

## Appendix 1 main.py

```
main.py 7 X
1  import logging
2
3  from services.user_service import UserService
4  from services.product_service import ProductService
5  from services.order_service import OrderService
6  from controllers.authentication_controller import AuthenticationController
7  from controllers.product_controller import ProductController
8  from controllers.order_controller import OrderController
9  from views.cli_view import CLIView
10
11  """
12  Entry point of the Inventory Management System (IMS).
13  """
14
15  # Configuration of log message.
16  logging.basicConfig(
17      filename='data/system.log',
18      level=logging.INFO,
19      format='%(asctime)s - %(levelname)s - %(message)s'
20  )
21
22
23  def main():
24      """
25      Initialises and starts the IMS application.
26      """
27      # Initialise services
28      user_service = UserService()
29      product_service = ProductService()
30      order_service = OrderService()
31
32      # Initialise controllers
33      auth_controller = AuthenticationController(user_service)
34      product_controller = ProductController(product_service)
35      order_controller = OrderController(order_service, product_service)
36
37      # Initialise views
38      cli_view = CLIView(auth_controller, product_controller, order_controller)
39
40      # Start the application
41      cli_view.main_menu()
42
43
44  if __name__ == '__main__':
45      main()
46
```

## Appendix 2 company\_validation\_api.py

```
1  from flask import Flask, request, jsonify
2
3  app = Flask(__name__)
4
5  # Reference: https://pypi.org/project/Flask
6
7  """
8  Assumed only authorised Companies allow to create customer account.
9  Simulates an API for validating company IDs.
10 """
11
12 valid_company_ids = ['COMPANY123', 'COMPANY456', 'COMPANY789']
13
14
15 @app.route('/validate_company', methods=['GET'])
16 def validate_company():
17     """
18     Validates a company ID against a list of valid IDs.
19     return JSON response indicating whether the company ID is valid.
20     """
21     company_id = request.args.get('company_id')
22     if company_id in valid_company_ids:
23         return jsonify({'valid': True}), 200 # OK Status code
24     else:
25         return jsonify({'valid': False}), 404 # Not Found Status code
26
27
28 if __name__ == '__main__':
29     app.run(host='127.0.0.1', port=5000)
30
```

## Appendix 3 authentication\_controller.py

```
authentication_controller.py  ×
1  import logging
2  import time
3  import re
4  import requests
5  import bcrypt
6
7  from models.user import Clerk, Customer
8
9  """
10 Reference:
11 https://pypi.org/project/requests
12 https://pypi.org/project/bcrypt
13 """
14
15
16 class AuthenticationController:
17     """
18     Controller for handling user authentication and security features,
19     manages user login, registration and account lockout.
20     """
21
22     def __init__(self, user_service):
23         """
24         Initialises the AuthenticationController.
25         """
26         self.user_service = user_service # Provides user data operations.
27         self.MAX_ATTEMPTS = 3 # Default the max allowed failed logins.
28         self.LOCKOUT_TIME = 300 # Default how long (seconds) remains locked.
29
30     def login(self, username, password):
31         """
32         Authenticates a user.
33         Implements account lockout after multiple failed attempts.
34         """
35         current_time = time.time()
36         user = self.user_service.get_user_by_username(username)
37
38         if user:
39             # Account lockout mechanism
40             if user.is_locked:
41                 time_since_locked = current_time - user.locked_time
42                 if time_since_locked < self.LOCKOUT_TIME:
43                     remaining_lock_time = int(
44                         self.LOCKOUT_TIME - time_since_locked
45                     )
46                     minutes, seconds = divmod(remaining_lock_time, 60)
47                     print(f"Account is locked. Please try again "
48                           f"after {minutes} minute(s) "
49                           f"and {seconds} second(s).")
50                     return None
51             else:
52                 # Unlock the account after lockout duration has passed
53                 user.is_locked = False
54                 user.login_attempts = 0
55                 user.locked_time = None
56                 self.user_service.save_users()
```

```

57
58     if user.check_password(password):
59         # Successful login
60         user.login_attempts = 0
61         self.user_service.save_users()
62         logging.info(f"{username} logged in successfully.")
63
64         # Perform MFA (simulation)
65         if self.perform_mfa(user):
66             return user
67         else:
68             print("Multi-Factor Authentication failed.")
69             return None
70     else:
71         # Invalid login attempt
72         user.login_attempts += 1
73         remaining_attempts = self.MAX_ATTEMPTS - user.login_attempts
74         if remaining_attempts > 0:
75             print(f"Invalid username or password. "
76                   f"You have {remaining_attempts} "
77                   f"attempt(s) remaining.")
78         else:
79             # Lock the account
80             user.is_locked = True
81             user.locked_time = current_time
82             logging.warning(f"{username} account locked "
83                             f"due to multiple failed attempts.")
84             print(f"Account locked due to multiple failed attempts. "
85                   f"Please try again "
86                   f"after {int(self.LOCKOUT_TIME / 60)} minutes.")
87             self.user_service.save_users()
88             return None
89     else:
90         print("Invalid username or password.")
91         return None

```



```

92
93     def perform_mfa(self, user):
94         """
95         Simulates Multi-Factor Authentication.
96         """
97         # Generating a code for simulation purposes.
98         mfa_code = '123456'
99         input_code = input(
100             "Enter the MFA code sent to your registered device: "
101         )
102         if input_code == mfa_code:
103             return True
104         else:
105             return False
106
107     def register_customer(self, username, password, company_id):
108         """
109         Registers a new customer after validating inputs and company ID.
110         """
111         if (
112             self.validate_username(username)
113             and self.validate_password(password)
114         ):
115             # Validate company ID via API
116             try:
117                 response = requests.get(
118                     'http://localhost:5000/validate_company',
119                     params={'company_id': company_id}
120                 )
121                 if (
122                     response.status_code == 200
123                     and response.json().get('valid')
124                 ):
125                     if not self.user_service.get_user_by_username(username):
126                         # Hash the password with bcrypt
127                         hashed_password = bcrypt.hashpw(
128                             password.encode('utf-8'),
129                             bcrypt.gensalt()
130                         )
131                         new_customer = Customer(username,
132                                                 hashed_password, company_id)
133                         self.user_service.add_user(new_customer)
134                         logging.info(f"New customer registered: {username}")
135                         print("Registration successful.")
136                     else:
137                         print("Username already exists.")
138                 else:
139                     print("Invalid company ID.")
140             except requests.RequestException as e:
141                 print("Unable to connect to the company validation service. "
142                       "Please try again later.")
143                 logging.error(f"Error connecting to "
144                              f"company validation API: {e}")
145         else:
146             print("Invalid username or password format.")

```

```

147
148 def register_clerk(self, username, password):
149     """
150     Registers a new clerk after validating inputs.
151     """
152     if (
153         self.validate_username(username)
154         and self.validate_password(password)
155     ):
156         if not self.user_service.get_user_by_username(username):
157             # Hash the password with bcrypt
158             hashed_password = bcrypt.hashpw(
159                 password.encode('utf-8'), bcrypt.gensalt()
160             )
161             new_clerk = Clerk(username, hashed_password)
162             self.user_service.add_user(new_clerk)
163             logging.info(f"New clerk registered: {username}")
164             print("Clerk registered successfully.")
165             return True
166         else:
167             print("Username already exists.")
168     else:
169         print("Invalid username or password format.")
170     return False
171
172 def validate_username(self, username):
173     """
174     Validates the username using regex.
175     Rule:
176     Only alphanumeric characters (A-Z, a-z, 0-9)
177     Min. 5 and max. 20 characters in length
178     No space at the beginning or end
179     """
180     pattern = re.compile(r'^[A-Za-z0-9]{5,20}$')
181     return pattern.match(username)
182
183 def validate_password(self, password):
184     """
185     Validates the password for strength.
186     Rule:
187     At least 8 characters long
188     At least one uppercase letter ([A-Z])
189     At least one lowercase letter ([a-z])
190     At least one digit ([0-9])
191     At least one special character from the set [!@#$%^&+=]
192     """
193     if len(password) < 8:
194         return False
195     if not re.search(r'[A-Z]', password):
196         return False
197     if not re.search(r'[a-z]', password):
198         return False
199     if not re.search(r'[0-9]', password):
200         return False
201     if not re.search(r'[!@#$%^&+=]', password):
202         return False
203     return True
204

```

## Appendix 4 order\_controller.py

```
order_controller.py  X
1  from models.order import Order
2
3
4  class OrderController:
5      """
6      Manages order processing logic,
7      handles placing orders and listing customer orders.
8      """
9
10     def __init__(self, order_service, product_service):
11         """
12         Initialises the OrderController.
13         """
14         self.order_service = order_service # Order data operations.
15         self.product_service = product_service # Product data operation.
16
17     def place_order(self, customer_username, items):
18         """
19         Places a new order.
20         """
21         total_price = 0
22         for product_id, quantity in items:
23             product = self.product_service.get_product_by_id(product_id)
24             if not product or product.stock_quantity < quantity:
25                 print(
26                     f"Product '{product_id}' is not available in the "
27                     f"required quantity."
28                 )
29                 return False
30             total_price += product.price * quantity
31
32         order = Order(customer_username, items, total_price)
33         self.order_service.add_order(order)
34
35         # Update stock quantities
36         for product_id, quantity in items:
37             product = self.product_service.get_product_by_id(product_id)
38             product.stock_quantity -= quantity
39         self.product_service.save_products()
40
41         print("Order placed successfully.")
42         return True
43
44     def list_customer_orders(self, customer_username):
45         """
46         Lists all orders placed by a customer.
47         """
48         return self.order_service.list_orders_by_customer(customer_username)
49
```

## Appendix 5 product\_controller.py

```
1  from models.product import Product
2
3
4  class ProductController:
5      """
6      Manages product-related operations,
7      handles CRUD operations for products.
8      """
9
10     def __init__(self, product_service):
11         """
12         Initialises the ProductController.
13         """
14         self.product_service = product_service # Product data operation.
15
16     def add_product(self, product_id, name, category, price, stock_quantity):
17         """
18         Adds a new product to the inventory via product_service
19         """
20         if self.product_service.get_product_by_id(product_id):
21             print(f"Product with ID '{product_id}' already exists.")
22             return False
23         product = Product(product_id, name, category, price, stock_quantity)
24         self.product_service.add_product(product)
25         print(f"Product '{name}' added successfully.")
26         return True
27
28     def update_stock(self, product_id, quantity):
29         """
30         Updates the stock quantity of a product.
31         """
32         return self.product_service.update_stock(product_id, quantity)
33
34     def delete_product(self, product_id):
35         """
36         Deletes a product from the inventory.
37         """
38         if self.product_service.delete_product(product_id):
39             print(f"Product with ID '{product_id}' deleted successfully.")
40             return True
41         else:
42             print(f"No product found with ID '{product_id}'.")
43             return False
44
45     def list_products(self):
46         """
47         Lists all products in the inventory.
48         """
49         return self.product_service.products
50
```



## Appendix 6 order.py

```
order.py  ▸ ×
1  import uuid
2  from datetime import datetime
3
4
5  class Order:
6      """
7      Represents a customer's order.
8      """
9
10     def __init__(self, customer_username, items, total_price):
11         """
12         Initialises an Order object.
13         """
14         self.order_id = str(uuid.uuid4()) # Unique UUID for the order.
15         self.customer_username = customer_username
16         self.items = items # List of product_id, quantity.
17         self.total_price = total_price # The total cost of all items.
18         # Timestamp (YYYY-MM-DD HH:MM:SS) when the order was created.
19         self.order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
20
```

## Appendix 7 product.py

```
product.py  ▸ ×
1  class Product:
2      """
3      Represents a product in the inventory.
4      """
5
6     def __init__(self, product_id, name, category, price, stock_quantity):
7         """
8         Initialises a Product object.
9         """
10         self.product_id = product_id # The unique identifier for product.
11         self.name = name # The product name
12         self.category = category # The category to which the product belongs.
13         self.price = price # The price of the product.
14         self.stock_quantity = stock_quantity # Qty. of product in stock.
15
```

## Appendix 8 user.py

```
user.py  X
1  import bcrypt
2
3
4  class User:
5      """
6      Base class representing a user.
7      """
8
9      def __init__(self, username, password, role=None,
10                  login_attempts=0,
11                  is_locked=False, locked_time=None
12                  ):
13          """
14          Initialises a User object.
15          """
16          self.username = username
17          self.role = role # Role of the user.
18          self.password = password # Store bcrypt-hashed password as bytes
19          self.login_attempts = login_attempts # No. of failed login attempts.
20          self.is_locked = is_locked # Indicates if the user is locked out
21          self.locked_time = locked_time # Timestamp when account was locked
22
23      def check_password(self, plain_password):
24          """
25          Validates the input password against
26          the stored bcrypt-hashed password.
27          Return True if the password matches.
28          """
29          return bcrypt.checkpw(plain_password.encode('utf-8'), self.password)
30
31
32  class Admin(User):
33      """Admin inheriting from User."""
34      def __init__(self, username, password,
35                  login_attempts=0,
36                  is_locked=False, locked_time=None):
37          """
38          Initialises an Admin object.
39          """
40          super().__init__(username, password, role='Admin',
41                          login_attempts=login_attempts,
42                          is_locked=is_locked, locked_time=locked_time
43                          )
44
45  class Clerk(User):
46      """Clerk inheriting from User."""
47      def __init__(self, username, password,
48                  login_attempts=0,
49                  is_locked=False, locked_time=None):
50          """
51          Initialises a Clerk object.
52          """
53          super().__init__(username, password, role='Clerk',
54                          login_attempts=login_attempts,
55                          is_locked=is_locked, locked_time=locked_time
56                          )
57
58
59  class Customer(User):
60      """
61      Customer inheriting from User.
62      """
63      def __init__(self, username, password, company_id,
64                  login_attempts=0, is_locked=False,
65                  locked_time=None):
66          """
67          Initialises a Customer object.
68          """
69          super().__init__(
70              username, password, role='Customer',
71              login_attempts=login_attempts, is_locked=is_locked,
72              locked_time=locked_time
73              )
74          self.company_id = company_id # Company ID associated with customer.
75
76
```

## Appendix 9 order\_service.py

```
order_service.py
1 import json
2 import os
3 from models.order import Order
4
5
6 class OrderService:
7     """
8     Handles order data operations,
9     manages loading, saving, adding and
10    retrieving orders to/from persistent storage.
11    """
12
13    DATA_FILE = 'data/orders.json' # Orders data file.
14
15    def __init__(self):
16        """
17        Initialises the OrderService by
18        loading orders data from storage.
19        """
20        self.orders = self.load_orders()
21
22    def load_orders(self):
23        """
24        Loads orders from the JSON file.
25        """
26        if not os.path.exists(self.DATA_FILE):
27            return []
28
29        with open(self.DATA_FILE, 'r') as file:
30            order_dicts = json.load(file)
31
32        orders = []
33        for order_data in order_dicts:
34            customer_username = order_data['customer_username']
35            items = order_data['items']
36            total_price = order_data['total_price']
37
38            order = Order(customer_username, items, total_price)
39            order.order_id = order_data.get('order_id', order.order_id)
40            order.order_date = order_data.get('order_date', order.order_date)
41
42            orders.append(order)
43        return orders
44
45    def save_orders(self):
46        """
47        Saves orders to the JSON file.
48        """
49        order_dicts = [order.__dict__ for order in self.orders]
50        with open(self.DATA_FILE, 'w') as file:
51            json.dump(order_dicts, file)
52
53    def add_order(self, order):
54        """
55        Adds a new order.
56        """
57        self.orders.append(order)
58        self.save_orders()
59
60    def get_order_by_id(self, order_id):
61        """
62        Retrieves an order by its ID.
63        """
64        for order in self.orders:
65            if order.order_id == order_id:
66                return order
67        return None
68
69    def list_orders_by_customer(self, customer_username):
70        """
71        Retrieves all orders placed by a specific customer.
72        """
73        return [order for order in self
74                .orders if order.customer_username == customer_username]
75
```

## Appendix 10 product\_service.py

```
1  import os
2  import json
3  from models.product import Product
4
5
6  class ProductService:
7      """
8      Manages products, including
9      adding new products, updating existing products,
10     and saving/loading products to/from persistent storage.
11     """
12
13     PRODUCTS_FILE = 'data/products.json' # Product data file.
14
15     def __init__(self):
16         """
17         Initialises the ProductService by
18         loading existing products from storage.
19         """
20         self.products = self.load_products()
21
22     def add_product(self, product):
23         """
24         Adds a new product to the inventory.
25         """
26         if self.get_product_by_id(product.product_id):
27             print(f"Product with ID '{product.product_id}' already exists.")
28             return False
29         self.products.append(product)
30         self.save_products()
31         return True
32
33     def update_stock(self, product_id, quantity):
34         """
35         Updates the stock quantity of a product.
36         """
37         product = self.get_product_by_id(product_id)
38         if product:
39             product.stock_quantity = quantity
40             self.save_products()
41             return True
42         else:
43             print(f"No product found with ID '{product_id}'.")
44             return False
45
46     def get_product_by_id(self, product_id):
47         """
48         Retrieves a product by its ID.
49         """
50         for product in self.products:
51             if product.product_id == product_id:
52                 return product
53         return None
```



```

54
55     def save_products(self):
56         """
57         Saves current products to JSON file for persistent storage.
58         """
59         products_data = []
60         for product in self.products:
61             product_data = {
62                 'product_id': product.product_id,
63                 'name': product.name,
64                 'category': product.category,
65                 'price': product.price,
66                 'stock_quantity': product.stock_quantity
67             }
68             products_data.append(product_data)
69
70         os.makedirs('data', exist_ok=True)
71         with open(self.PRODUCTS_FILE, 'w') as f:
72             json.dump(products_data, f, indent=4)
73
74     def load_products(self):
75         """
76         Loads products from a JSON file and returns a list of Product objects.
77         """
78         try:
79             with open(self.PRODUCTS_FILE, 'r') as f:
80                 products_data = json.load(f)
81
82                 products = []
83                 for product_data in products_data:
84                     product = Product(
85                         product_id=product_data['product_id'],
86                         name=product_data['name'],
87                         category=product_data['category'],
88                         price=product_data['price'],
89                         stock_quantity=product_data['stock_quantity']
90                     )
91                     products.append(product)
92                 return products
93         except FileNotFoundError:
94             return []
95         except json.JSONDecodeError as e:
96             print("An error occurred while loading products:", str(e))
97             return []
98
99     def delete_product(self, product_id):
100         """
101         Deletes a product from the inventory.
102         """
103         product = self.get_product_by_id(product_id)
104         if product:
105             self.products.remove(product)
106             self.save_products()
107             return True
108         else:
109             return False
110

```

## Appendix 11 user\_service.py

```
1  import os
2  import json
3  from models.user import Admin, Clerk, Customer
4
5
6  class UserService:
7      """
8      Manages user accounts, including
9      adding new users, authenticating existing users,
10     and saving/loading users to/from persistent storage.
11     """
12
13     USERS_FILE = 'data/users.json' # Users data file.
14
15     def __init__(self):
16         """
17         Initialises the UserService by
18         loading existing users from storage.
19         """
20         self.users = self.load_users()
21
22     def add_user(self, user):
23         """
24         Adds a new user to the system.
25         """
26         if self.get_user_by_username(user.username):
27             print(f"User with username '{user.username}' already exists.")
28             return False
29
30         self.users.append(user)
31         self.save_users()
32         return True
33
34     def get_user_by_username(self, username):
35         """
36         Retrieves a user object by username.
37         """
38         for user in self.users:
39             if user.username == username:
40                 return user
41         return None
```

```

42
43     def save_users(self):
44         """
45         Saves the current list of users to JSON file for persistent storage.
46         The hashed password is stored directly as a UTF-8 string in JSON.
47         """
48         users_data = []
49         for user in self.users:
50             # Convert bytes hashed password to a UTF-8 string before saving
51             hashed_pw_str = (user.password.decode('utf-8')
52                             if isinstance(user.password, bytes)
53                             else user.password)
54
55             user_data = {
56                 'username': user.username,
57                 'password': hashed_pw_str,
58                 'role': user.role,
59                 'login_attempts': user.login_attempts,
60                 'is_locked': user.is_locked,
61                 'locked_time': user.locked_time
62             }
63
64             if user.role == 'Customer':
65                 user_data['company_id'] = user.company_id
66
67             users_data.append(user_data)
68
69             os.makedirs('data', exist_ok=True)
70             with open(self.USERS_FILE, 'w') as f:
71                 json.dump(users_data, f, indent=4)
72
73     def load_users(self):
74         """
75         Loads users from JSON file and returns a list of User objects.
76         """
77         try:
78             with open(self.USERS_FILE, 'r') as f:
79                 users_data = json.load(f)
80
81             users = []
82             for user_data in users_data:
83                 username = user_data['username']
84                 role = user_data['role']
85                 hashed_password_str = user_data['password']
86                 login_attempts = user_data.get('login_attempts', 0)
87                 is_locked = user_data.get('is_locked', False)
88                 locked_time = user_data.get('locked_time', None)
89
90                 # Convert the hashed password string back to bytes
91                 hashed_password = hashed_password_str.encode('utf-8')
92
93                 if role == 'Admin':
94                     user = Admin(
95                         username,
96                         hashed_password,
97                         login_attempts=login_attempts,
98                         is_locked=is_locked,
99                         locked_time=locked_time
100                     )

```

```

101         elif role == 'Clerk':
102             user = Clerk(
103                 username,
104                 hashed_password,
105                 login_attempts=login_attempts,
106                 is_locked=is_locked,
107                 locked_time=locked_time
108             )
109         elif role == 'Customer':
110             company_id = user_data.get('company_id', '')
111             user = Customer(
112                 username,
113                 hashed_password,
114                 company_id=company_id,
115                 login_attempts=login_attempts,
116                 is_locked=is_locked,
117                 locked_time=locked_time
118             )
119         else:
120             # Unknown role, skip user
121             continue
122
123         users.append(user)
124
125     return users
126 except FileNotFoundError:
127     # No users file exists, return an empty list
128     return []
129 except json.JSONDecodeError as e:
130     print("An error occurred while loading users:", str(e))
131     return []
132
133 def remove_user(self, username):
134     """
135     Removes a user from the system based on the username.
136     """
137     user = self.get_user_by_username(username)
138     if user:
139         self.users.remove(user)
140         self.save_users()
141         print(f"User '{username}' removed successfully.")
142     else:
143         print(f"No user found with username '{username}'.")
144
145 def unlock_user_account(self, username):
146     """
147     Unlocks a user's account by resetting login attempts and lock status.
148     """
149     user = self.get_user_by_username(username)
150     if user:
151         user.is_locked = False
152         user.login_attempts = 0
153         self.save_users()
154         print(f"User '{username}' account has been unlocked.")
155     else:
156         print(f"No user found with username '{username}'.")
157

```



## Appendix 12 cli\_view.py

```
1 class CLIView:
2     """
3     Command-Line Interface view for the application.
4     """
5
6     def __init__(self, auth_controller, product_controller, order_controller):
7         """
8         Initialises the CLIView with controllers.
9         """
10        self.auth_controller = auth_controller
11        self.product_controller = product_controller
12        self.order_controller = order_controller
13
14    def main_menu(self):
15        """
16        Displays the main menu options (Login, Register, Exit)
17        and collects user input in a loop until the user chooses to exit.
18        """
19        while True:
20            print("\n=== Welcome to IMS ===")
21            print("1. Login")
22            print("2. Register as Customer")
23            print("3. Exit")
24            choice = input("Please select an option: ")
25
26            if choice == '1':
27                self.login()
28            elif choice == '2':
29                self.register_customer()
30            elif choice == '3':
31                print("Goodbye!")
32                break
33            else:
34                print("Invalid option. Please try again.")
35
36    def login(self):
37        """
38        Handles user login process.
39        """
40        print("\n--- Login ---")
41        username = input("Username: ")
42        password = input("Password: ")
43        user = self.auth_controller.login(username, password)
44
45        if user:
46            print(f"\nWelcome, {user.username}!")
47            if user.role == 'Admin':
48                self.admin_menu(user)
49            elif user.role == 'Clerk':
50                self.clerk_menu()
51            elif user.role == 'Customer':
52                self.customer_menu(user)
53        else:
54            print("Login failed.")
```

```

55
56     def register_customer(self):
57         """
58         Handles customer registration process.
59         """
60         print("\n--- Customer Registration ---")
61         username = input("Choose a username: ")
62         password = input("Enter a password: ")
63         company_id = input("Enter your company ID: ")
64         if self.auth_controller.register_customer(
65             username, password, company_id
66         ):
67             print("You can now log in with your new account.")
68
69     def admin_menu(self, admin_user):
70         """
71         Displays the admin menu options and handles user input
72         """
73         while True:
74             print("\n--- Admin Menu ---")
75             print("1. Manage Users")
76             print("2. Manage Products")
77             print("3. View System Logs")
78             print("4. Logout")
79             choice = input("Please select an option: ")
80
81             if choice == '1':
82                 self.manage_users()
83             elif choice == '2':
84                 self.manage_products()
85             elif choice == '3':
86                 self.view_system_logs()
87             elif choice == '4':
88                 print("Logging out.")
89                 break
90             else:
91                 print("Invalid option. Please try again.")

```

```

92
93     def clerk_menu(self):
94         """
95         Displays the clerk menu options and handles user input.
96         """
97         while True:
98             print("\n--- Clerk Menu ---")
99             print("1. Manage Products")
100            print("2. Logout")
101            choice = input("Please select an option: ")
102
103            if choice == '1':
104                self.manage_products()
105            elif choice == '2':
106                print("Logging out.")
107                break
108            else:
109                print("Invalid option. Please try again.")
110
111     def customer_menu(self, customer_user):
112         """
113         Displays the customer menu options and handles user input.
114         """
115         while True:
116             print("\n--- Customer Menu ---")
117             print("1. Place Order")
118             print("2. View Order History")
119             print("3. Logout")
120             choice = input("Please select an option: ")
121
122             if choice == '1':
123                 self.place_order(customer_user)
124             elif choice == '2':
125                 self.view_order_history(customer_user)
126             elif choice == '3':
127                 print("Logging out.")
128                 break
129             else:
130                 print("Invalid option. Please try again.")
131
132     # Define methods for managing users, products, placing orders, etc.
133     """
134     The system focuses on security measures;
135     not all methods have been developed due to resource limitations,
136     e.g. password reset flow,
137     wrap stock input in a try/except to handle invalid integer
138     """

```

```

139
140     def manage_users(self):
141         """
142         Handles user management options.
143         """
144         while True:
145             print("\n--- Manage Users ---")
146             print("1. View All Users")
147             print("2. Add Clerk")
148             print("3. Remove User")
149             print("4. Unlock User Account")
150             print("5. Back to Admin Menu")
151             choice = input("Please select an option: ")
152
153             if choice == '1':
154                 self.view_all_users()
155             elif choice == '2':
156                 self.add_clerk()
157             elif choice == '3':
158                 self.remove_user()
159             elif choice == '4':
160                 self.unlock_user_account()
161             elif choice == '5':
162                 break
163             else:
164                 print("Invalid option. Please try again.")
165
166     def view_all_users(self):
167         """
168         Displays all users in the system.
169         """
170         print("\n--- All Users ---")
171         users = self.auth_controller.user_service.users
172         for user in users:
173             print(f"Username: {user.username}, "
174                   f"Role: {user.role}, "
175                   f"Locked: {user.is_locked}"
176                   )
177
178     def add_clerk(self):
179         """
180         Handles the process of adding a new clerk
181         """
182         print("\n--- Add Clerk ---")
183         username = input("Enter clerk username: ")
184         password = input("Enter clerk password: ")
185         if self.auth_controller.register_clerk(username, password):
186             print("Clerk added successfully.")

```



```

187
188     def remove_user(self):
189         """
190         Handles the removal of a user.
191         """
192         print("\n--- Remove User ---")
193         username = input("Enter username to remove: ")
194         self.auth_controller.user_service.remove_user(username)
195
196     def unlock_user_account(self):
197         """
198         Handles unlocking a user account.
199         """
200         print("\n--- Unlock User Account ---")
201         username = input("Enter username to unlock: ")
202         self.auth_controller.user_service.unlock_user_account(username)
203
204     def view_system_logs(self):
205         """
206         Displays system logs.
207         """
208         print("\n--- System Logs ---")
209         try:
210             with open('data/system.log', 'r') as log_file:
211                 logs = log_file.read()
212                 print(logs)
213         except FileNotFoundError:
214             print("No system log file found.")
215
216     def manage_products(self):
217         """
218         Handles product management options.
219         """
220         while True:
221             print("\n--- Manage Products ---")
222             print("1. View All Products")
223             print("2. Add Product")
224             print("3. Update Product Stock")
225             print("4. Delete Product")
226             print("5. Back to Previous Menu")
227             choice = input("Please select an option: ")
228
229             if choice == '1':
230                 self.view_all_products()
231             elif choice == '2':
232                 self.add_product()
233             elif choice == '3':
234                 self.update_product_stock()
235             elif choice == '4':
236                 self.delete_product()
237             elif choice == '5':
238                 break
239             else:
240                 print("Invalid option. Please try again.")

```

```

241
242     def view_all_products(self):
243         """
244         Displays all products in the inventory.
245         """
246         print("\n--- All Products ---")
247         products = self.product_controller.list_products()
248         for product in products:
249             print(
250                 f"ID: {product.product_id}, Name: {product.name}, "
251                 f"Category: {product.category}, Price: {product.price}, "
252                 f"Stock: {product.stock_quantity}"
253             )
254
255     def add_product(self):
256         """
257         Handles the process of adding a new product.
258         """
259         print("\n--- Add Product ---")
260         product_id = input("Enter product ID: ")
261         name = input("Enter product name: ")
262         category = input("Enter product category: ")
263         price = float(input("Enter product price (numeric only): "))
264         stock_quantity = int(input("Enter stock quantity (integer only): "))
265         self.product_controller.add_product(
266             product_id, name, category, price, stock_quantity
267         )
268
269     def update_product_stock(self):
270         """
271         Handles updating the stock quantity of a product.
272         """
273         print("\n--- Update Product Stock ---")
274         product_id = input("Enter product ID: ")
275         quantity = int(input("Enter new stock quantity: "))
276         self.product_controller.update_stock(product_id, quantity)
277
278     def delete_product(self):
279         """
280         Handles the deletion of a product.
281         """
282         print("\n--- Delete Product ---")
283         product_id = input("Enter product ID to delete: ")
284         self.product_controller.delete_product(product_id)

```

```

285
286     def place_order(self, customer_user):
287         """
288         Handles the process of placing an order.
289         """
290         print("\n--- Place Order ---")
291         products = self.product_controller.list_products()
292         print("Available Products:")
293         for product in products:
294             print(f"ID: {product.product_id}, "
295                   f"Name: {product.name}, "
296                   f"Price: {product.price}, "
297                   f"Stock: {product.stock_quantity}"
298                 )
299
300         items = []
301         while True:
302             product_id = input(
303                 "Enter product ID to add to cart (or 'done' to finish): "
304             )
305             if product_id.lower() == 'done':
306                 print("\nThe statement will be sent to your Company")
307                 break
308             quantity = int(input("Enter quantity: "))
309             items.append((product_id, quantity))
310
311         if items:
312             self.order_controller.place_order(customer_user.username, items)
313         else:
314             print("No items selected.")
315
316     def view_order_history(self, customer_user):
317         """
318         Displays the order history for a customer.
319         """
320         print("\n--- Order History ---")
321         orders = self.order_controller.list_customer_orders(
322             customer_user.username
323         )
324         if orders:
325             for order in orders:
326                 print(f"Order ID: {order.order_id}, "
327                       f"Date: {order.order_date}, "
328                       f"Total Price: {order.total_price}"
329                     )
330                 for item in order.items:
331                     product_id, quantity = item
332                     product = (
333                         self.product_controller.product_service
334                         .get_product_by_id(product_id)
335                     )
336                     if product:
337                         print(
338                             f"Product: {product.name}, "
339                             f"Quantity: {quantity}, "
340                             f"Price: {product.price}"
341                         )
342             else:
343                 print("You have no previous orders.")
344

```

## Appendix 13 populate\_data.py

```
1  """
2  Utility script that creates initial users (Admin, Clerk, Customer)
3  and a sample set of products.
4  Writes data to JSON files in the "data" directory.
5  """
6
7  # Reference: https://pypi.org/project/bcrypt
8
9  import os
10 import bcrypt
11
12 from services.user_service import UserService
13 from services.product_service import ProductService
14 from models.user import Admin, Clerk, Customer
15 from models.product import Product
16
17
18 def populate_users():
19     """
20     Creates default users and saves them using UserService.
21     """
22     user_service = UserService()
23
24     # Define passwords that meet the security policy
25     admin_password_plain = 'Admin@1234'
26     clerk_password_plain = 'Clerk@1234'
27     customer_password_plain = 'Customer@1234'
28
29     # Hash passwords with bcrypt
30     admin_password_hashed = bcrypt.hashpw(
31         admin_password_plain.encode('utf-8'),
32         bcrypt.gensalt()
33     )
34     clerk_password_hashed = bcrypt.hashpw(
35         clerk_password_plain.encode('utf-8'),
36         bcrypt.gensalt()
37     )
38     customer_password_hashed = bcrypt.hashpw(
39         customer_password_plain.encode('utf-8'),
40         bcrypt.gensalt()
41     )
42
43     # Create users
44     admin = Admin(username='admin',
45                   password=admin_password_hashed,
46                   login_attempts=0,
47                   is_locked=False, locked_time=None)
48     clerk = Clerk(username='clerk',
49                   password=clerk_password_hashed,
50                   login_attempts=0,
51                   is_locked=False, locked_time=None)
52     customer = Customer(username='customer',
53                         password=customer_password_hashed,
54                         company_id='COMPANY123',
55                         login_attempts=0,
56                         is_locked=False, locked_time=None)
57
58     # Add users
59     user_service.add_user(admin)
60     user_service.add_user(clerk)
61     user_service.add_user(customer)
62
63     print("Users populated successfully.")
```

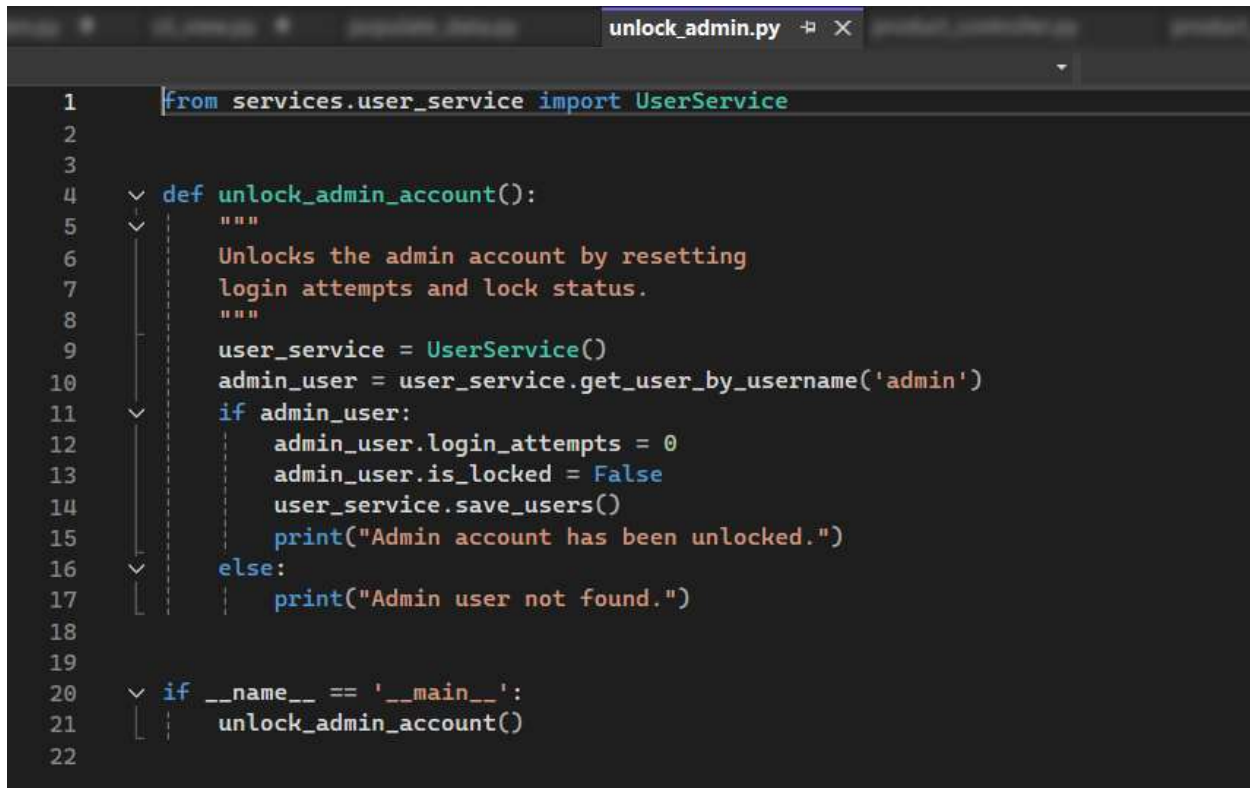


```

64
65
66     def populate_products():
67         """
68         Creates default products and saves them using ProductService.
69         """
70         product_service = ProductService()
71
72         # Add sample products
73         products = [
74             Product(product_id='1', name='Shampoo',
75                     category='Hair Care',
76                     price=9.99, stock_quantity=100),
77             Product(product_id='2', name='Conditioner',
78                     category='Hair Care',
79                     price=8.99, stock_quantity=80),
80             Product(product_id='3', name='Hair Gel',
81                     category='Hair Styling',
82                     price=5.99, stock_quantity=150),
83             Product(product_id='4', name='Hair Spray',
84                     category='Hair Styling',
85                     price=6.99, stock_quantity=120),
86             Product(product_id='5', name='Hair Serum',
87                     category='Hair Care', price=15.99,
88                     stock_quantity=200)
89         ]
90
91         for product in products:
92             product_service.add_product(product)
93
94         print("Products populated successfully.")
95
96
97     if __name__ == '__main__':
98         os.makedirs('data', exist_ok=True)
99         populate_users()
100         populate_products()
101         print("Data population complete.")
102

```

## Appendix 14 unlock\_admin.py



```
1  from services.user_service import UserService
2
3
4  def unlock_admin_account():
5      """
6      Unlocks the admin account by resetting
7      login attempts and lock status.
8      """
9      user_service = UserService()
10     admin_user = user_service.get_user_by_username('admin')
11     if admin_user:
12         admin_user.login_attempts = 0
13         admin_user.is_locked = False
14         user_service.save_users()
15         print("Admin account has been unlocked.")
16     else:
17         print("Admin user not found.")
18
19
20 if __name__ == '__main__':
21     unlock_admin_account()
22
```