Appendix 1 main.py

main.py

```python
import logging

from services.user_service import UserService
from services.product_service import ProductService
from services.order_service import OrderService
from controllers.authentication_controller import AuthenticationController
from controllers.product_controller import ProductController
from controllers.order_controller import OrderController
from views.cli_view import CLIView

"""
Entry point of the Inventory Management System (IMS).
"""

# Configuration of log message.
logging.basicConfig(
    filename='data/system.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)


def main():
    """
    Initialises and starts the IMS application.
    """
    # Initialise services
    user_service = UserService()
    product_service = ProductService()
    order_service = OrderService()

    # Initialise controllers
    auth_controller = AuthenticationController(user_service)
    product_controller = ProductController(product_service)
    order_controller = OrderController(order_service, product_service)

    # Initialise views
    cli_view = CLIView(auth_controller, product_controller, order_controller)

    # Start the application
    cli_view.main_menu()


if __name__ == '__main__':
    main()
```

Appendix 2 company_validation_api.py

company_validation_api.py

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

# Reference: https://pypi.org/project/Flask

"""
Assumed only authorised Companies allow to create customer account.
Simulates an API for validating company IDs.
"""

valid_company_ids = ['COMPANY123', 'COMPANY456', 'COMPANY789']


@app.route('/validate_company', methods=['GET'])
def validate_company():
    """
    Validates a company ID against a list of valid IDs.
    return JSON response indicating whether the company ID is valid.
    """
    company_id = request.args.get('company_id')
    if company_id in valid_company_ids:
        return jsonify({'valid': True}), 200  # OK Status code
    else:
        return jsonify({'valid': False}), 404  # Not Found Status code


if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)

```

Appendix 3 authentication_controller.py

authentication_controller.py

```python
import logging
import time
import re
import requests
import bcrypt

from models.user import Clerk, Customer

"""
Reference:
https://pypi.org/project/requests
https://pypi.org/project/bcrypt
"""


class AuthenticationController:
    """
    Controller for handling user authentication and security features,
    manages user login, registration and account lockout.
    """

    def __init__(self, user_service):
        """
        Initialises the AuthenticationController.
        """
        self.user_service = user_service  # Provides user data operations.
        self.MAX_ATTEMPTS = 3  # Default the max allowed failed logins.
        self.LOCKOUT_TIME = 300  # Default how long (seconds) remains locked.

    def login(self, username, password):
        """
        Authenticates a user.
        Implements account lockout after multiple failed attempts.
        """
        current_time = time.time()
        user = self.user_service.get_user_by_username(username)

        if user:
            # Account lockout mechanism
            if user.is_locked:
                time_since_locked = current_time - user.locked_time
                if time_since_locked < self.LOCKOUT_TIME:
                    remaining_lock_time = int(
                        self.LOCKOUT_TIME - time_since_locked
                        )
                    minutes, seconds = divmod(remaining_lock_time, 60)
                    print(f"Account is locked. Please try again "
                          f"after {minutes} minute(s) "
                          f"and {seconds} second(s).")
                    return None
                else:
                    # Unlock the account after lockout duration has passed
                    user.is_locked = False
                    user.login_attempts = 0
                    user.locked_time = None
                    self.user_service.save_users()
```

```python

            if user.check_password(password):
                # Successful login
                user.login_attempts = 0
                self.user_service.save_users()
                logging.info(f"{username} logged in successfully.")

                # Perform MFA (simulation)
                if self.perform_mfa(user):
                    return user
                else:
                    print("Multi-Factor Authentication failed.")
                    return None
            else:
                # Invalid login attempt
                user.login_attempts += 1
                remaining_attempts = self.MAX_ATTEMPTS - user.login_attempts
                if remaining_attempts > 0:
                    print(f"Invalid username or password. "
                          f"You have {remaining_attempts} "
                          f"attempt(s) remaining.")
                else:
                    # Lock the account
                    user.is_locked = True
                    user.locked_time = current_time
                    logging.warning(f"{username} account locked "
                                    f"due to multiple failed attempts.")
                    print(f"Account locked due to multiple failed attempts. "
                          f"Please try again "
                          f"after {int(self.LOCKOUT_TIME / 60)} minutes.")
                self.user_service.save_users()
                return None
        else:
            print("Invalid username or password.")
            return None
```

```python

    def perform_mfa(self, user):
        """
        Simulates Multi-Factor Authentication.
        """
        # Generating a code for simulation purposes.
        mfa_code = '123456'
        input_code = input(
            "Enter the MFA code sent to your registered device: "
            )
        if input_code == mfa_code:
            return True
        else:
            return False

    def register_customer(self, username, password, company_id):
        """
        Registers a new customer after validating inputs and company ID.
        """
        if (
            self.validate_username(username)
            and self.validate_password(password)
        ):
            # Validate company ID via API
            try:
                response = requests.get(
                    'http://localhost:5000/validate_company',
                    params={'company_id': company_id}
                    )
                if (
                    response.status_code == 200
                    and response.json().get('valid')
                ):
                    if not self.user_service.get_user_by_username(username):
                        # Hash the password with bcrypt
                        hashed_password = bcrypt.hashpw(
                         password.encode('utf-8'),
                         bcrypt.gensalt()
                         )
                        new_customer = Customer(username,
                                                hashed_password, company_id)
                        self.user_service.add_user(new_customer)
                        logging.info(f"New customer registered: {username}")
                        print("Registration successful.")
                    else:
                        print("Username already exists.")
                else:
                    print("Invalid company ID.")
            except requests.RequestException as e:
                print("Unable to connect to the company validation service. "
                      "Please try again later.")
                logging.error(f"Error connecting to "
                              f"company validation API: {e}")
        else:
            print("Invalid username or password format.")
```

```python

        def register_clerk(self, username, password):
            """
            Registers a new clerk after validating inputs.
            """
            if (
                self.validate_username(username)
                and self.validate_password(password)
            ):
                if not self.user_service.get_user_by_username(username):
                    # Hash the password with bcrypt
                    hashed_password = bcrypt.hashpw(
                        password.encode('utf-8'), bcrypt.gensalt()
                        )
                    new_clerk = Clerk(username, hashed_password)
                    self.user_service.add_user(new_clerk)
                    logging.info(f"New clerk registered: {username}")
                    print("Clerk registered successfully.")
                    return True
                else:
                    print("Username already exists.")
            else:
                print("Invalid username or password format.")
            return False

        def validate_username(self, username):
            """
            Validates the username using regex.
            Rule:
             Only alphanumeric characters (A-Z, a-z, 0-9)
             Min. 5 and max. 20 characters in length
             No space at the beginning or end
            """
            pattern = re.compile(r'^[A-Za-z0-9]{5,20}$')
            return pattern.match(username)

        def validate_password(self, password):
            """
            Validates the password for strength.
            Rule:
             At least 8 characters long
             At least one uppercase letter ([A-Z])
             At least one lowercase letter ([a-z])
             At least one digit ([0-9])
             At least one special character from the set [@#$%^&+=]
            """
            if len(password) < 8:
                return False
            if not re.search(r'[A-Z]', password):
                return False
            if not re.search(r'[a-z]', password):
                return False
            if not re.search(r'[0-9]', password):
                return False
            if not re.search(r'[@#$%^&+=]', password):
                return False
            return True

```

Appendix 4 order_controller.py

order_controller.py

```python
from models.order import Order


class OrderController:
    """
    Manages order processing logic,
    handles placing orders and listing customer orders.
    """

    def __init__(self, order_service, product_service):
        """
        Initialises the OrderController.
        """
        self.order_service = order_service  # Order data operations.
        self.product_service = product_service  # Product data operation.

    def place_order(self, customer_username, items):
        """
        Places a new order.
        """
        total_price = 0
        for product_id, quantity in items:
            product = self.product_service.get_product_by_id(product_id)
            if not product or product.stock_quantity < quantity:
                print(
                    f"Product '{product_id}' is not available in the "
                    f"required quantity."
                    )
                return False
            total_price += product.price * quantity

        order = Order(customer_username, items, total_price)
        self.order_service.add_order(order)

        # Update stock quantities
        for product_id, quantity in items:
            product = self.product_service.get_product_by_id(product_id)
            product.stock_quantity -= quantity
        self.product_service.save_products()

        print("Order placed successfully.")
        return True

    def list_customer_orders(self, customer_username):
        """
        Lists all orders placed by a customer.
        """
        return self.order_service.list_orders_by_customer(customer_username)

```

Appendix 5 product_controller.py

product_controller.py

```python
from models.product import Product


class ProductController:
    """
    Manages product-related operations,
    handles CRUD operations for products.
    """

    def __init__(self, product_service):
        """
        Initialises the ProductController.
        """
        self.product_service = product_service  # Product data operation.

    def add_product(self, product_id, name, category, price, stock_quantity):
        """
        Adds a new product to the inventory via product_service
        """
        if self.product_service.get_product_by_id(product_id):
            print(f"Product with ID '{product_id}' already exists.")
            return False
        product = Product(product_id, name, category, price, stock_quantity)
        self.product_service.add_product(product)
        print(f"Product '{name}' added successfully.")
        return True

    def update_stock(self, product_id, quantity):
        """
        Updates the stock quantity of a product.
        """
        return self.product_service.update_stock(product_id, quantity)

    def delete_product(self, product_id):
        """
        Deletes a product from the inventory.
        """
        if self.product_service.delete_product(product_id):
            print(f"Product with ID '{product_id}' deleted successfully.")
            return True
        else:
            print(f"No product found with ID '{product_id}'.")
            return False

    def list_products(self):
        """
        Lists all products in the inventory.
        """
        return self.product_service.products

```

Appendix 6 order.py

order.py

```python
import uuid
from datetime import datetime


class Order:
    """
    Represents a customer's order.
    """

    def __init__(self, customer_username, items, total_price):
        """
        Initialises an Order object.
        """
        self.order_id = str(uuid.uuid4())  # Unique UUID for the order.
        self.customer_username = customer_username
        self.items = items  # list of product_id, quantity.
        self.total_price = total_price  # The total cost of all items.
        # Timestamp (YYYY-MM-DD HH:MM:SS) when the order was created.
        self.order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

```

Appendix 7 product.py

product.py

```python
class Product:
    """
    Represents a product in the inventory.
    """

    def __init__(self, product_id, name, category, price, stock_quantity):
        """
        Initialises a Product object.
        """
        self.product_id = product_id  # The unique identifier for product.
        self.name = name  # The product name
        self.category = category  # The category to which the product belongs.
        self.price = price  # The price of the product.
        self.stock_quantity = stock_quantity  # Qty. of product in stock.

```

Appendix 8 user.py

user.py

```python
import bcrypt


class User:
    """
    Base class representing a user.
    """

    def __init__(self, username, password, role=None,
                 login_attempts=0,
                 is_locked=False, locked_time=None
                 ):
        """
        Initialises a User object.
        """
        self.username = username
        self.role = role  # Role of the user.
        self.password = password  # Store bcrypt-hashed password as bytes
        self.login_attempts = login_attempts  # No. of failed login attempts.
        self.is_locked = is_locked  # Indicates if the user is locked out
        self.locked_time = locked_time  # Timestamp when account was locked

    def check_password(self, plain_password):
        """
        Validates the input password against
        the stored bcrypt-hashed password.
        Return True if the password matches.
        """
        return bcrypt.checkpw(plain_password.encode('utf-8'), self.password)


class Admin(User):
    """Admin inheriting from User."""
    def __init__(self, username, password,
                 login_attempts=0,
                 is_locked=False, locked_time=None):
        """
        Initialises an Admin object.
        """
        super().__init__(username, password, role='Admin',
                         login_attempts=login_attempts,
                         is_locked=is_locked, locked_time=locked_time
                         )


class Clerk(User):
    """Clerk inheriting from User."""
    def __init__(self, username, password,
                 login_attempts=0,
                 is_locked=False, locked_time=None):
        """
        Initialises a Clerk object.
        """
        super().__init__(username, password, role='Clerk',
                         login_attempts=login_attempts,
                         is_locked=is_locked, locked_time=locked_time
                         )


class Customer(User):
    """
    Customer inheriting from User.
    """
    def __init__(self, username, password, company_id,
                 login_attempts=0, is_locked=False,
                 locked_time=None):
        """
        Initialises a Customer object.
        """
        super().__init__(
            username, password, role='Customer',
            login_attempts=login_attempts, is_locked=is_locked,
            locked_time=locked_time
            )
        self.company_id = company_id  # Company ID associated with customer.

```

Appendix 9 order_service.py

order_service.py

```python
import json
import os
from models.order import Order


class OrderService:
    """
    Handles order data operations,
    manages loading, saving, adding and
    retrieving orders to/from persistent storage.
    """

    DATA_FILE = 'data/orders.json'  # Orders data file.

    def __init__(self):
        """
        Initialises the OrderService by
        loading orders data from storage.
        """
        self.orders = self.load_orders()

    def load_orders(self):
        """
        Loads orders from the JSON file.
        """
        if not os.path.exists(self.DATA_FILE):
            return []

        with open(self.DATA_FILE, 'r') as file:
            order_dicts = json.load(file)

        orders = []
        for order_data in order_dicts:
            customer_username = order_data['customer_username']
            items = order_data['items']
            total_price = order_data['total_price']

            order = Order(customer_username, items, total_price)
            order.order_id = order_data.get('order_id', order.order_id)
            order.order_date = order_data.get('order_date', order.order_date)

            orders.append(order)
        return orders

    def save_orders(self):
        """
        Saves orders to the JSON file.
        """
        order_dicts = [order.__dict__ for order in self.orders]
        with open(self.DATA_FILE, 'w') as file:
            json.dump(order_dicts, file)

    def add_order(self, order):
        """
        Adds a new order.
        """
        self.orders.append(order)
        self.save_orders()

    def get_order_by_id(self, order_id):
        """
        Retrieves an order by its ID.
        """
        for order in self.orders:
            if order.order_id == order_id:
                return order
        return None

    def list_orders_by_customer(self, customer_username):
        """
        Retrieves all orders placed by a specific customer.
        """
        return [order for order in self
                .orders if order.customer_username == customer_username]

```

Appendix 10 product_service.py

product_service.py

```python
import os
import json
from models.product import Product


class ProductService:
    """
    Manages products, including
    adding new products, updating existing products,
    and saving/loading products to/from persistent storage.
    """

    PRODUCTS_FILE = 'data/products.json'  # Product data file.

    def __init__(self):
        """
        Initialises the ProductService by
        loading existing products from storage.
        """
        self.products = self.load_products()

    def add_product(self, product):
        """
        Adds a new product to the inventory.
        """
        if self.get_product_by_id(product.product_id):
            print(f"Product with ID '{product.product_id}' already exists.")
            return False
        self.products.append(product)
        self.save_products()
        return True

    def update_stock(self, product_id, quantity):
        """
        Updates the stock quantity of a product.
        """
        product = self.get_product_by_id(product_id)
        if product:
            product.stock_quantity = quantity
            self.save_products()
            return True
        else:
            print(f"No product found with ID '{product_id}'.")
            return False

    def get_product_by_id(self, product_id):
        """
        Retrieves a product by its ID.
        """
        for product in self.products:
            if product.product_id == product_id:
                return product
        return None
```

```python

    def save_products(self):
        """
        Saves current products to JSON file for persistent storage.
        """
        products_data = []
        for product in self.products:
            product_data = {
                'product_id': product.product_id,
                'name': product.name,
                'category': product.category,
                'price': product.price,
                'stock_quantity': product.stock_quantity
            }
            products_data.append(product_data)

        os.makedirs('data', exist_ok=True)
        with open(self.PRODUCTS_FILE, 'w') as f:
            json.dump(products_data, f, indent=4)

    def load_products(self):
        """
        Loads products from a JSON file and returns a list of Product objects.
        """
        try:
            with open(self.PRODUCTS_FILE, 'r') as f:
                products_data = json.load(f)

            products = []
            for product_data in products_data:
                product = Product(
                    product_id=product_data['product_id'],
                    name=product_data['name'],
                    category=product_data['category'],
                    price=product_data['price'],
                    stock_quantity=product_data['stock_quantity']
                )
                products.append(product)
            return products
        except FileNotFoundError:
            return []
        except json.JSONDecodeError as e:
            print("An error occurred while loading products:", str(e))
            return []

    def delete_product(self, product_id):
        """
        Deletes a product from the inventory.
        """
        product = self.get_product_by_id(product_id)
        if product:
            self.products.remove(product)
            self.save_products()
            return True
        else:
            return False

```

Appendix 11 user_service.py

user_service.py

```python
import os
import json
from models.user import Admin, Clerk, Customer


class UserService:
    """
    Manages user accounts, including
    adding new users, authenticating existing users,
    and saving/loading users to/from persistent storage.
    """

    USERS_FILE = 'data/users.json'  # Users data file.

    def __init__(self):
        """
        Initialises the UserService by
        loading existing users from storage.
        """
        self.users = self.load_users()

    def add_user(self, user):
        """
        Adds a new user to the system.
        """
        if self.get_user_by_username(user.username):
            print(f"User with username '{user.username}' already exists.")
            return False

        self.users.append(user)
        self.save_users()
        return True

    def get_user_by_username(self, username):
        """
        Retrieves a user object by username.
        """
        for user in self.users:
            if user.username == username:
                return user
        return None
```

```python

    def save_users(self):
        """
        Saves the current list of users to JSON file for persistent storage.
        The hashed password is stored directly as a UTF-8 string in JSON.
        """
        users_data = []
        for user in self.users:
            # Convert bytes hashed password to a UTF-8 string before saving
            hashed_pw_str = (user.password.decode('utf-8')
                             if isinstance(user.password, bytes)
                             else user.password)

            user_data = {
                'username': user.username,
                'password': hashed_pw_str,
                'role': user.role,
                'login_attempts': user.login_attempts,
                'is_locked': user.is_locked,
                'locked_time': user.locked_time
            }

            if user.role == 'Customer':
                user_data['company_id'] = user.company_id

            users_data.append(user_data)

        os.makedirs('data', exist_ok=True)
        with open(self.USERS_FILE, 'w') as f:
            json.dump(users_data, f, indent=4)

    def load_users(self):
        """
        Loads users from JSON file and returns a list of User objects.
        """
        try:
            with open(self.USERS_FILE, 'r') as f:
                users_data = json.load(f)

            users = []
            for user_data in users_data:
                username = user_data['username']
                role = user_data['role']
                hashed_password_str = user_data['password']
                login_attempts = user_data.get('login_attempts', 0)
                is_locked = user_data.get('is_locked', False)
                locked_time = user_data.get('locked_time', None)

                # Convert the hashed password string back to bytes
                hashed_password = hashed_password_str.encode('utf-8')

                if role == 'Admin':
                    user = Admin(
                        username,
                        hashed_password,
                        login_attempts=login_attempts,
                        is_locked=is_locked,
                        locked_time=locked_time
                    )
```

```python
                    elif role == 'Clerk':
                        user = Clerk(
                            username,
                            hashed_password,
                            login_attempts=login_attempts,
                            is_locked=is_locked,
                            locked_time=locked_time
                        )
                    elif role == 'Customer':
                        company_id = user_data.get('company_id', '')
                        user = Customer(
                            username,
                            hashed_password,
                            company_id=company_id,
                            login_attempts=login_attempts,
                            is_locked=is_locked,
                            locked_time=locked_time
                        )
                    else:
                        # Unknown role, skip user
                        continue

                    users.append(user)

                return users
            except FileNotFoundError:
                # No users file exists, return an empty list
                return []
            except json.JSONDecodeError as e:
                print("An error occurred while loading users:", str(e))
                return []

        def remove_user(self, username):
            """
            Removes a user from the system based on the username.
            """
            user = self.get_user_by_username(username)
            if user:
                self.users.remove(user)
                self.save_users()
                print(f"User '{username}' removed successfully.")
            else:
                print(f"No user found with username '{username}'.")

        def unlock_user_account(self, username):
            """
            Unlocks a user's account by resetting login attempts and lock status.
            """
            user = self.get_user_by_username(username)
            if user:
                user.is_locked = False
                user.login_attempts = 0
                self.save_users()
                print(f"User '{username}' account has been unlocked.")
            else:
                print(f"No user found with username '{username}'.")

```

Appendix 12 cli_view.py

cli_view.py

```python
class CLIView:
    """
    Command-Line Interface view for the application.
    """

    def __init__(self, auth_controller, product_controller, order_controller):
        """
        Initialises the CLIView with controllers.
        """
        self.auth_controller = auth_controller
        self.product_controller = product_controller
        self.order_controller = order_controller

    def main_menu(self):
        """
        Displays the main menu options (Login, Register, Exit)
        and collects user input in a loop until the user chooses to exit.
        """
        while True:
            print("\n=== Welcome to IMS ===")
            print("1. Login")
            print("2. Register as Customer")
            print("3. Exit")
            choice = input("Please select an option: ")

            if choice == '1':
                self.login()
            elif choice == '2':
                self.register_customer()
            elif choice == '3':
                print("Goodbye!")
                break
            else:
                print("Invalid option. Please try again.")

    def login(self):
        """
        Handles user login process.
        """
        print("\n--- Login ---")
        username = input("Username: ")
        password = input("Password: ")
        user = self.auth_controller.login(username, password)

        if user:
            print(f"\nWelcome, {user.username}!")
            if user.role == 'Admin':
                self.admin_menu(user)
            elif user.role == 'Clerk':
                self.clerk_menu()
            elif user.role == 'Customer':
                self.customer_menu(user)
        else:
            print("Login failed.")
```

```python

    def register_customer(self):
        """
        Handles customer registration process.
        """
        print("\n--- Customer Registration ---")
        username = input("Choose a username: ")
        password = input("Enter a password: ")
        company_id = input("Enter your company ID: ")
        if self.auth_controller.register_customer(
            username, password, company_id
        ):
            print("You can now log in with your new account.")

    def admin_menu(self, admin_user):
        """
        Displays the admin menu options and handles user input
        """
        while True:
            print("\n--- Admin Menu ---")
            print("1. Manage Users")
            print("2. Manage Products")
            print("3. View System Logs")
            print("4. Logout")
            choice = input("Please select an option: ")

            if choice == '1':
                self.manage_users()
            elif choice == '2':
                self.manage_products()
            elif choice == '3':
                self.view_system_logs()
            elif choice == '4':
                print("Logging out.")
                break
            else:
                print("Invalid option. Please try again.")
```

```python

    def clerk_menu(self):
        """
        Displays the clerk menu options and handles user input.
        """
        while True:
            print("\n--- Clerk Menu ---")
            print("1. Manage Products")
            print("2. Logout")
            choice = input("Please select an option: ")

            if choice == '1':
                self.manage_products()
            elif choice == '2':
                print("Logging out.")
                break
            else:
                print("Invalid option. Please try again.")

    def customer_menu(self, customer_user):
        """
        Displays the customer menu options and handles user input.
        """
        while True:
            print("\n--- Customer Menu ---")
            print("1. Place Order")
            print("2. View Order History")
            print("3. Logout")
            choice = input("Please select an option: ")

            if choice == '1':
                self.place_order(customer_user)
            elif choice == '2':
                self.view_order_history(customer_user)
            elif choice == '3':
                print("Logging out.")
                break
            else:
                print("Invalid option. Please try again.")

    # Define methods for managing users, products, placing orders, etc.
    """
    The system focuses on security measures;
    not all methods have been developed due to resource limitations,
    e.g. password reset flow,
    wrap stock input in a try/except to handle invalid integer
    """
```

```python

    def manage_users(self):
        """
        Handles user management options.
        """
        while True:
            print("\n--- Manage Users ---")
            print("1. View All Users")
            print("2. Add Clerk")
            print("3. Remove User")
            print("4. Unlock User Account")
            print("5. Back to Admin Menu")
            choice = input("Please select an option: ")

            if choice == '1':
                self.view_all_users()
            elif choice == '2':
                self.add_clerk()
            elif choice == '3':
                self.remove_user()
            elif choice == '4':
                self.unlock_user_account()
            elif choice == '5':
                break
            else:
                print("Invalid option. Please try again.")

    def view_all_users(self):
        """
        Displays all users in the system.
        """
        print("\n--- All Users ---")
        users = self.auth_controller.user_service.users
        for user in users:
            print(f"Username: {user.username}, "
                  f"Role: {user.role}, "
                  f"Locked: {user.is_locked}"
                  )

    def add_clerk(self):
        """
        Handles the process of adding a new clerk
        """
        print("\n--- Add Clerk ---")
        username = input("Enter clerk username: ")
        password = input("Enter clerk password: ")
        if self.auth_controller.register_clerk(username, password):
            print("Clerk added successfully.")
```

```python

    def remove_user(self):
        """
        Handles the removal of a user.
        """
        print("\n--- Remove User ---")
        username = input("Enter username to remove: ")
        self.auth_controller.user_service.remove_user(username)

    def unlock_user_account(self):
        """
        Handles unlocking a user account.
        """
        print("\n--- Unlock User Account ---")
        username = input("Enter username to unlock: ")
        self.auth_controller.user_service.unlock_user_account(username)

    def view_system_logs(self):
        """
        Displays system logs.
        """
        print("\n--- System Logs ---")
        try:
            with open('data/system.log', 'r') as log_file:
                logs = log_file.read()
                print(logs)
        except FileNotFoundError:
            print("No system log file found.")

    def manage_products(self):
        """
        Handles product management options.
        """
        while True:
            print("\n--- Manage Products ---")
            print("1. View All Products")
            print("2. Add Product")
            print("3. Update Product Stock")
            print("4. Delete Product")
            print("5. Back to Previous Menu")
            choice = input("Please select an option: ")

            if choice == '1':
                self.view_all_products()
            elif choice == '2':
                self.add_product()
            elif choice == '3':
                self.update_product_stock()
            elif choice == '4':
                self.delete_product()
            elif choice == '5':
                break
            else:
                print("Invalid option. Please try again.")
```

```python

    def view_all_products(self):
        """
        Displays all products in the inventory.
        """
        print("\n--- All Products ---")
        products = self.product_controller.list_products()
        for product in products:
            print(
                f"ID: {product.product_id}, Name: {product.name}, "
                f"Category: {product.category}, Price: {product.price}, "
                f"Stock: {product.stock_quantity}"
            )

    def add_product(self):
        """
        Handles the process of adding a new product.
        """
        print("\n--- Add Product ---")
        product_id = input("Enter product ID: ")
        name = input("Enter product name: ")
        category = input("Enter product category: ")
        price = float(input("Enter product price (numeric only): "))
        stock_quantity = int(input("Enter stock quantity (integer only): "))
        self.product_controller.add_product(
            product_id, name, category, price, stock_quantity
            )

    def update_product_stock(self):
        """
        Handles updating the stock quantity of a product.
        """
        print("\n--- Update Product Stock ---")
        product_id = input("Enter product ID: ")
        quantity = int(input("Enter new stock quantity: "))
        self.product_controller.update_stock(product_id, quantity)

    def delete_product(self):
        """
        Handles the deletion of a product.
        """
        print("\n--- Delete Product ---")
        product_id = input("Enter product ID to delete: ")
        self.product_controller.delete_product(product_id)
```

```python

        def place_order(self, customer_user):
            """
            Handles the process of placing an order.
            """
            print("\n--- Place Order ---")
            products = self.product_controller.list_products()
            print("Available Products:")
            for product in products:
                print(f"ID: {product.product_id}, "
                      f"Name: {product.name}, "
                      f"Price: {product.price}, "
                      f"Stock: {product.stock_quantity}"
                      )

            items = []
            while True:
                product_id = input(
                    "Enter product ID to add to cart (or 'done' to finish): "
                    )
                if product_id.lower() == 'done':
                    print("\nThe statement will be sent to your Company")
                    break
                quantity = int(input("Enter quantity: "))
                items.append((product_id, quantity))

            if items:
                self.order_controller.place_order(customer_user.username, items)
            else:
                print("No items selected.")

        def view_order_history(self, customer_user):
            """
            Displays the order history for a customer.
            """
            print("\n--- Order History ---")
            orders = self.order_controller.list_customer_orders(
                customer_user.username
                )
            if orders:
                for order in orders:
                    print(f"Order ID: {order.order_id}, "
                          f"Date: {order.order_date}, "
                          f"Total Price: {order.total_price}"
                          )
                    for item in order.items:
                        product_id, quantity = item
                        product = (
                            self.product_controller.product_service
                            .get_product_by_id(product_id)
                            )
                        if product:
                            print(
                                f"Product: {product.name}, "
                                f"Quantity: {quantity}, "
                                f"Price: {product.price}"
                                )
            else:
                print("You have no previous orders.")

```

Appendix 13 populate_data.py

populate_data.py

```python
"""
Utility script that creates initial users (Admin, Clerk, Customer)
and a sample set of products.
Writes data to JSON files in the "data" directory.
"""

# Reference: https://pypi.org/project/bcrypt

import os
import bcrypt

from services.user_service import UserService
from services.product_service import ProductService
from models.user import Admin, Clerk, Customer
from models.product import Product


def populate_users():
    """
    Creates default users and saves them using UserService.
    """
    user_service = UserService()

    # Define passwords that meet the security policy
    admin_password_plain = 'Admin@1234'
    clerk_password_plain = 'Clerk@1234'
    customer_password_plain = 'Customer@1234'

    # Hash passwords with bcrypt
    admin_password_hashed = bcrypt.hashpw(
        admin_password_plain.encode('utf-8'),
        bcrypt.gensalt()
        )
    clerk_password_hashed = bcrypt.hashpw(
        clerk_password_plain.encode('utf-8'),
        bcrypt.gensalt()
        )
    customer_password_hashed = bcrypt.hashpw(
        customer_password_plain.encode('utf-8'),
        bcrypt.gensalt()
        )

    # Create users
    admin = Admin(username='admin',
                  password=admin_password_hashed,
                  login_attempts=0,
                  is_locked=False, locked_time=None)
    clerk = Clerk(username='clerk',
                  password=clerk_password_hashed,
                  login_attempts=0,
                  is_locked=False, locked_time=None)
    customer = Customer(username='customer',
                        password=customer_password_hashed,
                        company_id='COMPANY123',
                        login_attempts=0,
                        is_locked=False, locked_time=None)

    # Add users
    user_service.add_user(admin)
    user_service.add_user(clerk)
    user_service.add_user(customer)

    print("Users populated successfully.")
```

```python


def populate_products():
    """
    Creates default products and saves them using ProductService.
    """
    product_service = ProductService()

    # Add sample products
    products = [
        Product(product_id='1', name='Shampoo',
                category='Hair Care',
                price=9.99, stock_quantity=100),
        Product(product_id='2', name='Conditioner',
                category='Hair Care',
                price=8.99, stock_quantity=80),
        Product(product_id='3', name='Hair Gel',
                category='Hair Styling',
                price=5.99, stock_quantity=150),
        Product(product_id='4', name='Hair Spray',
                category='Hair Styling',
                price=6.99, stock_quantity=120),
        Product(product_id='5', name='Hair Serum',
                category='Hair Care', price=15.99,
                stock_quantity=200)
    ]

    for product in products:
        product_service.add_product(product)

    print("Products populated successfully.")


if __name__ == '__main__':
    os.makedirs('data', exist_ok=True)
    populate_users()
    populate_products()
    print("Data population complete.")

```

Appendix 14 unlock_admin.py

unlock_admin.py

```python
from services.user_service import UserService


def unlock_admin_account():
    """
    Unlocks the admin account by resetting
    login attempts and lock status.
    """
    user_service = UserService()
    admin_user = user_service.get_user_by_username('admin')
    if admin_user:
        admin_user.login_attempts = 0
        admin_user.is_locked = False
        user_service.save_users()
        print("Admin account has been unlocked.")
    else:
        print("Admin user not found.")


if __name__ == '__main__':
    unlock_admin_account()

```