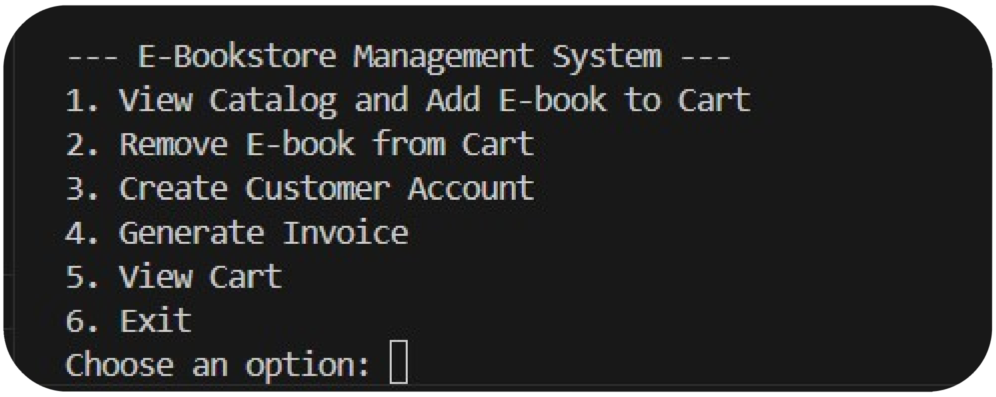
E-BOOKSTORE MANAGEMENT SYSTEM

name



```
--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option: 
```

Table of contents

I. Abstract

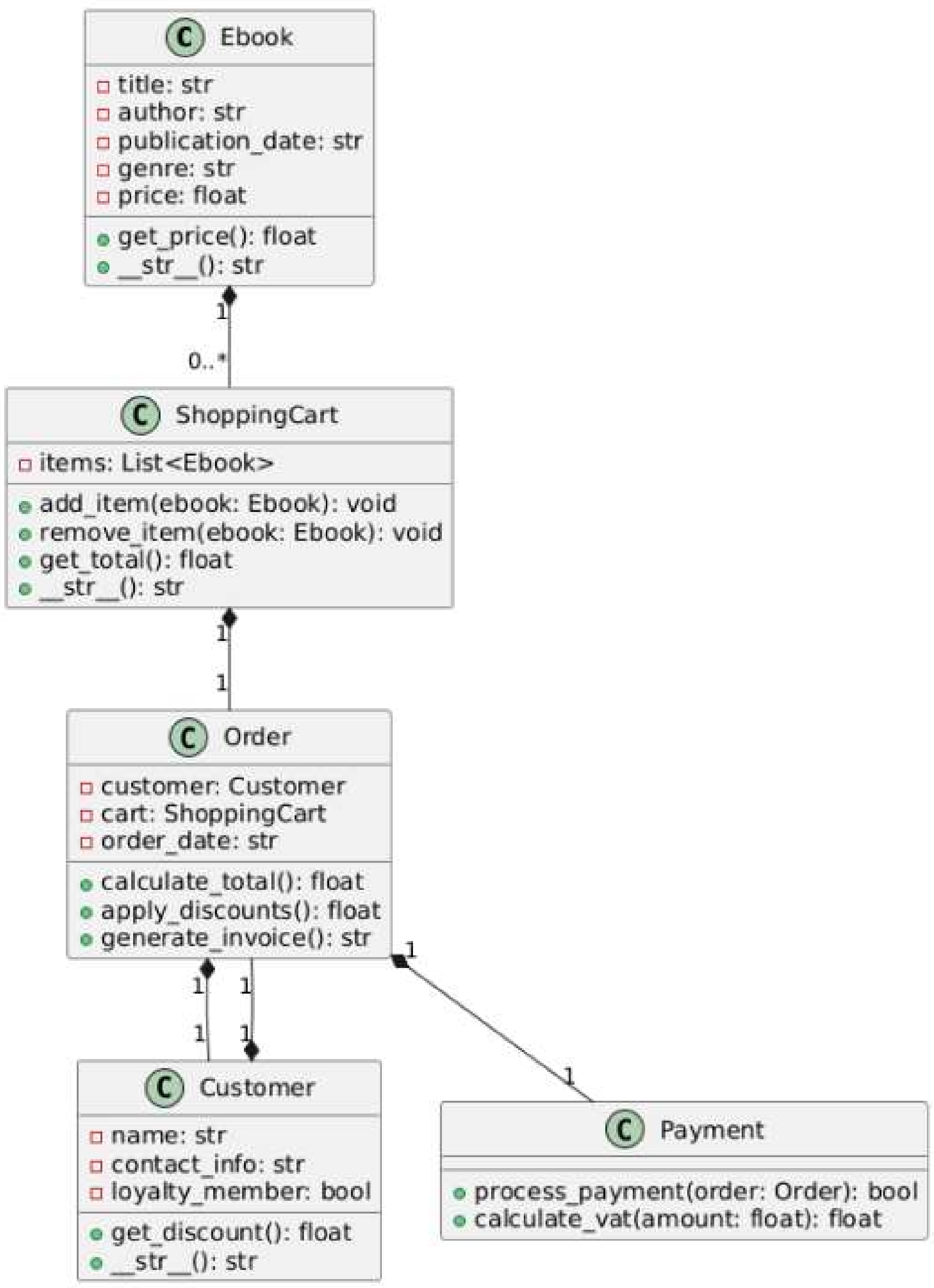
This report presents the development of an E-Bookstore Management System designed to improve the operational efficiency and user experience of The Great E-books Store. With the rapid growth of digital book sales, this system addresses the need for a streamlined, customer-centric solution that includes comprehensive catalog management, customer information handling, shopping cart functionality, and automated payment and invoicing processes. Developed using Object-Oriented Programming (OOP) principles, the system comprises key classes like Ebook, Customer, ShoppingCart, Order, and Payment, each fulfilling specific roles within the software. Additionally, the system supports a loyalty program and bulk discounts to enhance customer engagement. The modular and scalable design allows for future expansions and integrations, positioning the E-Bookstore Management System as a flexible solution for evolving business needs. Through UML class diagrams and Python code implementation, this project exemplifies an effective approach to developing a robust e-commerce management application.

II. Introduction

The E-Bookstore Management System is designed to meet the growing demand for seamless, efficient digital book purchasing and management experiences. In today’s digital age, online bookstores must be equipped with functionalities that enhance customer satisfaction, streamline transactions, and manage comprehensive catalogs of e-books. This project, created for The Great E-books Store, introduces a software solution that supports catalog management, customer account services, shopping cart functionality, and flexible payment options, including discounts and loyalty incentives. The system is built using Object-Oriented Programming (OOP) principles and is designed to be modular, scalable, and easy to maintain. The goal of the E-Bookstore Management System is not only to improve internal operations but also to create a user-friendly, rewarding experience for customers, from browsing books to completing purchases.

III. Description of Classes and Relationships

UML CLASS DIAGRAM



IV. Description of Classes and Relationships

1. **Ebook Class**:

- **Attributes**: Stores details about an e-book, such as title, author, publication_date, genre, and price.
- **Methods**:
    - get_price(): Returns the price of the e-book.
    - __str__(): Provides a string representation of the e-book.

2. **Customer Class**:
    - **Attributes**: Stores customer details including name, contact_info, and loyalty_member status.
    - **Methods**:
        - get_discount(): Calculates the discount rate for the customer based on loyalty status.
        - __str__(): Provides a string representation of the customer.
3. **ShoppingCart Class**:
    - **Attributes**: Holds a list of Ebook objects in items.
    - **Methods**:
        - add_item(ebook): Adds an Ebook to the cart.
        - remove_item(ebook): Removes an Ebook from the cart.
        - get_total(): Calculates the total price of items in the cart.
        - __str__(): Provides a string representation of the cart.
4. **Order Class**:
    - **Attributes**: Contains an instance of Customer, ShoppingCart, and order_date.
    - **Methods**:
        - calculate_total(): Calculates the total after discounts.
        - apply_discounts(): Applies customer and bulk purchase discounts.

        - generate_invoice(): Generates an invoice with pricing, discounts, and VAT.

5. **Payment Class**:
    - **Methods**:
        - process_payment(order): Simulates processing the payment for an Order.
        - calculate_vat(amount): Adds VAT to the total order amount.

Relationships

- **Composition**:
    - Order has a composition relationship with both ShoppingCart and Customer, meaning each Order depends on having a specific ShoppingCart and Customer.
- **Aggregation**:
    - ShoppingCart has an aggregation relationship with Ebook, where multiple Ebook objects can be added to a single cart but exist independently outside of it.
- **Association**:
    - Payment is associated with Order, processing the order's payment and VAT calculation.

V. Python Code

```
# Ebook.py
class Ebook:
    def __init__(self, title, author, publication_date, genre, price):
        self._title = title
        self._author = author
        self._publication_date = publication_date
        self._genre = genre
        self._price = price
    def get_price(self):
        return self._price

    def __str__(self):
        return f"Ebook(title={self._title}, author={self._author}, price={self._price})"
```

```
# customer.py
class Customer:
    def __init__(self, name, contact_info, loyalty_member=False):
        self._name = name
        self._contact_info = contact_info
        self._loyalty_member = loyalty_member

    def is_loyalty_member(self):
        return self._loyalty_member

    def __str__(self):
        return f"Customer(name={self._name}, loyalty_member={self._loyalty_member})"
```

```
# customer.py
class Customer:
    def __init__(self, name, contact_info, loyalty_member=False):
        self._name = name
        self._contact_info = contact_info
        self._loyalty_member = loyalty_member

    def is_loyalty_member(self):
        return self._loyalty_member

    def __str__(self):
        return f"Customer(name={self._name}, loyalty_member={self._loyalty_member})"
```

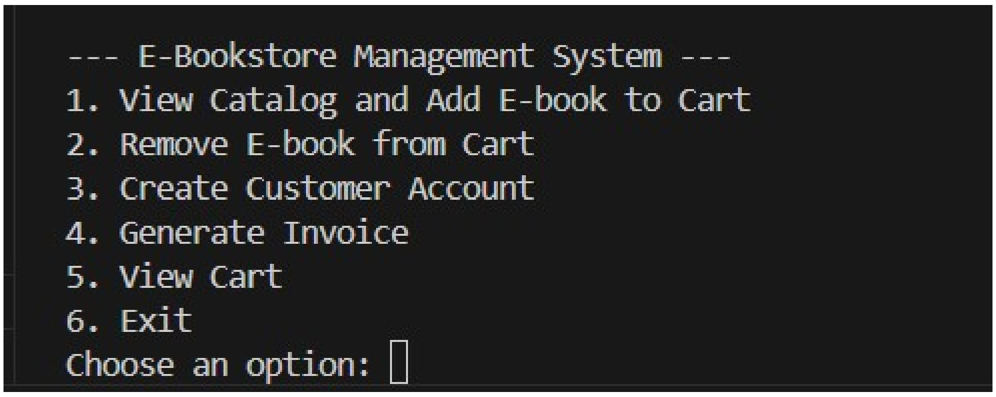
```
# shopping_cart.py
from ebook import Ebook

class ShoppingCart:
    def __init__(self):
        self._items = []

    def add_item(self, ebook):
        if isinstance(ebook, Ebook):
            self._items.append(ebook)

    def remove_item(self, ebook):
        if ebook in self._items:
            self._items.remove(ebook)

    def calculate_total(self):
        return sum(item.get_price() for item in self._items)

    def __str__(self):
        return f"ShoppingCart(items={[str(item) for item in self._items]})"
```

```
# discount.py
class Discount:
    LOYALTY_DISCOUNT = 0.10
    BULK_DISCOUNT = 0.20
    BULK_THRESHOLD = 5

    def apply_discount(self, total, loyalty_member=False, item_count=0):
        discount = 0
        if loyalty_member:
            discount += total * self.LOYALTY_DISCOUNT
        if item_count >= self.BULK_THRESHOLD:
            discount += total * self.BULK_DISCOUNT
        return total - discount
```

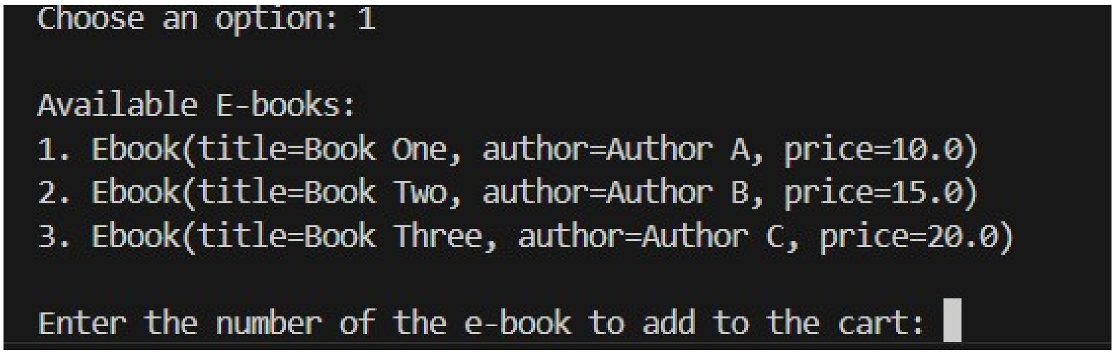
```
# order.py
from discount import Discount
from shopping_cart import ShoppingCart

class Order:
    VAT_RATE = 0.08

    def __init__(self, customer, cart):
        self._customer = customer
        self._cart = cart
        self._discount_handler = Discount()

    def calculate_total_with_discount_and_vat(self):
        item_count = len(self._cart._items)
        subtotal = self._cart.calculate_total()
        discounted_total = self._discount_handler.apply_discount(
            subtotal,
            loyalty_member=self._customer.is_loyalty_member(),
            item_count=item_count
        )
        return discounted_total + (discounted_total * self.VAT_RATE)

    def generate_invoice(self):
        total = self.calculate_total_with_discount_and_vat()
        return f"Invoice for {self._customer._name}: Total amount after discounts and
VAT is {total:.2f}"
```

```
# test_cases.py
from ebook import Ebook
from customer import Customer
from shopping_cart import ShoppingCart
from order import Order

# Initialize instances
cart = ShoppingCart()
customer = None

# Sample e-books for the store's catalog
catalog = [
    Ebook("Book One", "Author A", "2021-01-01", "Fiction", 10.0),
    Ebook("Book Two", "Author B", "2021-01-02", "Non-Fiction", 15.0),
    Ebook("Book Three", "Author C", "2022-03-15", "Science", 20.0)
]

def display_catalog():
    print("\nAvailable E-books:")
    for i, ebook in enumerate(catalog, start=1):
        print(f"{i}. {ebook}")

def add_ebook_to_cart():
    display_catalog()
    choice = int(input("\nEnter the number of the e-book to add to the cart: ")) - 1
    if 0 <= choice < len(catalog):
        cart.add_item(catalog[choice])
        print(f"{catalog[choice]._title} added to cart.")
    else:
        print("Invalid choice.")

def remove_ebook_from_cart():
    print("\nE-books in Cart:")
    for i, ebook in enumerate(cart._items, start=1):
        print(f"{i}. {ebook}")
    choice = int(input("\nEnter the number of the e-book to remove from the cart: ")) - 1
    if 0 <= choice < len(cart._items):
        removed_ebook = cart._items[choice]
        cart.remove_item(removed_ebook)
        print(f"{removed_ebook._title} removed from cart.")
    else:
        print("Invalid choice.")

def create_customer_account():
    global customer
    name = input("Enter customer name: ")
    contact_info = input("Enter contact info (email or phone): ")
    loyalty = input("Is the customer a loyalty member? (y/n): ").lower() == 'y'
    customer = Customer(name, contact_info, loyalty_member=loyalty)
    print(f"Customer account created for {name}.")

def generate_invoice():
    if not customer:
        print("Please create a customer account first.")
    elif not cart._items:
        print("The cart is empty. Add e-books to the cart before generating an invoice.")
    else:
        order = Order(customer, cart)
        print("\n" + order.generate_invoice())

def main():
    while True:
        print("\n--- E-Bookstore Management System ---")
        print("1. View Catalog and Add E-book to Cart")
        print("2. Remove E-book from Cart")
        print("3. Create Customer Account")
        print("4. Generate Invoice")
        print("5. View Cart")
        print("6. Exit")

        choice = input("Choose an option: ")

        if choice == '1':
            add_ebook_to_cart()
        elif choice == '2':
            remove_ebook_from_cart()
        elif choice == '3':
            create_customer_account()
        elif choice == '4':
            generate_invoice()
        elif choice == '5':
            print("\nShopping Cart:", cart)
        elif choice == '6':
            print("Exiting the system.")
            break
        else:
            print("Invalid option. Please try again.")
```

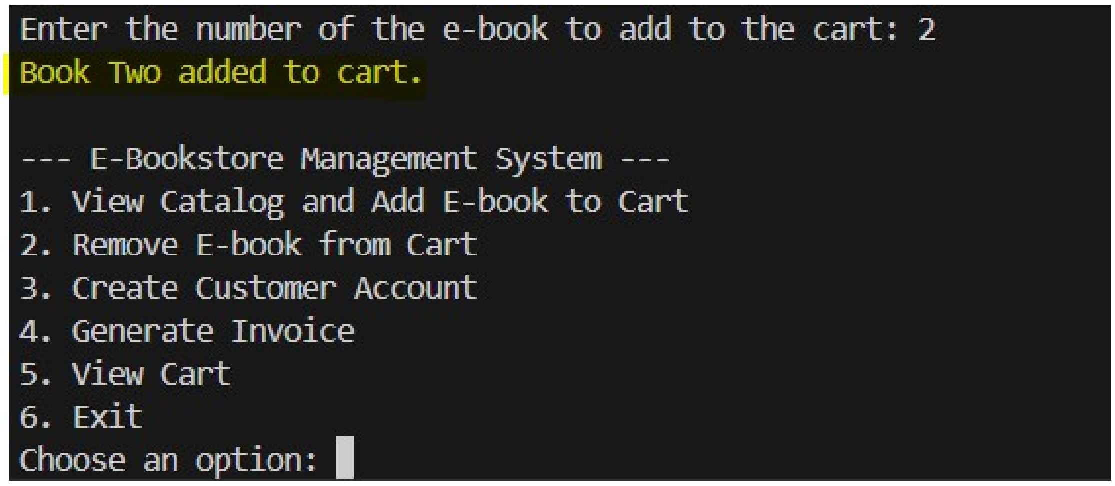
VI. Test



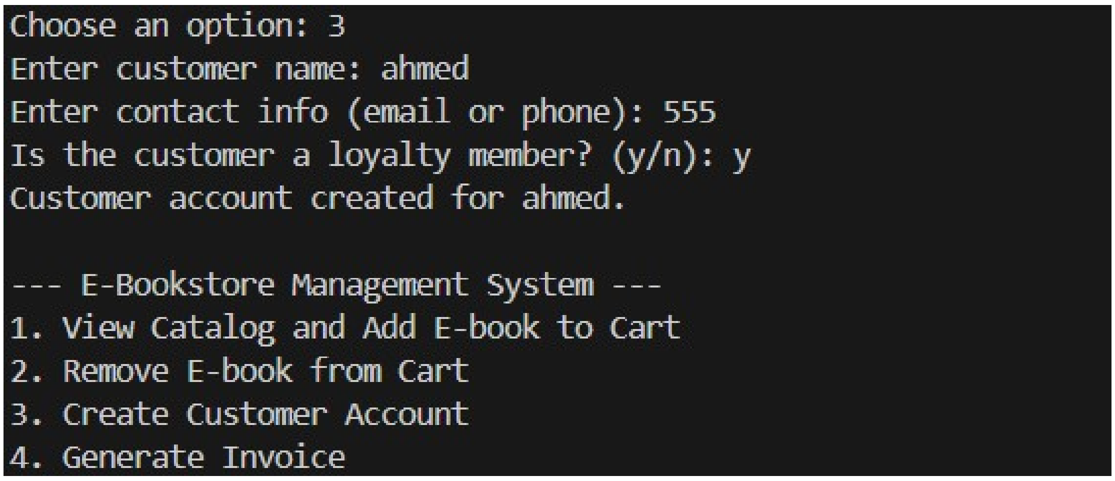
```
--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option: 
```



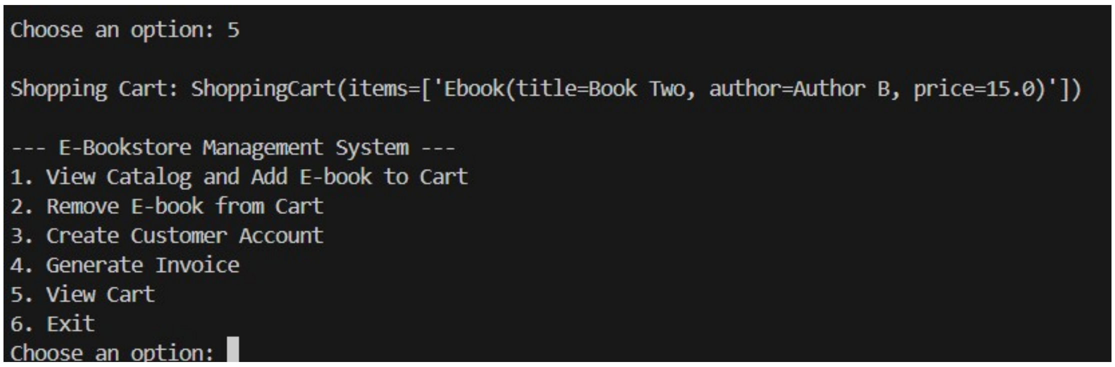
```
Choose an option: 1

Available E-books:
1. Ebook(title=Book One, author=Author A, price=10.0)
2. Ebook(title=Book Two, author=Author B, price=15.0)
3. Ebook(title=Book Three, author=Author C, price=20.0)

Enter the number of the e-book to add to the cart: 
```



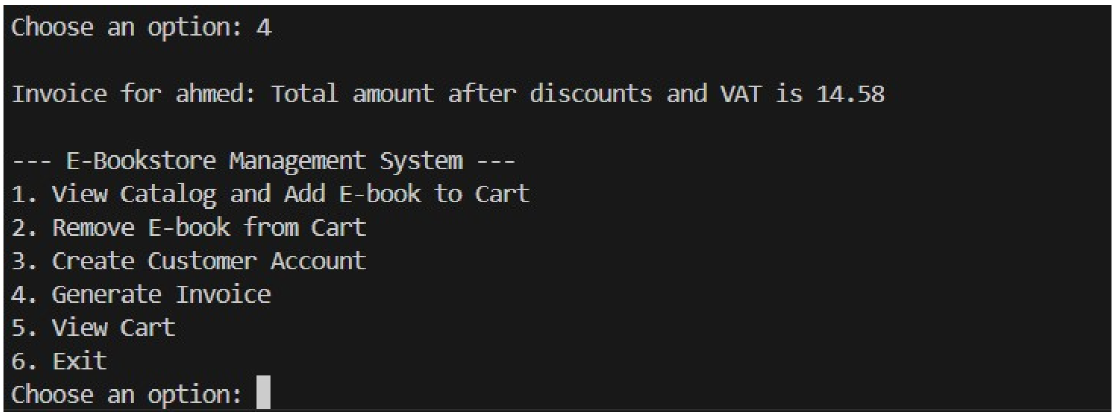
```
Enter the number of the e-book to add to the cart: 2
Book Two added to cart.

--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option: 
```



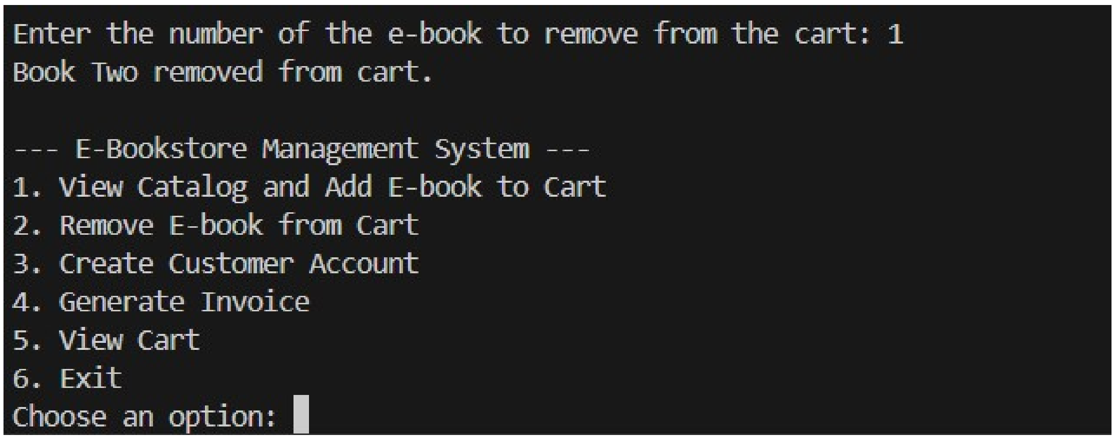
```
Choose an option: 3
Enter customer name: ahmed
Enter contact info (email or phone): 555
Is the customer a loyalty member? (y/n): y
Customer account created for ahmed.

--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
```



```
Choose an option: 5

Shopping Cart: ShoppingCart(items=['Ebook(title=Book Two, author=Author B, price=15.0)'])

--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option:
```



```
Choose an option: 4

Invoice for ahmed: Total amount after discounts and VAT is 14.58

--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option:
```



```
Enter the number of the e-book to remove from the cart: 1
Book Two removed from cart.

--- E-Bookstore Management System ---
1. View Catalog and Add E-book to Cart
2. Remove E-book from Cart
3. Create Customer Account
4. Generate Invoice
5. View Cart
6. Exit
Choose an option:
```

VII. Conclusion

The E-Bookstore Management System fulfills the primary requirements for managing an online bookstore by integrating key functionalities, including catalog management, customer information processing, a dynamic shopping cart, and streamlined payment and invoicing. This project has been implemented using an object-oriented approach, as reflected in the UML class diagram and Python code. Through classes such as Ebook, Customer, ShoppingCart, Order, and Payment, the system encapsulates the bookstore's essential components while maintaining modularity and scalability. This approach allows for future expansions, such as adding new payment options or expanding discount structures. By enhancing customer interactions and supporting efficient e-commerce transactions, this E-Bookstore Management System represents a significant step toward modernizing the Great E-books Store's operations and positioning it to compete effectively in the digital market.

VIII. References

[1]. Zhang, Y. (2008, August). Research on Personalized E-Bookstore Service System. In *2008 ISECS International Colloquium on Computing, Communication, Control, and Management* (Vol. 3, pp. 259-262). IEEE.

[2]. Aref, M. M., Elbagoury, B. M., & Hassanin, W. (2022). Designing an Intelligent Agents for E-Bookstore System Web-Based System. In *Proceedings of Sixth International Congress on Information and Communication Technology: ICICT 2021, London, Volume 4* (pp. 321-330). Springer Singapore.

[3]. Fan, J. C. (2006, October). Trust and Electronic Commerce-A Test of an E-Bookstore. In *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)* (pp. 110-117). IEEE.

[4]. Choy, W. Y. (2001). *Electronic bookstore/Choy Wai Yin* (Doctoral dissertation, University of Malaya).