = item['product']

quantity = item['quantity']

if product.stock >= quantity:

total\_amount += product.price \* quantity

product.update\_stock(-quantity)

else:

return f"Insufficient stock for {product.name}"

transaction = {

'customer\_id': customer\_id,

'items': cart,

'total\_amount': total\_amount

}

self.transactions.append(transaction)

return "Sale processed successfully."

def get\_sales\_report(self):

return self.transactions

class Accounting:

def \_\_init\_\_(self):

self.records = []

def record\_transaction(self, transaction\_type, amount, description):

record = {

'type': transaction\_type,

'amount': amount,

'description': description,

}

self.records.append(record)

def generate\_financial\_report(self):

# Simple report generation logic

income = sum(record['amount'] for record in self.records if record['type'] == 'income')

expense = sum(record['amount'] for record in self.records if record['type'] == 'expense')

profit = income - expense

return {'income': income, 'expense': expense, 'profit': profit}

class HR:

def \_\_init\_\_(self):

self.employees = {}

def add\_employee(self, employee\_id, name, position, salary):

self.employees[employee\_id] = {

'name': name,

'position': position,

'salary': salary

}

def remove\_employee(self, employee\_id):

if employee\_id in self.employees:

del self.employees[employee\_id]

def get\_employee(self, employee\_id):

return self.employees.get(employee\_id)

def generate\_payroll\_report(self):

# Simple payroll report

total\_payroll = sum(emp['salary'] for emp in self.employees.values())

return {'total\_payroll': total\_payroll}

class ShoppingCart:

def \_\_init\_\_(self):

self.items = []

def add\_item(self, product, quantity):

self.items.append({'product': product, 'quantity': quantity})

def remove\_item(self, product\_id):

self.items = [item for item in self.items if item['product'].product\_id != product\_id]

def get\_cart\_items(self):

return self.items

def clear\_cart(self):

self.items = []

# Example Usage (Illustrative)

if \_\_name\_\_ == '\_\_main\_\_':

# Initialize departments

inventory = Inventory()

sales = Sales()

accounting = Accounting()

hr = HR()

# Create categories

category1 = Category(1, "Electronics", "Electronic gadgets")

category2 = Category(2, "Clothing", "Apparel and accessories")

# Create products

product1 = Product(101, "Laptop", "High-performance laptop", category1, 1200.00, 10)

product2 = Product(102, "T-Shirt", "Cotton T-Shirt", category2, 25.00, 50)

# Add products to inventory

inventory.add\_product(product1)

inventory.add\_product(product2)

# HR: Add an employee

hr.add\_employee("E1001", "John Doe", "Manager", 60000)

# Customer adds items to cart

cart = ShoppingCart()

cart.add\_item(product1, 1)

cart.add\_item(product2, 2)

# Process sale

sale\_result = sales.process\_sale(cart.get\_cart\_items(), "C1001")

print(sale\_result)

# Accounting: Record the sale

accounting.record\_transaction('income', 1200.00 + 2 \* 25.00, 'Product sales')

# Generate reports

financial\_report = accounting.generate\_financial\_report()

payroll\_report = hr.generate\_payroll\_report()

print("Financial Report:", financial\_report)

print("Payroll Report:", payroll\_report)

print("Remaining stock of Laptop:", inventory.get\_product(101).stock)

print("Remaining stock of T-Shirt:", inventory.get\_product(102).stock)

```

```python

from typing import List, Dict, Union, Optional

import pytest

class Product:

"""

Represents a product in the online shopping system.

"""

def \_\_init\_\_(self, product\_id: int, name: str, description: str, category: 'Category', price: float, stock: int):

"""

Initializes a new product.

Args:

product\_id (int): The unique identifier for the product.

name (str): The name of the product.

description (str): A description of the product.

category (Category): The category to which the product belongs.

price (float): The price of the product.

stock (int): The number of units currently in stock.

"""

self.product\_id = product\_id

self.name = name

self.description = description

self.category = category

self.price = price

self.stock = stock

def update\_stock(self, quantity: int) -> None:

"""

Updates the stock level of the product.

Args:

quantity (int): The quantity to add to (positive) or subtract from (negative) the stock.

"""

self.stock += quantity

def get\_details(self) -> Dict[str, Union[int, str, 'Category', float]]:

"""

Returns a dictionary containing the product's details.

Returns:

Dict[str, Union[int, str, Category, float]]: A dictionary of product details.

"""

return {

'product\_id': self.product\_id,

'name': self.name,

'description': self.description,

'category': self.category,

'price': self.price,

'stock': self.stock

}

def \_\_str\_\_(self) -> str:

"""

Returns a string representation of the product.

Returns:

str: A string representation of the product.

"""

return f"{self.name} (ID: {self.product\_id}), Price: ${self.price}, Stock: {self.stock}"

class Category:

"""

Represents a product category.

"""

def \_\_init\_\_(self, category\_id: int, name: str, description: str):

"""

Initializes a new category.

Args:

category\_id (int): The unique identifier for the category.

name (str): The name of the category.

description (str): A description of the category.

"""

self.category\_id = category\_id

self.name = name

self.description = description

def get\_details(self) -> Dict[str, Union[int, str]]:

"""

Returns a dictionary containing the category's details.

Returns:

Dict[str, Union[int, str]]: A dictionary of category details.

"""

return {

'category\_id': self.category\_id,

'name': self.name,

'description': self.description

}

def \_\_str\_\_(self) -> str:

"""

Returns a string representation of the category.

Returns:

str: A string representation of the category.

"""

return self.name

class Inventory:

"""

Manages the inventory of products.

"""

def \_\_init\_\_(self):

"""

Initializes an empty inventory.

"""

self.products: Dict[int, Product] = {}

def add\_product(self, product: Product) -> None:

"""

Adds a product to the inventory.

Args:

product (Product): The product to add.

"""

if not isinstance(product, Product):

raise TypeError("product must be an instance of Product class")

self.products[product.product\_id] = product

def remove\_product(self, product\_id: int) -> None:

"""

Removes a product from the inventory.

Args:

product\_id (int): The ID of the product to remove.

"""

if product\_id in self.products:

del self.products[product\_id]

def get\_product(self, product\_id: int) -> Optional[Product]:

"""

Retrieves a product from the inventory.

Args:

product\_id (int): The ID of the product to retrieve.

Returns:

Optional[Product]: The product if found, otherwise None.

"""

return self.products.get(product\_id)

def update\_stock(self, product\_id: int, quantity: int) -> None:

"""

Updates the stock level of a product in the inventory.

Args:

product\_id (int): The ID of the product to update.

quantity (int): The quantity to add to (positive) or subtract from (negative) the stock.

"""

product = self.get\_product(product\_id)

if product:

product.update\_stock(quantity)

def list\_products(self) -> List[Product]:

"""

Returns a list of all products in the inventory.

Returns:

List[Product]: A list of products.

"""

return list(self.products.values())

class Sales:

"""

Processes sales transactions.

"""

def \_\_init\_\_(self):

"""

Initializes an empty sales transaction list.

"""

self.transactions: List[Dict[str, Union[str, List[Dict[str, Union[Product, int]]], float]]] = []

def process\_sale(self, cart: List[Dict[str, Union[Product, int]]], customer\_id: str) -> str:

"""

Processes a sale given a shopping cart and customer ID.

Args:

cart (List[Dict[str, Union[Product, int]]]): A list of items in the cart, where each item is a

dictionary

containing the product and quantity.

customer\_id (str): The ID of the customer making the purchase.

Returns:

str: A message indicating the result of the sale processing.

"""

total\_amount = 0

for item in cart:

product = item['product']

quantity = item['quantity']

if product.stock >= quantity:

total\_amount += product.price \* quantity

product.update\_stock(-quantity)

else:

return f"Insufficient stock for {product.name}"

transaction = {

'customer\_id': customer\_id,

'items': cart,

'total\_amount': total\_amount

}

self.transactions.append(transaction)

return "Sale processed successfully."

def get\_sales\_report(self) -> List[Dict[str, Union[str, List[Dict[str, Union[Product, int]]], float]]]: """

Returns a list of all sales transactions.

Returns:

List[Dict[str, Union[str, List[Dict[str, Union[Product, int]]], float]]]: A list of sales transactions.

"""

return self.transactions

class Accounting:

"""

Manages financial records and generates reports.

"""

def \_\_init\_\_(self):

"""

Initializes an empty list of financial records.

"""

self.records: List[Dict[str, Union[str, float]]] = []

def record\_transaction(self, transaction\_type: str, amount: float, description: str) -> None:

"""

Records a financial transaction.

Args:

transaction\_type (str): The type of transaction (e.g., 'income', 'expense').

amount (float): The amount of the transaction.

description (str): A description of the transaction.

"""

record = {

'type': transaction\_type,

'amount': amount,

'description': description,

}

self.records.append(record)

def generate\_financial\_report(self) -> Dict[str, float]:

"""

Generates a simple financial report.

Returns:

Dict[str, float]: A dictionary containing the total income, expense, and profit.

"""

income = sum(record['amount'] for record in self.records if record['type'] == 'income')

expense = sum(record['amount'] for record in self.records if record['type'] == 'expense')

profit = income - expense

return {'income': income, 'expense': expense, 'profit': profit}

class HR:

"""

Manages employee records and generates payroll reports.

"""

def \_\_init\_\_(self):

"""

Initializes an empty employee dictionary.

"""

self.employees: Dict[str, Dict[str, Union[str, int]]] = {}

def add\_employee(self, employee\_id: str, name: str, position: str, salary: int) -> None:

"""

Adds a new employee to the system.

Args:

employee\_id (str): The unique identifier for the employee.

name (str): The name of the employee.

position (str): The job position of the employee.

salary (int): The salary of the employee.

"""

self.employees[employee\_id] = {

'name': name,

'position': position,

'salary': salary

}

def remove\_employee(self, employee\_id: str) -> None:

"""

Removes an employee from the system.

Args:

employee\_id (str): The ID of the employee to remove.

"""

if employee\_id in self.employees:

del self.employees[employee\_id]

def get\_employee(self, employee\_id: str) -> Optional[Dict[str, Union[str, int]]]:

"""

Retrieves an employee's information.

Args:

employee\_id (str): The ID of the employee to retrieve.

Returns:

Optional[Dict[str, Union[str, int]]]: A dictionary containing the employee's information, or None if not found.

"""

return self.employees.get(employee\_id)

def generate\_payroll\_report(self) -> Dict[str, int]:

"""

Generates a simple payroll report.

Returns:

Dict[str, int]: A dictionary containing the total payroll.

"""

total\_payroll = sum(emp['salary'] for emp in self.employees.values())

return {'total\_payroll': total\_payroll}

class ShoppingCart:

"""

Represents a shopping cart for customers.

"""

def \_\_init\_\_(self):

"""

Initializes an empty shopping cart.

"""

self.items: List[Dict[str, Union[Product, int]]] = []

def add\_item(self, product: Product, quantity: int) -> None:

"""

Adds an item to the shopping cart.

Args:

product (Product): The product to add.

quantity (int): The quantity of the product to add.

"""

self.items.append({'product': product, 'quantity': quantity})

def remove\_item(self, product\_id: int) -> None:

"""

Removes an item from the shopping cart.

Args:

product\_id (int): The ID of the product to remove.

"""

self.items = [item for item in self.items if item['product'].product\_id != product\_id]

def get\_cart\_items(self) -> List[Dict[str, Union[Product, int]]]:

"""

Returns a list of all items in the shopping cart.

Returns:

List[Dict[str, Union[Product, int]]]: A list of items in the cart.

"""

return self.items

def clear\_cart(self) -> None:

"""

Clears all items from the shopping cart.

"""

self.items = []

# Example Usage (Illustrative)

if \_\_name\_\_ == '\_\_main\_\_':

# Initialize departments

inventory = Inventory()

sales = Sales()

accounting = Accounting()

hr = HR()

# Create categories

category1 = Category(1, "Electronics", "Electronic gadgets")

category2 = Category(2, "Clothing", "Apparel and accessories")

# Create products

product1 = Product(101, "Laptop", "High-performance laptop", category1, 1200.00, 10)

product2 = Product(102, "T-Shirt", "Cotton T-Shirt", category2, 25.00, 50)

# Add products to inventory

inventory.add\_product(product1)

inventory.add\_product(product2)

# HR: Add an employee

hr.add\_employee("E1001", "John Doe", "Manager", 60000)

# Customer adds items to cart

cart = ShoppingCart()

cart.add\_item(product1, 1)

cart.add\_item(product2, 2)

# Process sale

sale\_result = sales.process\_sale(cart.get\_cart\_items(), "C1001")

print(sale\_result)

# Accounting: Record the sale

accounting.record\_transaction('income', 1200.00 + 2 \* 25.00, 'Product sales')

# Generate reports

financial\_report = accounting.generate\_financial\_report()

payroll\_report = hr.generate\_payroll\_report()

print("Financial Report:", financial\_report)

print("Payroll Report:", payroll\_report)

print("Remaining stock of Laptop:", inventory.get\_product(101).stock)

print("Remaining stock of T-Shirt:", inventory.get\_product(102).stock)

# Pytest tests

class TestProduct:

def test\_product\_creation(self):

category = Category(1, "Test Category", "Test Description")

product = Product(1, "Test Product", "Test Description", category, 100.0, 10)

assert product.product\_id == 1

assert product.name == "Test Product"

assert product.description == "Test Description"

assert product.category == category

assert product.price == 100.0

assert product.stock == 10

def test\_update\_stock(self):

category = Category(1, "Test Category", "Test Description")

product = Product(1, "Test Product", "Test Description", category, 100.0, 10)

product.update\_stock(5)

assert product.stock == 15

product.update\_stock(-2)

assert product.stock == 13

def test\_get\_details(self):

category = Category(1, "Test Category", "Test Description")

product = Product(1, "Test Product", "Test Description", category, 100.0, 10)

details = product.get\_details()

assert details['product\_id'] == 1

assert details['name'] == "Test Product"

assert details['description'] == "Test Description"

assert details['category'] == category

assert details['price'] == 100.0

assert details['stock'] == 10

class TestInventory:

def test\_add\_product(self):

inventory = Inventory()

category = Category

Flow started with ID: 04afd2cd-a8a8-4984-8560-045ed833f546

# Agent: Junior python developer

## Task: you have to write python solution without python type hints and pydocs for this problem 'Write an object oriented Python program with crewai for a web based online shopping cart system that includes inventory management, sales processing, accounting, HR functions, and item categorization. The system should handle stock levels, process sales transactions, generate financial reports, manage employee records, and organize products into various categories.the program must have the following features: - Organized all dearptments into separate classes organize all prodcut categories in separate classes appealing user interface and user experience customers must be able to give review and feedback about the products '

# Agent: Junior python developer

## Final Answer:

```python

class Product:

def \_\_init\_\_(self, product\_id, name, description, price, category):

self.product\_id = product\_id

self.name = name

self.description = description

self.price = price

self.category = category

self.stock = 0

self.reviews = []

def add\_stock(self, quantity):

self.stock += quantity

def remove\_stock(self, quantity):

if self.stock >= quantity:

self.stock -= quantity

return True

else:

return False

def add\_review(self, customer\_name, rating, comment):

self.reviews.append({"customer": customer\_name, "rating": rating, "comment": comment})

def get\_average\_rating(self):

if not self.reviews:

return 0

total\_rating = sum(review["rating"] for review in self.reviews)

return total\_rating / len(self.reviews)

def \_\_str\_\_(self):

return f"{self.name} ({self.product\_id}) - ${self.price} - Stock: {self.stock}"

class Category:

def \_\_init\_\_(self, name, description):

self.name = name

self.description = description

self.products = []

def add\_product(self, product):

self.products.append(product)

def remove\_product(self, product):

if product in self.products:

self.products.remove(product)

def \_\_str\_\_(self):

return self.name

class Inventory:

def \_\_init\_\_(self):

self.products = {}

def add\_product(self, product):

if product.product\_id not in self.products:

self.products[product.product\_id] = product

return True

else:

return False

def remove\_product(self, product\_id):

if product\_id in self.products:

del self.products[product\_id]

return True

else:

return False

def get\_product(self, product\_id):

return self.products.get(product\_id)

def update\_stock(self, product\_id, quantity):

product = self.get\_product(product\_id)

if product:

product.add\_stock(quantity)

return True

return False

def display\_inventory(self):

for product\_id, product in self.products.items():

print(product)

class SalesProcessor:

def \_\_init\_\_(self):

self.transactions = []

def process\_sale(self, cart, customer):

total\_amount = 0

for item in cart.items:

total\_amount += item.product.price \* item.quantity

transaction = {

"transaction\_id": len(self.transactions) + 1,

"customer": customer,

"items": cart.items,

"total\_amount": total\_amount,

"timestamp": "placeholder" # Replace with actual timestamp

}

self.transactions.append(transaction)

cart.clear()

return transaction

def get\_sales\_report(self):

# Placeholder for generating sales reports

report = "Sales Report:\n"

for transaction in self.transactions:

report += f"Transaction ID: {transaction['transaction\_id']}, Customer: {transaction['customer']}, Total: ${transaction['total\_amount']}\n"

return report

class Accounting:

def \_\_init\_\_(self):

self.records = []

def record\_transaction(self, transaction, description):

record = {

"transaction\_id": len(self.records) + 1,

"transaction": transaction,

"description": description,

"timestamp": "placeholder" # Replace with actual timestamp

}

self.records.append(record)

def generate\_financial\_reports(self):

# Placeholder for generating financial reports

report = "Financial Report:\n"

for record in self.records:

report += f"Transaction ID: {record['transaction\_id']}, Description: {record['description']}, Amount: ${record['transaction']['total\_amount']}\n"

return report

class HR:

def \_\_init\_\_(self):

self.employees = {}

def add\_employee(self, employee\_id, name, position, salary):

if employee\_id not in self.employees:

self.employees[employee\_id] = {"name": name, "position": position, "salary": salary}

return True

else:

return False

def remove\_employee(self, employee\_id):

if employee\_id in self.employees:

del self.employees[employee\_id]

return True

else:

return False

def update\_employee\_info(self, employee\_id, name=None, position=None, salary=None):

if employee\_id in self.employees:

if name:

self.employees[employee\_id]["name"] = name

if position:

self.employees[employee\_id]["position"] = position

if salary:

self.employees[employee\_id]["salary"] = salary

return True

else:

return False

def get\_employee\_details(self, employee\_id):

return self.employees.get(employee\_id)

def manage\_payroll(self):

# Placeholder for managing payroll

payroll\_report = "Payroll Report:\n"

for employee\_id, employee in self.employees.items():

payroll\_report += f"Employee ID: {employee\_id}, Name: {employee['name']}, Salary: ${employee['salary']}\n"

return payroll\_report

class ShoppingCart:

def \_\_init\_\_(self):

self.items = []

def add\_item(self, product, quantity):

self.items.append(CartItem(product, quantity))

def remove\_item(self, product):

self.items = [item for item in self.items if item.product != product]

def update\_quantity(self, product, quantity):

for item in self.items:

if item.product == product:

item.quantity = quantity

break

def clear(self):

self.items = []

def get\_total(self):

total = 0

for item in self.items:

total += item.product.price \* item.quantity

return total

def display\_cart(self):

if not self.items:

print("Your cart is empty.")

else:

print("Shopping Cart:")

for item in self.items:

print(f"{item.product.name} x {item.quantity} = ${item.product.price \* item.quantity}")

print(f"Total: ${self.get\_total()}")

class CartItem:

def \_\_init\_\_(self, product, quantity):

self.product = product

self.quantity = quantity

class WebInterface:

def \_\_init\_\_(self, inventory, sales\_processor, accounting, hr):

self.inventory = inventory

self.sales\_processor = sales\_processor

self.accounting = accounting

self.hr = hr

self.categories = {} # Dictionary to store categories

self.shopping\_cart = ShoppingCart()

def add\_category(self, category):

if category.name not in self.categories:

self.categories[category.name] = category

return True

return False

def remove\_category(self, category\_name):

if category\_name in self.categories:

del self.categories[category\_name]

return True

return False

def display\_categories(self):

print("Available Categories:")

for category\_name, category in self.categories.items():

print(f"- {category\_name}")

def add\_product\_to\_category(self, product, category\_name):

if category\_name in self.categories:

self.categories[category\_name].add\_product(product)

product.category = category\_name

return True

return False

def display\_products\_by\_category(self, category\_name):

if category\_name in self.categories:

print(f"Products in {category\_name}:")

for product in self.categories[category\_name].products:

print(product)

else:

print("Category not found.")

def display\_product\_details(self, product\_id):

product = self.inventory.get\_product(product\_id)

if product:

print("Product Details:")

print(f"Name: {product.name}")

print(f"Description: {product.description}")

print(f"Price: ${product.price}")

print(f"Stock: {product.stock}")

print(f"Category: {product.category}")

print("Reviews:")

for review in product.reviews:

print(f"- {review['customer']}: {review['rating']} stars, {review['comment']}")

else:

print("Product not found.")

def add\_to\_cart(self, product\_id, quantity):

product = self.inventory.get\_product(product\_id)

if product:

if product.stock >= quantity:

self.shopping\_cart.add\_item(product, quantity)

product.remove\_stock(quantity) # Update inventory

print(f"{quantity} {product.name}(s) added to cart.")

else:

print("Not enough stock.")

else:

print("Product not found.")

def view\_cart(self):

self.shopping\_cart.display\_cart()

def checkout(self, customer\_name):

if not self.shopping\_cart.items:

print("Your cart is empty.")

return None

transaction = self.sales\_processor.process\_sale(self.shopping\_cart, customer\_name)

self.accounting.record\_transaction(transaction, f"Sale to {customer\_name}")

print("Checkout complete. Thank you for your purchase!")

print(f"Transaction ID: {transaction['transaction\_id']}, Total: ${transaction['total\_amount']}")

return transaction

def process\_return(self, transaction\_id):

#logic to process returns

pass

def simulate\_web\_interaction(self):

# Placeholder for simulating user interaction through a web interface

print("\nSimulating Web Interaction...\n")

# Example Usage:

self.display\_categories()

product\_id = input("Enter product ID to view details (or 'q' to quit): ")

if product\_id != 'q':

self.display\_product\_details(int(product\_id))

product\_id = input("Enter product ID to add to cart (or 'q' to skip): ")

if product\_id != 'q':

quantity = int(input("Enter quantity: "))

self.add\_to\_cart(int(product\_id), quantity)

self.view\_cart()

checkout = input("Proceed to checkout? (y/n): ")

if checkout == 'y':

customer\_name = input("Enter your name: ")

self.checkout(customer\_name)

def integrate\_crewai(self):

# Placeholder for integrating CrewAI for intelligent automation

print("Integrating CrewAI for automated tasks...")

# Add CrewAI code here to automate tasks like inventory management,

# customer support, or marketing campaigns.

pass

# Example Usage

inventory = Inventory()

sales\_processor = SalesProcessor()

accounting = Accounting()

hr = HR()

web\_interface = WebInterface(inventory, sales\_processor, accounting, hr)

# Create Categories

electronics\_category = Category("Electronics", "Electronic devices and accessories")

clothing\_category = Category("Clothing", "Apparel for men and women")

web\_interface.add\_category(electronics\_category)

web\_interface.add\_category(clothing\_category)

# Create Products

product1 = Product(1, "Laptop", "High-performance laptop", 1200, "Electronics")

product2 = Product(2, "Smartphone", "Latest smartphone model", 800, "Electronics")

product3 = Product(3, "T-shirt", "Cotton T-shirt", 25, "Clothing")

# Add Products to Inventory

inventory.add\_product(product1)

inventory.add\_product(product2)

inventory.add\_product(product3)

# Add products to categories

web\_interface.add\_product\_to\_category(product1, "Electronics")

web\_interface.add\_product\_to\_category(product2, "Electronics")

web\_interface.add\_product\_to\_category(product3, "Clothing")

# Add Stock

inventory.update\_stock(1, 10)

inventory.update\_stock(2, 15)

inventory.update\_stock(3, 50)

# Example Review

product1.add\_review("Alice", 5, "Great laptop, fast and reliable!")

product1.add\_review("Bob", 4, "Good value for the price.")

# Add Employee

hr.add\_employee(101, "John Doe", "Sales Manager", 60000)

# Simulate web interaction

web\_interface.simulate\_web\_interaction()

# Get reports

print(sales\_processor.get\_sales\_report())

print(accounting.generate\_financial\_reports())

print(hr.manage\_payroll())

# Integrate CrewAI

web\_interface.integrate\_crewai()

```

```python

import pytest

from typing import List, Dict, Optional, Union

from datetime import datetime

class Product:

"""

Represents a product in the online store.

Attributes:

product\_id (int): Unique identifier for the product.

name (str): Name of the product.

description (str): Detailed description of the product.

price (float): Price of the product.

category (str): Category the product belongs to.

stock (int): Current stock level of the product.

reviews (List[Dict[str, Union[str, int]]]): List of reviews for the product.

"""

def \_\_init\_\_(self, product\_id: int, name: str, description: str, price: float, category: str):

"""

Initializes a new Product instance.

"""

self.product\_id = product\_id

self.name = name

self.description = description

self.price = price

self.category = category

self.stock = 0

self.reviews: List[Dict[str, Union[str, int]]] = []

def add\_stock(self, quantity: int) -> None:

"""

Adds stock to the product.

Args:

quantity (int): The quantity to add.

"""

self.stock += quantity

def remove\_stock(self, quantity: int) -> bool:

"""

Removes stock from the product.

Args:

quantity (int): The quantity to remove.

Returns:

bool: True if the stock was successfully removed, False otherwise.

"""

if self.stock >= quantity:

self.stock -= quantity

return True

else:

return False

def add\_review(self, customer\_name: str, rating: int, comment: str) -> None:

"""

Adds a review to the product.

Args:

customer\_name (str): Name of the customer who wrote the review.

rating (int): Rating given by the customer (e.g., 1-5).

comment (str): Customer's comments about the product.

"""

self.reviews.append({"customer": customer\_name, "rating": rating, "comment": comment})

def get\_average\_rating(self) -> float:

"""

Calculates the average rating for the product.

Returns:

float: The average rating. Returns 0 if there are no reviews.

"""

if not self.reviews:

return 0

total\_rating = sum(review["rating"] for review in self.reviews)

return total\_rating / len(self.reviews)

def \_\_str\_\_(self) -> str:

"""

Returns a string representation of the product.

"""

return f"{self.name} ({self.product\_id}) - ${self.price} - Stock: {self.stock}"

class Category:

"""

Represents a category of products.

Attributes:

name (str): Name of the category.

description (str): Description of the category.

products (List[Product]): List of products in the category.

"""

def \_\_init\_\_(self, name: str, description: str):

"""

Initializes a new Category instance.

"""

self.name = name

self.description = description

self.products: List[Product] = []

def add\_product(self, product: Product) -> None:

"""

Adds a product to the category.

Args:

product (Product): The product to add.

"""

self.products.append(product)

def remove\_product(self, product: Product) -> None:

"""

Removes a product from the category.

Args:

product (Product): The product to remove.

"""

if product in self.products:

self.products.remove(product)

def \_\_str\_\_(self) -> str:

"""

Returns a string representation of the category.

"""

return self.name

class Inventory:

"""

Manages the inventory of products.

Attributes:

products (Dict[int, Product]): Dictionary of products, with product\_id as key.

"""

def \_\_init\_\_(self):

"""

Initializes a new Inventory instance.

"""

self.products: Dict[int, Product] = {}

def add\_product(self, product: Product) -> bool:

"""

Adds a product to the inventory.

Args:

product (Product): The product to add.

Returns:

bool: True if the product was successfully added, False otherwise.

"""

if product.product\_id not in self.products:

self.products[product.product\_id] = product

return True

else:

return False

def remove\_product(self, product\_id: int) -> bool:

"""

Removes a product from the inventory.

Args:

product\_id (int): The ID of the product to remove.

Returns:

bool: True if the product was successfully removed, False otherwise.

"""

if product\_id in self.products:

del self.products[product\_id]

return True

else:

return False

def get\_product(self, product\_id: int) -> Optional[Product]:

"""

Retrieves a product from the inventory.

Args:

product\_id (int): The ID of the product to retrieve.

Returns:

Optional[Product]: The product if found, None otherwise.

"""

return self.products.get(product\_id)

def update\_stock(self, product\_id: int, quantity: int) -> bool:

"""

Updates the stock level of a product.

Args:

product\_id (int): The ID of the product to update.

quantity (int): The quantity to add to the stock.

Returns:

bool: True if the stock was successfully updated, False otherwise.

"""

product = self.get\_product(product\_id)

if product:

product.add\_stock(quantity)

return True

return False

def display\_inventory(self) -> None:

"""

Displays the current inventory.

"""

for product\_id, product in self.products.items():

print(product)

class SalesProcessor:

"""

Processes sales transactions.

Attributes:

transactions (List[Dict[str, Union[int, str, List[CartItem], float, datetime]]]): List of sales transactions.

"""

def \_\_init\_\_(self):

"""

Initializes a new SalesProcessor instance.

"""

self.transactions: List[Dict[str, Union[int, str, List['CartItem'], float, datetime]]] = []

def process\_sale(self, cart: 'ShoppingCart', customer: str) -> Dict[str, Union[int, str, List['CartItem'], float, datetime]]:

"""

Processes a sale transaction.

Args:

cart (ShoppingCart): The shopping cart containing the items to be sold.

customer (str): The name of the customer making the purchase.

Returns:

Dict[str, Union[int, str, List[CartItem], float, datetime]]: A dictionary representing the sale transaction.

"""

total\_amount = 0

for item in cart.items:

total\_amount += item.product.price \* item.quantity

transaction = {

"transaction\_id": len(self.transactions) + 1,

"customer": customer,

"items": cart.items,

"total\_amount": total\_amount,

"timestamp": datetime.now()

}

self.transactions.append(transaction)

cart.clear()

return transaction

def get\_sales\_report(self) -> str:

"""

Generates a sales report.

Returns:

str: A string containing the sales report.

"""

report = "Sales Report:\n"

for transaction in self.transactions:

report += f"Transaction ID: {transaction['transaction\_

React Framework with Tailwind

import React, { useEffect, useState } from "react";

import axios from "axios";

const App = () => {

const [products, setProducts] = useState([]);

const [review, setReview] = useState("");

const [selectedProduct, setSelectedProduct] = useState(null);

useEffect(() => {

axios.get("http://localhost:5000/")

.then(response => {

setProducts(response.data.products);

})

.catch(error => console.error("Error fetching products:", error));

}, []);

const submitReview = (productId) => {

axios.post("http://localhost:5000/add\_review", { product\_id: productId, review })

.then(() => {

alert("Review added successfully");

setReview("");

})

.catch(error => console.error("Error adding review:", error));

};

return (

<div className="container mx-auto p-4">

<h1 className="text-2xl font-bold mb-4">Online Shopping Cart</h1>

<div className="grid grid-cols-3 gap-4">

{products.map((product) => (

<div key={product.id} className="border p-4 rounded-lg shadow-md">

<img src={product.image} alt={product.name} className="w-full h-40 object-cover rounded-md mb-2" />

<h2 className="text-xl font-semibold">{product.name}</h2>

<p className="text-gray-600">{product.category}</p>

<p className="text-lg font-bold">${product.price}</p>

<button

className="mt-2 bg-blue-500 text-white px-4 py-2 rounded"

onClick={() => setSelectedProduct(product.id)}

>

Add Review

</button>

</div>

))}

</div>

{selectedProduct && (

<div className="mt-4 p-4 border rounded">

<h3 className="text-lg font-semibold">Add Review</h3>

<textarea

className="w-full p-2 border rounded"

value={review}

onChange={(e) => setReview(e.target.value)}

></textarea>

<button

className="mt-2 bg-green-500 text-white px-4 py-2 rounded"

onClick={() => submitReview(selectedProduct)}

>

Submit Review

</button>

</div>

)}

</div>

);

};

export default App;