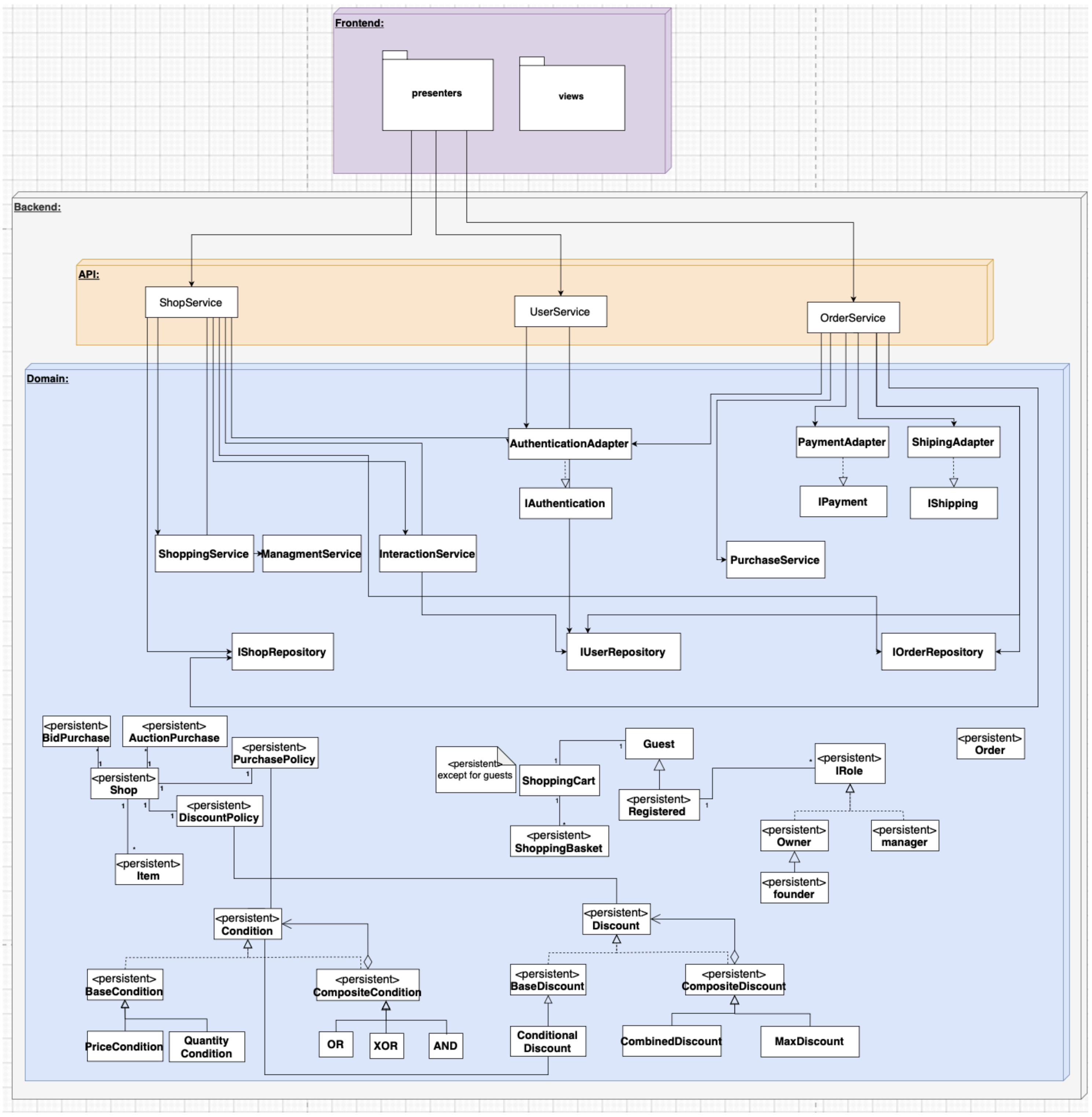
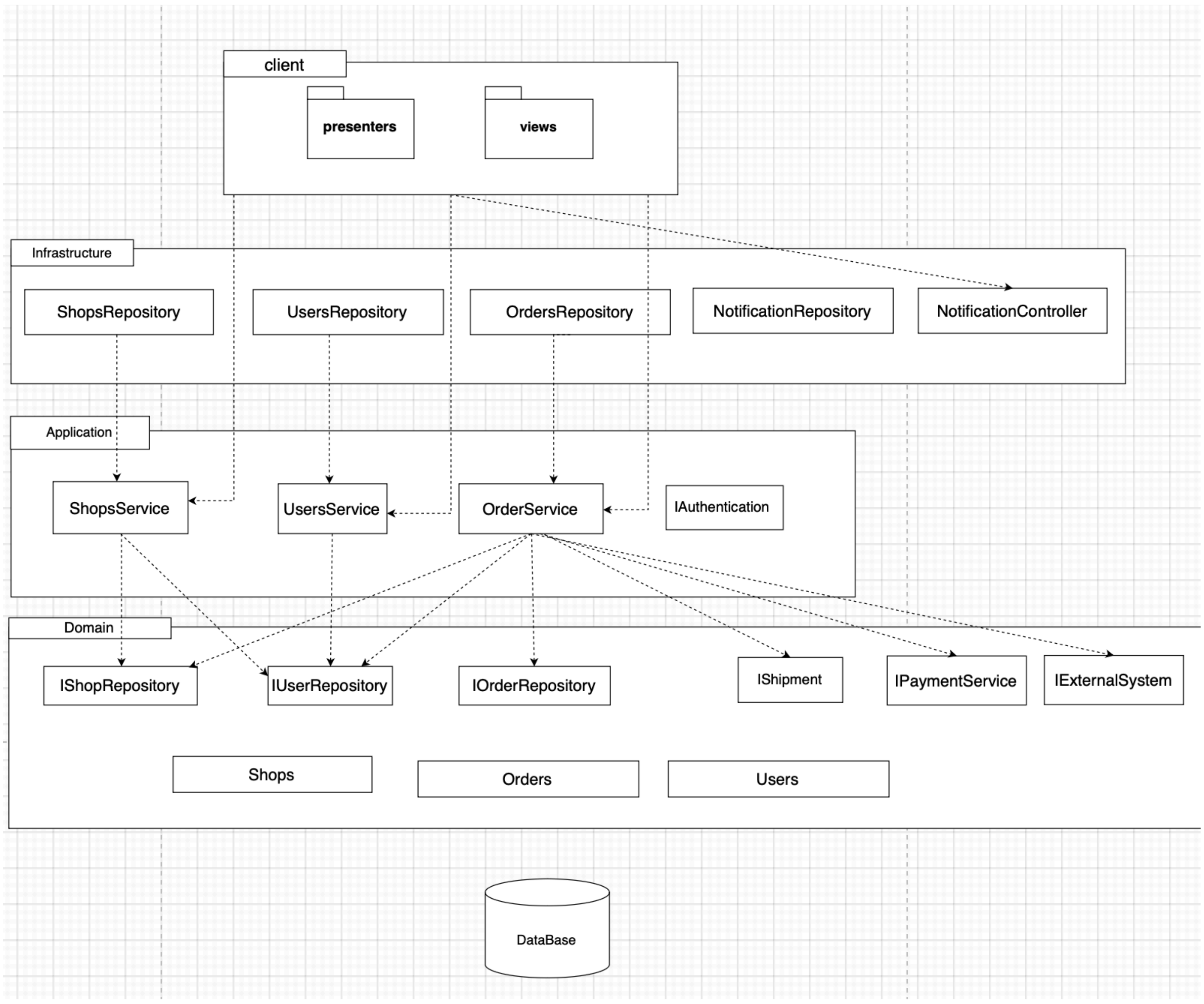
מידול מערכת:

תוכן עניינים

מודל מחלקות לבן:



דיאגרמת ארכיטקטורה:



תרחישי שימוש+ מבחני קבלה:

מערכת:

1.

Use-case: **System initialization**

1. **Actor**: system manager
2. **Trigger:** the system manager initiates the system startup process.
3. **Precondition**:
    1. There is a user with the system manager role.
    2. Configuration files and an initialization state file are available and valid
4. **Parameters:**

    1. A configuration file containing database login information, external services, system administrator information, etc.

    2. A state initialization file containing a list of use cases and arguments for running them
5. **Main Scenario:**
    1. **User:** runs the trading system.
    2. **User:** logs in to the system
    3. **System:** verify user successfully logged in.
    4. **User**: initializes the system
    5. **system**: loads the configuration file
    6. **System:** creates connections to the external services and databases as defined in the file
    7. **system:** loads the initialization state file
    8. **system:** runs the use cases and arguments defined in the file
    9. **system:** enters the initial state as defined
6. **Alternative Flows:**
    1. the user failed to login.

        >> System notifies the user that he failed to log in.
    2. failed to read the configuration file or create connections to external services:
        >> Appropriate message to the user
    3. Failed to run a init file:

        >>The system reports a failure to read or run the usage stories from the file

Test Name	Setup and Parameters	Expected Result
System initialization - Success	1. User u1 is created. 2. User u1 logs in.	Trading system is on.

	3. User u1 is assigned as a system manager. 4. User u1 initializes the trading system.	
System initialization - User is not system manager	1. User u1 is created. 2. User u1 logs in. 3. User u1 initializes the trading system.	Trading system remains off.

2.

Use-case: **payment service**

1. **Actor**: user
2. **Trigger:** the user proceeds an order.
3. **Precondition**:

   connection to the payment service is established.
4. **Parameters:** orderDetails,userPaymentDetailes,shopDetails.
5. **Main Scenario:**
   1. **User:** proceeds order
   2. **System:** sends details to payment service.
   3. **PaymentService:** sends confirmation message if charge was successful.
6. **Alternative Flows:**
   1. The charge failed.

      >> System notifies the user that the order cannot proceed.

3.

Use-case: **Supply service**

1. **Actor**: user
2. **Trigger:** the user proceeds an order.
3. **Precondition**:

   connection to the supply service is established.
4. **Parameters:** orderDetails,userDeliveryDetailes.
5. **Main Scenario:**
   1. **System:** sends details to Supply service.
   2. **SupplyService:** sends confirmation message if supply request was successful.
6. **Alternative Flows:**

   The delivery request failed.

   >> System notifies the user the order cannot be delivered.

Test Name	Setup and Parameters	Expected Result
successfulBuyCartContentTest	1. Prepare valid payment and shipment details and configure external services to return success. 2. Shop owner registers, creates a shop, and lists an item for sale. 3. Buyer registers, logs in, and adds one unit of the item to their cart. 4. Buyer places the order using the valid payment and shipment details.	- An **Order** is created with valid paymentId and shipmentId. - The item's stock in the shop decreases by 1. - Buyer's cart is emptied. - The new order appears in the buyer's personal order history. - External payment and shipment services were invoked.
BuyCartContentTest_paymentFails/ shipmentFails	1. Prepare payment/shipment details that will be rejected 2. Shop owner registers, creates a shop, and lists an item. 3. Buyer registers, logs in, and adds one unit of the item to their cart. 4. Buyer attempts to place the order with the invalid payment details (shipment details remain valid).	- The order request is denied (purchase fails). - Buyer is informed that the payment details are invalid. - The cart still contains the item. - Shop stock remains unchanged.

4. real time notification

4.1 Use-case: **Purchase Notification**

1. **Actor:** user
2. **Trigger:** A customer completes a purchase from the store.

3. **Precondition:**
   1.The store is active.
   2.The user is store owner
   3.User is logged in
4. **Parameters:** notification Info
5. **Main Scenario:**
   System: Detects the completed purchase.
   System: Sends a real-time alert to the store owners.
6. **Alternative Flows:**
   1. user is not logged in.
      >> **System:** Queues the notification and sends it when the user reconnects.

Test Name	Setup and Parameters	Expected Result
Real-time purchase notification- Success	1.User u1 is registered. 2.User u1 create shop 3.User u1 add item1 to shop. 4.User u2 is registered. 5.User u2 successfully bought a cart with item1.	User u1 receives the real time notification.
delayed purchase notification- failed	1.User u1 is registered. 2.User u1 create shop 3.User u1 add item1 to shop. 4.User u1 is logged out. 4.User u2 is registered. 5.User u2 successfully bought a cart with item1. 6.User u1 is logged in.	Notification is saved in u1's notification queue.

4.2 Use-case: **Store Closing Notification**

1. **Actor:** user
2. **Trigger:** The store is closed by system manager/ founder
3. **Precondition**
   1. The user has an active role in the store.
   2.user is logged in
4. **Parameters:** notification Info
5. **Main Scenario:**
   System: Detects the store has been closed.
   System: Sends a real-time alert to the store owners and managers

6. **Alternative Flows:**
   1. user is not logged in.
      >> **System:** Queues the notification and sends it when the user reconnects.

Test Name	Setup and Parameters	Expected Result
Real-time close shop notification- Success	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u1 add appointment to u2 in shop1. 5.User u1 close shop1.	User u2 receives the real time notification.
delayed close shop notification- failed	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u1 add appointment to u2 in shop1. 4.User u2 logged out. 5.User u1 close shop1.	Notification is saved in u2's notification queue..

4.3 Use-case: **Store Reopening Notification**

1. **Actor:** user
2. **Trigger:** The store is reopened.
3. **Precondition:**
   1. The store was previously closed.
   2. User is logged in.
4. **Parameters:** notification Info.
5. **Main Scenario:**
   System: Detects that the store is now open.
   System: Sends a real-time notification to the store owners and managers.
6. **Alternative Flows:**
   1. user is not logged in.
   >> **System:** Queues the notification and sends it when the user reconnects.

4.4 Use-case: **Appointment Removed Notification**

1. **Actor:** user
2. **Trigger:** The store owner's role is revoked.

3. **Precondition:**
   1.The store owner had a role in the shop
   2.User is logged in
4. **Parameters:** notification Info
5. **Main Scenario:**
   System: Detects the removal of user's appointment.
   System: Sends a real-time alert to the.
6. **Alternative Flows:**
   1. user is not logged in.
   >> **System:** Queues the notification and sends it when the user reconnects.

Test Name	Setup and Parameters	Expected Result
Real-time remove appointment notification- Success	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u1 add appointment to u2 in shop1. 5.User u1 remove u2's appointment in shop1.	User u2 receives the real time notification.
remove appointment notification- failed	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u1 add appointment to u2 in shop1. 5.User u2 is logged out 5.User u1 remove u2's appointment in shop1.	Notification is saved in u2's notification queue.

4.5 Use-case: **Message/Inquiry Notification**

1. **Actor:** Subscriber (e.g., store owner or manager with permission)
2. **Trigger:** Another user sends a message or inquiry to the shop.
3. **Precondition:**
   1. The user is registered in the system.
   2. The subscriber is the store owner or manager with permission.
   3. Subscriber is logged in
   4. User sent a message.
4. **Parameters:** notification Info

5. **Main Scenario:**
   System: Receives a new message or inquiry directed to the subscriber's shop.
   System: Sends a real-time notification to the shop owners and managers
6. **Alternative Flows:**
   1. The subscriber is offline or cannot be reached.
   >> **System:** Queues the notification and delivers it when the subscriber reconnects.

Test Name	Setup and Parameters	Expected Result
Real-time get message notification- Success	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u2 send a message to shop1.	User u1 receives the real time notification.
Real-time get message notification- failed	1.User u1 is registered. 2.User u2 is registered. 3.User u1 create shop1 4.User u1 logged out. 4.User u2 send a message to shop1.	Notification is saved in u2's notification queue.

5.
<u>Use-case: **delayed notification**</u>

1. **Actor**: user
2. **Trigger:** A real-time event occurs while the subscriber is not online.
3. **Precondition**:
   1. user is registered
   2. the user isn't logged in.
4. **Parameters:**
5. **Main Scenario:**
   1. **System:** Detects a relevant event (e.g., message, purchase, store update).
   2. **System:** Checks if the subscriber is currently online/active in the market.

3. **System:** If not, store the notification in a pending state.

4. **System:** Upon the subscriber's next login or visit to the market, displays the stored notifications.

6. **Alternative Flows:**

   1. user is logged in.

   >> System will send a real-time notification.

Test Name	Setup and Parameters	Expected Result
delayed notification-success	1.user u1 register 2.user u1 logged out 3.user u1 get notification 4.user u1 logged in	u2 receives all the notifications that are in his notification queue.

משתמשים:

1. פעולות מבקר-אורח:

1.1

Use case: **Login as Guest**

1. **Actors**: User
2. **Trigger**: User entered the system as guest
3. **Precondition**:
   1. The user is not currently in the system
4. **Parameters**: None
5. **Main Scenario:**
   1. **System:** Generates a new session token
   2. **User:** Enters the system with the session token

3. **System:** Assigns a temporary ID to the guest

4. **System:** Assigns an empty shopping cart to the guest using the same ID

6. **Alternative flows**: None

Test Name	**Setup & Parameters**	**Expected Results**
testEnterSystemAsGuest_ShouldCreateTokenAndEmptyCart	Guest enters system	- Returns a new guest token - An empty cart is created

1.2

Use case: **exit as Guest**

1. **Actors**: User

2. **Trigger**: User exit the system as guest

3. **Precondition**:

    1. The user is currently in the system as a guest

4. **Parameters**: GuestSessionToken

5. **Main Scenario**:

    1. **User:** Authenticates the session token via the authentication service

    2. **User:** Chooses to exit

    3. **System:** Deletes the user's shopping cart

    4. **System:** Disassociates the session token from the guest (deletes session)

6. **Alternative flows**: Non

Test Name	Setup & Parameters	Expected Results
SuccessfulGuestExit	1. User login into the system as guest 2. Guest chose to exit the system	1. Guest isn't logged into the system 2. Guests shopping cart is deleted 3. Guest's TempID1 is deleted

1.3

Use case: **Register**

1. **Actors**: User

2. **Trigger:** User registers in the system

3. **Precondition**:

    1. The user is currently in the system as a guest

4. **Parameters**: sessionToken, Username, Password, dateOfBirth

5. **Main Scenario**:

    1. **User:** Chooses “Register”

    2. **System:** Loads the registration page

    3. **User:** Enters personal information

    4. **System:** Registers a new user in the system

    5. **System:** Transfers the guest’s cart to the new registered user

6. **Alternative flows:**

1. user enter an existing userName

    >> **System:** Displays an error message

Test Name	Setup & Parameters	Expected Results
testRegisterNewUser_WithValidDetails_ShouldSucceed	1. User logs in as guest 2. Guest chose register 3. Guest insert information correctly	1. New user account is created and logged in 2. Guest cart is transferred
testRegisterDuplicateUsername_ShouldFail	1. User logs in as guest 2. Guest chose register 3. Guest insert an existing username	1. Registration is rejected with an error 2. No new account is created

1.4

Use case: **Login as Registered**

1. **Actors**: User

2. **Trigger**: User entered the system as Registered

3. **Precondition**:
    1. The user was not in the system as user
    2. The user was registered

4. **Parameters**: GuestSessionToken, Username, Password

5. **Main Scenario**:

    1. User: Authenticate The Guest’s SessionToken using the authentication service

2. System: Loads login page

3. User: Enters username and password

4. System: Checks if the username matches the password

5. System: Confirms they match

6. System: Generates new session-token

7. System: Associates new session-token with the new logged-in user

8. System: Registers the active user as the user linked to the username

9. System: Loads main page

6. **Alternative flows**:

    1. **username and password don't match.**

        >> System: Sends an error message

Test Name	Setup & Parameters	Expected Results
testLoginUser_WithValidCredentials_ShouldSucceed	1. Guest logs in 2. Registers “user123” 3. Logs in with correct username & password	1. Login succeeds 2. Valid token returned
testLoginUser_WithInvalidUsername_ShouldFail	1. Guest logs in 2. Registers “user123” 3. Attempts login with wrong username	1. Login fails 2. No token returned
testLoginUser_WithInvalidPassword_ShouldFail	1. Guest logs in 2. Registers “user123” 3. Attempts login with wrong password	1. Login fails 2. No token returned

2. פעולות קנייה של מבקר-אורח:

2.1:

Use case: **Getting information on various shops and items**

1. **Actors**: Guest
2. **Trigger**: User entered the system
3. **Precondition**: The user was not in the system
4. **Parameters**: None
5. **Main Scenario**:
    1. **System**: get all the shops
    2. **System**: get all the shop's items
    3. **System**: send the data to the user
6. **Alternative flows**: None

Test Name	Setup & Parameters	Expected Results
GetShopsAndItems	1. User is not in the system 2. User enters the system	1. System retrieves all shops 2. System retrieves all items 3. System sends the data to the user 4. User receives all the data

2.2

a.

Use case: **Items search for non-specific store**

1. **Actors**: Guest
2. **Trigger**: User request items search
3. **Precondition**:
4. **Parameters**: ItemName, Category, keyWord, PriceRange, ItemRate, ShopRate
5. **Main Scenario**:
    1. **System**: search in all the shops
    2. **System**: filter all the items using the provided parameters
    3. **System**: send the data to the user

6. **Alternative flows**: None

Test Name	Setup & Parameters	Expected Results
searchItemsWithFilters	1. User enters the system 2. User requests item search with parameters: ItemName, Category, KeyText, PriceRange, ItemRate, ShopRate	1. System searches in all shops 2. System filters items using provided parameters 3. System sends filtered data to the user
searchItemsWithoutFilters	1. User is not in the system 2. User enters the system 3. User requests item search without parameters	1. System searches in all shops 2. System retrieves all available items 3. System sends all available items to the user
emptySearchResults	1. User is not in the system 2. User enters the system 3. User requests item search with parameters that do not match any items	1. System searches in all shops 2. No matching items found 3. System returns "No items found" message

b.

<u>Use case**: Items search for specific store**</u>

1. **Actors**: Guest

2. **Trigger**: User request items search

3. **Precondition**:

    1. Shop exists

4. **Parameters**: ShopID, ItemName, Category, KeyText, PriceRange, ItemRate

5. **Main Scenario**:

1. System: search ShopID

2. System: validates that the shop exists

3. System: filter all the items in this shop, using the provided parameters

4. System: send the data to the user

6. **Alternative flows:**

Shop was not found:

>>System notifies to the user that the shop was not found

Test Name	Setup & Parameters	Expected Results
searchItemsInSpecificShop	1. User enters the system 2. User requests item search with ShopID, ItemName, Category, KeyText, PriceRange, ItemRate	1. System searches for ShopID 2. System validates that the shop exists 3. System filters items in this shop using provided parameters 4. System sends filtered data to the user
ShopNotFound	1. User enters the system 2. User requests item search with a non-existent ShopID	1. System searches for ShopID 2. System returns "Shop not found" message

2.3

<u>Use case**: Item adding to shop basket**</u>

1. **Actors**: Guest

2. **Trigger**: User request items saving

3. **Precondition**:

1. Shop exists

2. Items exists

3. Items available

4. **Parameters**: SessionToken, ShopID, ItemsID (Collection)

5. **Main Scenario:**

1. **User:** choose an item
2. **System**: validates that the items exist

3. **System**: validates that the items are available
4. **System**: save the items in the user’s basket
5. **System**: send confirmation to user

6. **Alternative flows**:

Items were not available:

>>System notifies to the user that one or more items were not available

Test Name	Setup & Parameters	Expected Results
addItemToBasketTest	1. User enters the system 2. User requests to add items to the cart.	1. System adds items to the respective basket
AddItemsToCart- item not available	1. User enters the system 2. User requests to add items to the cart.	1. system check for availability and find none 2. System didn't add items to the cart

2.4 a.

Use case: **Check cart content**

1. **Actors**: Guest

2. **Trigger**: User requests the cart’s content

3. **Precondition**:

4. **Parameters**: SessionToken, CartID

5. **Main Scenario**:

1. System: authenticate the user using the authentication service

2. System: search CartID

3. System: send the cart data to the user

6. **Alternative flows**: None

Test Name	Setup & Parameters	Expected Results
checkCartContentTest	1. User enters the system 2. User requests cart content	1. System searches for CartID 2. System validates that the cart exists 3. System verifies that the cart belongs to the user 4. System sends cart data to the user

b.

Use case: **Remove cart content**

1. **Actors**: Guest

2. **Trigger**: User requests to delete some of the cart's content

3. **Precondition**:

    1. The selected items are in the user's cart

4. **Parameters**: SessionToken, CartID, ItemsToDeleteID (Collection)

5. **Main Scenario:**

    1. System: authenticate the user using the authentication service

    2. System: search CartID

    3. System: validates that the cart exists

    4. System: verifies that the UserID of the session matches the cart owner

    5. System: search the selected items with ItemsToDeleteID

    6. System: validates that all the items exist in the cart

    7. System: asks for user confirmation

    8. User: approves deletion

    9. System: delete the items from the user's baskets

    10. System: send confirmation to user

6. **Alternative flows:** None

Test Name	**Setup & Parameters**	**Expected Results**
changeCartContentTest	1. Owner creates a shop with items A, B, C 2. Buyer enters the system (guest) 3. Buyer adds item A and item B to their cart 4. Buyer requests removal of item A (by ID)	Cart now contains exactly one item-B
removeItemFromCart_InvalidShopOrItem_ShouldFail	1. User enters the system 2. Attempts to remove an item using a non-existent shop ID 3. Attempts to remove a non-existent item in a valid shop	1. Both removal attempts are rejected 2. Cart remains empty

removeItemFromCart_BeforeAddingAny_ShouldFail	1. User enters the system 2. Without adding any items, attempts to remove an item	1. Removal is rejected 2. Cart remains empty

2.5

Use case: **Buy cart content**

1. **Actors**: Guest

2. **Trigger**: User requests to buy the cart's content

3. **Precondition**:

    1. There is at least one item in the cart

    2. All the items in the user's cart are available for purchase

4. **Parameters**: SessionToken, CartID

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: search CartID

    3. System: validates that the cart exists

    4. System: verifies that the UserID of the session matches the cart owner

    5. System: validates that all the items in the cart available

    6. System: for each item, calculate the item's price according to the shop DiscountPolicy

    7. System: Calculate the final price for all the items

    8. System: asks for user confirmation

    9. User: approves confirmation

    10. System: verifies that the user's able to buy the items, according to each shop PurchasePolicy

    11. System: charge the user using the payment service

    12. System: using the shipping service

    13. System: send confirmation to user

6. **Alternative flows:**

Items were not available:

    >>System notifies to the user that one or more items were not available

One or more of the PurchasePolicy does not approve purchase:

>>System acknowledges

Charging failed due to payment service error

>>System acknowledges

Shipping failed due to shipping service error

>>System acknowledges

Test Name	Setup & Parameters	Expected Results
successfulBuyCartContentTest	1. Buyer logs in and adds one unit of an item to their cart 2. User requests to buy cart content 3. User enter payment details 4. User enter shipment details.	1. System searches for CartID 2. System validates cart existence 3. System verifies cart ownership 4. System validates item availability 5. System calculates final price with discounts 6. System validates payment and shipment details. 7. System charges the user 8. System sends confirmation to user
ItemUnavailableForPurchase	1. User enters the system 2. User requests to buy cart content but an item is unavailable	1. System validates cart existence 2. System verifies cart ownership 3. System checks item availability 4. System returns "One or more items are unavailable" message
BuyCartContentTest_paymentFails/ shipmentFails	1. User enters the system 2. User requests to buy cart content 3. User enter payment details 4. User enter shipment details.	1. System validates cart existence 2. System verifies cart ownership 3. System checks item availability 4. System validates payment and shipment details. 5. System returns "One or more details are wrong" message

Differences between a guest and a registered user for requirements 1.2 & 2.1-2.5

The changes are:

1.2

1. **Actors**: Registered

4. **Parameters**: SessionToken, CartID

5**. Main Scenario**:

We're adding this:

> **System**: saves the cart data that belongs to CartID and UserID assigned to SessionToken
>
> **System**: send confirmation to user

6. **Alternative flows:**

We're adding this:

Cart was not saved:

> >>System notifies to the user that the cart was not saved

2.3

1. **Actors**: Registered

5. **Main Scenario**:

We're adding this:

> **System**: saves the basket data that belongs to ShopID, ItemsID, UserID (that assigned to SessionToken)
>
> **System**: send confirmation to user

6. **Alternative flows:**

We're adding this:

Basket was not saved:

> >>System notifies to the user that the basket was not saved

2.4

b.

1. **Actors**: Registered

5. **Main Scenario:**

We're adding this:

> **System**: saves the changes were made that belongs to ShopID, UserID (that assigned to SessionToken)
>
> **System**: send confirmation to user

6. **Alternative flows:**

We're adding this:

Basket was not saved:

>>System notifies to the user that the changes were not saved

2.5

1. **Actors**: Registered

4. **Parameters**: SessionToken, CartID

5. **Main Scenario**:

We're adding this:

System: saves the changes were made, as history purchase, that belongs to CartID, UserID (that assigned to SessionToken)

System: send confirmation to user

6. **Alternative flows:**

We're adding this:

Changes were not saved:

>>System notifies to the user that the changes were not saved

3. פעולות קנייה של מבקר-מנוי בשוק:

3.1

Use case: **Logout**

1. **Actors**: Registered

2. **Trigger**: User request to logout

3. **Precondition**:

    1. User is logged in

4. **Parameters**: SessionToken

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged.

    3. System: saves the cart data that belongs to UserID assigned to SessionToken.

    4. System: changes the user status to Guest

6. **Alternative flows:** None

Test Name	Setup & Parameters	Expected Results

testLogout_RegisteredUser_ShouldReturnToGuestState	1. User 123 is created 2. User 123 logs in 3. User 123 requests logout	1. System verifies user login 2. System changes user status to Guest

3.2

<u>Use case: **Open shop**</u>

2. **Actors**: Registered

3. **Trigger**: User requests to open a shop

4. **Precondition:**

    1. User is logged in

    2. User has permission to open a shop (based on system rules)

Parameters: SessionToken, ShopName, ShopDetails

5. **Main Scenario:**

    System: authenticate the user using the authentication service

    System: verifies that the user with UserID of the session is logged in

    System: verifies that the user has permission to open a shop

    System: verifies that the shop name is unique

    System: creates a new shop with the given details

    System: assigns the user as the founder of the new shop

6. **Alternative flows**:

User not logged:

    >>System notifies the user that they are not logged in

Test Name	**Setup & Parameters**	**Expected Results**
openShopTest	1. User enters the system 2. User registers with valid credentials 3. User requests to open shop “MyShop” with a description	1. System verifies the user session 2. System confirms the shop name is unique 3. System creates the shop 4. System assigns ownership to the user 5. Confirmation is returned to the user
testCreateShop_WithGuestToken_ShouldFail	1. Guest enters the system 2. Guest requests to create shop “MyShop”	1. Shop creation is rejected (unauthorized) 2. Total shop count remains unchanged 3. No “MyShop” exists

3.4

a.

Use case: **Rate shop**

1. **Actors**: Registered

2. **Trigger**: User requests to rate a shop

3. **Precondition**:

    1. User is logged in

    2. User has purchased from the shop or product

4. **Parameters**: SessionToken, ShopID, Rating

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. System: verifies that the user has made a purchase related to the given ShopID

    4. System: stores the rating and associates it with the corresponding shop

    5. System: updates the average rating

    6. System: sends confirmation to the user

6. **Alternative flows:**

User not logged:

    >>System notifies the user that they are not logged in

User did not make a purchase:

    >>System notifies the user that a purchase is required before rating

Test Name	Setup & Parameters	Expected Results
rateShopTest	1. User 123 is created 2. User 123 logs in 3. User 123 purchased from ShopID 4. User 123 submits a rating	1. System verifies user login 2. System verifies purchase history 3. System stores rating 4. System updates average rating 5. User 123 receives confirmation
testRateShop_WhenNotLoggedIn_ShouldFail	1. User 123 is not logged in 2. User 123 submits a rating	1. System returns "User not logged in" error

rateShop_BeforePurchase_ShouldFail	1. Owner has a shop 2. Registered buyer logs in but makes no purchases 3. Buyer attempts to rate the shop	1. System returns "Purchase required before rating" error

b.

<u>Use case: **Rate product**</u>

1. **Actors**: Registered

2. **Trigger**: User requests to rate a product

3. **Precondition:**

    1. User is logged in

    2. User has purchased the product

4. **Parameters**: SessionToken, ProductID, Rating

5. **Main Scenario:**

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. System: verifies that the user has made a purchase related to the given ProductID

    4. System: stores the rating and associates it with the corresponding product

    5. System: updates the average rating

    6. System: sends confirmation to the user

6. **Alternative flows:**

User not logged:

>>System notifies the user that they are not logged in

User did not make a purchase:

>>System notifies the user that a purchase is required before rating

Test Name	**Setup & Parameters**	**Expected Results**

rateItemTest	1. User 123 is created 2. User 123 logs in 3. User 123 purchases ProductID 4. User 123 submits rating for ProductID	1. System verifies user login 2. System verifies purchase history 3. System stores rating 4. System updates average rating 5. User receives confirmation
RateProductNotLoggedIn	1. User 123 is not logged in 2. User 123 submits rating for ProductID	1. System returns "User not logged in" error
rateItem_BeforePurchase_ShouldFail	1. Owner has a shop with items 2. Registered buyer logs in but makes no purchases 3. Buyer attempts to rate an item by ID	1. System returns "Purchase required before rating" error

3.5

Use case: **Send message to shop**

1. **Actors**: Registered

2. **Trigger**: User requests to send a message to a shop

3. **Precondition**:

   1. User is logged in

   2. Shop exists

4. **Parameters**: SessionToken, ShopID, MessageText

5. **Main Scenario**:

   1. System: authenticate the user using the authentication service

   2. System: verifies that the user with UserID of the session is logged in

   3. System: verifies that the shop identified by ShopID exists

   4. System: stores the message

   5. System: forwards the message to the shop inbox

6. **Alternative flows:**

User not logged:

>>System notifies the user that they are not logged in

Message with invalid details

>>System notifies the user to correct the message

Test Name	Setup & Parameters	Expected Results
testSendMessage_ShouldSucceed	1. User 123 is created 2. User 123 logs in 3. ShopID exists 4. User 123 sends a message	1. System verifies user login 2. System verifies ShopID exists 3. System stores message 4. System forwards message to shop owner 5. User 123 receives confirmation
testSendMessage_EmptyContent_ShouldFail	1. Owner enters the system 2. Owner creates shop "MyShop" 3. Owner attempts to send a message with a valid title but empty content	1. sendMessage is rejected 2. No message is created

3.7

<u>Use case: **View personal purchase history**</u>

1. **Actors**: Registered

2. **Trigger**: User requests to view their purchase history

3. **Precondition**:

    1. User is logged in

4. **Parameters**: SessionToken

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. System: find all previous purchases made by the user

    4. System: sends the data to the user

6. **Alternative flows:**

User not logged:

>>System notifies the user that they are not logged in

Test Name	Setup & Parameters	Expected Results
testViewPersonalOrderHistory_Empty_ShouldReturnEmptyList	1. User 123 is created 2. User 123 logs in 3. User 123 requests purchase history	1. System verifies user login 2. System retrieves purchase history 3. System sends data to user
testViewPersonalOrderHistory_Empty_ShouldReturnEmptyList	1. User 123 is not logged in 2. User 123 requests purchase history	1. System returns "User not logged in" error

3.9

<u>Use case: **Submit bid offer**</u>

1. **Actors**: Registered

2. **Trigger**: User submits a bid on a product

3. **Precondition**:

    1. User is logged in

    2. Product is open for bidding

4. **Parameters**: SessionToken, ProductID, BidAmount

5. **Main Scenario:**

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. System: verifies that the product is available for bidding

    4. System: verifies that the user's able to buy the items, according to the shop PurchasePolicy

    5. System: calculate the item's price according to the shop DiscountPolicy

    6. System: saves the bid offer

    7. System: notifies the shop owner of the bid

6. **Alternative flows:**

User authentication failed:

>>System acknowledges

User not logged:

>>System notifies the user that they are not logged in

The User is not able to send bidding offer due to the PurchasePolicy:

>>System acknowledges

Product not open for bidding:

>>System notifies the user that the product cannot be bid on

Charging failed due to payment service error

>>System acknowledges

Shipping failed due to shipping service error

>>System acknowledges

Test Name	**Setup & Parameters**	**Expected Results**
submitValidBidTest	1. User 123 is registered 2. User 123 logs in 3. Product A is open for bidding 4. User 123 meets PurchasePolicy 5. User 123 submits a valid bid	1. System verifies login and policies 2. System stores the bid 3. System notifies shop owner
onlyHighestBidderCanPurchaseTest	1. Owner opens a shop with an item 2. Two buyers register and each submits a bid 3. Owner accepts the higher bid 4. Both buyers attempt purchase	1. Both bid submissions succeed 2. Owner's acceptance of the higher bid succeeds 3. Buyer1's purchase is rejected 4. Buyer2's purchase succeeds 5. Buyer2's order history contains one order
counterBidWorkflowTest	1. Owner opens shop "ShopCB" with item A 2. Buyer submits initial bid of 8.0 on item A 3. Owner issues a counter-offer at 10.0 (rejecting the original bid) 4. Buyer accepts the counter-offer 5. Buyer completes purchase of the countered bid	1. Initial bid is accepted 2. Counter-offer creation succeeds 3. Buyer's acceptance succeeds 4. Purchase after acceptance succeeds 5. Buyer's order history shows one order

AuctionPurchasePaymentFailure	1. User 123 wins auction 2. Payment service fails	1. System returns 'Payment failed' message
AuctionPurchaseShippingFailure	1. User 123 wins auction 2. Payment succeeds 3. Shipping service fails	1. System returns 'Shipping failed' message

3.10

Use case: **auction product purchase**

1. **Actors**: Registered

2. **Trigger**: User requests to directly purchase a product

3. **Precondition**:

    1. The auction is currently active (not expired or closed)

    2. The user is logged into the system

4. **Parameters**: SessionToken, ProductID

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. User: Enters a bid amount for the selected product in auction

    4. System: Validates the bid amount is higher than the current highest bid by at least the minimum increment

    5. System: Registers the new bid and updates the highest bid

    –when auction ended–

    6. System: charge the user using the payment service

    7. System: using the shipping service

    8. System: confirms purchase to user

6. **Alternative flows**:

User not logged:

    >>System notifies the user that they are not logged in

Product not available for auction:

    >>System notifies the user that the product is not available

Payment failed:

>>System notifies the user of payment failure

Test Name	**Setup & Parameters**	**Expected Results**
auctionFlowSuccessTest	1. Auction for Product A has ended 2. User 123 is the highest bidder 3. User 123 is logged in	1. System charges user 2. System initiates shipping 3. System confirms purchase
OfferBeforeStart_ShouldFail	1. An auction is created to start in the near future 2. A buyer attempts to place a bid before the auction’s start time	offer is not submitted
PurchaseBeforeEnd_ShouldFail	1. Auction has started but not yet ended 2. Buyer submits a valid bid after start 3. Buyer immediately attempts to purchase the item	1. The purchase request is rejected 2. No order appears in the buyer’s order history
NonWinnerCannotPurchase_WinnerCan	1. Owner opens an auction on Item B 2. auction start 3. Bob submits a lower bid 4. Alice submits a higher bid 5. Wait until auction end 6. Bob attempts purchase 7. Alice attempts purchase	1. Bob’s purchase is rejected and his order history remains empty 2. Alice’s purchase succeeds and her order history contains one auction-based order
AuctionPurchasePaymentFailure	1. User 123 wins auction 2. Payment service fails	1. System returns 'Payment failed' message
AuctionPurchaseShippingFailure	1. User 123 wins auction 2. Payment succeeds 3. Shipping service fails	1. System returns 'Shipping failed' message

3.11

Use case: **Lottery-based product purchase**

1. **Actors**: Registered

2. **Trigger**: User requests to enter a product purchase lottery

3. **Precondition**:

    1. User is logged in

    2. Product is listed as available for lottery

    3. Shop has defined lottery-based purchase policies

4. **Parameters**: SessionToken, ProductID

5. **Main Scenario**:

    1. System: authenticate the user using the authentication service

    2. System: verifies that the user with UserID of the session is logged in

    3. System: verifies that the product exists and is part of a lottery

    4. System: verifies that the user's able to buy the items, according to each shop PurchasePolicy

    5. System: calculate the item's price according to the shop DiscountPolicy

    6. System: registers the user for the lottery

    7. System: sends confirmation of registration

6. **Alternative flows**:

User authentication failed:

>>System acknowledges

One or more of the PurchasePolicy does not approve purchase:

>>System acknowledges

User not logged:

>>System notifies the user that they are not logged in

Product not in lottery mode:

>>System notifies the user that the product is not available via lottery

Registration failed:

>>System notifies the user that registration for the lottery failed

Test Name	**Setup & Parameters**	**Expected Results**
SuccessfulLotteryRegistration	1. User 123 is logged in 2. Product A is available via	1. System registers user for lottery 2. System sends confirmation

	lottery 3. User meets PurchasePolicy	
LotteryPolicyBlocked	1. User 123 is logged in 2. Product A is available via lottery 3. User does not meet PurchasePolicy	1. System acknowledges policy block
LotteryProductNotAvailable	1. User 123 is logged in 2. Product A is not available for lottery	1. System returns 'Product not in lottery mode' message
LotteryRegistrationFailed	1. User 123 is logged in 2. System fails to register lottery entry	1. System returns 'Lottery registration failed' message

4. פעולות של מבקר-מנוי בתפקידו כבעל חנות:

4.1

a.

Use-case: **Add item by shop owner**

1. **Actor**: shop owner

2. **Trigger:** shop owner requests to add item

3. **Precondition**:

   1. Shop owner is logged-in
   2. Item exists
   3. shop exists
   4. Item doesn’t belong to shop

4. **Parameters:** Session token, Shop id, Item name, category, Item price, Description

5. **Main Scenario:**

   1. **System**: verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with ID shopID exists.
   3. **System:** verify user with UserID is shop owner.
   4. **System:** add item to shop.
   5. **System:** send confirmation to the user that the item was added.

6. **Alternative Flows:**

   3. shop with shopID doesn’t exist.

>> System notifies the user that the shop doesn't exist.

4. User with UserID isn’t the shop owner.

   >> System notifies the user that they don't have permission to add an item.

Test Name	Setup & Parameters	Expected Results
testAddItemToShop_AsOwner_ShouldSucceed	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. Item i1 is created. 5. User u1 requests to add Item i1 to Shop s1.	User u1 receives confirmation that Item i1 was added to Shop s1.
testAddItemToShop_WithNonExistentShop_ShouldFail	1. User u1 is created. 2. User u1 logs in. 3. Item i1 is created. 4. User u1 requests to add Item i1 to Shop 999.	User u1 is notified that Shop 999 does not exist.
testAddItemToShop_WithNonExistentShop_ShouldFail	1. User u1 is created. 2. User u2 is created. 3. User u2 logs in. 4. Shop s1 is created by User u1. 5. Item i1 is created. 6. User u2 requests to add Item i1 to Shop s1.	User u2 is notified that they do not have permission to add items to Shop s1.
testAddItemToShop_WithDuplicateItem_ShouldFail	1. User u1 is created. 2. Shop s1 is created by User u1. 3. Item i1 is created. 4. User u1 adds item i1 to shop s1. 5. User u1 adds item i1 to shop s1.	User u1 is notified that he can’t add item i1 to shop s1, it already exists.

b.

<u>Use-case: **Remove item by shop owner**</u>

1. **Actor**: shop owner

2. **Trigger:** shop owner requests to remove item

3. **Precondition**:

1. Shop owner is logged-in
2. Item exists
3. shop exists
4. Item belongs to shop

4. **Parameters:** SessionToken, shopID, itemId

5. **Main Scenario:**

1. **System**: verifies that the user with UserID of the session is logged in
2. **System:** verify shop with ID shopID exists.
3. **System:** verify user with UserID is shop owner.
4. **System:** verify item with itemId, belongs to shop.
5. **System:** remove item from shop.

6. **Alternative Flows:**

1. Shop with shopID doesn’t exist.

   >> System notifies the user that the shop doesn't exist.

2. User with ownerId isn’t the shop owner.

   >> System notifies the user that they don't have permission to remove the item.

3. Item doesn’t belong to the shop.

   >> System notifies the user that it can't remove items that don't belong to the shop.

Test Name	**Setup & Parameters**	**Expected Results**
testRemoveItemFromShop_AsOwner_ShouldSucceed	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. Item i1 is created and added to Shop s1. 5. User u1 requests to remove Item i1 from Shop s1.	User u1 receives confirmation that Item i1 was removed from Shop s1.
testRemoveItemFromShop_WithNonExistentItem_ShouldFail	1. Owner has a shop with items 2. Owner attempts to remove an item using an invalid item ID	1. Removal request is rejected 2. Shop’s item count remains the same

		3. All original items are still present
testRemoveItemFromShop_AsNonOwner_ShouldFail	1. Owner has a shop with items 2. Another (non-owner) user attempts to remove one of those items	1. Removal request is rejected 2. Shop's item count remains the same 3. All original items are still present

c.

Use-case: **Change item by shop owner**

1. **Actor**: shop owner

2. **Trigger:** shop owner requests to change item details(name/price/quantity)

3. **Precondition**:

    5. Shop owner is logged-in
    6. Item exists
    7. shop exists
    8. Item belongs to shop

4. **Parameters:** SessionToken, shopID, itemId, newDetail

5. **Main Scenario:**

    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** verify item with itemId belongs to shop.
    5. **System:** change item details.

6. **Alternative Flows:**

    1. shop with shopID doesn't exist.

        >> System notifies the user: shop doesn't exist.

    2. User with UserID isn't the shop owner.

        >> System notifies the user: don't have permission to add items.

    3. Item doesn't belong to the shop.

        >> System notifies the user: can't change items that don't belong to the shop.

    4. User enters non_valid details.

>> System sends user: can’t change item, non-valid details given.

Test Name	Setup & Parameters	Expected Results
Change Item - Success	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. Item i1 is created and added to Shop s1. 5. User u1 requests to change details of Item i1. 6. User u1 enters new details.	User u1 receives confirmation that Item i1 details were updated.
Change Item - Shop Does Not Exist	1. User u1 is created. 2. User u1 logs in. 3. Item i1 is created. 4. User u1 creates Shop s1. 5. User u1 adds Item i1 to Shop s1. 6. User u1 requests to change item i1 from shop s999	User u1 is notified that Shop s999 does not exist.
Change Item - User Not Owner	1. User u1 is created. 2. User u2 is created. 3. User u2 logs in. 4. Shop s1 is created by User u1. 5. User u1 adds item i1 to shop s1. 6. User u2 requests to change Item i1 to Shop s1.	User u2 is notified that they do not have permission to change items from Shop s1.
Change Item - Invalid Details	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. Item i1 is created and added to Shop s1. 5. User u1 requests to change details of Item i1. 6. User u1 enters invalid details.	User u1 is notified that invalid details were provided.

4.2

.a

Use-case: **Add purchase/discount Type**

1. **Actor**: Shop owner

2. **Trigger:** Shop owner requests to add purchase/discount type.

3. **Precondition**:

   1. Shop owner is logged in.
   2. Shop exists.
   3. purchase/discount type does not belong to the shop.

4. **Parameters:** SessionToken, shopID, purchase/discountTypeDetails.

5. **Main Scenario:**

   1. **System**: verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with shopID exists.
   3. **System:** verify user with UserID is shop owner.
   4. **System:** verify purchase/discount type doesn’t already belong to the shop.
   5. **system:** creates purchase/discount type with purchase/discountTypeDetails.
   6. **System:** adds purchase/discount type to shop.

6. **Alternative Flows:**

   1. shop with shopID doesn’t exist.

      >> System notifies the user that shop doen’t exist.

   2. User with UserID isn’t the shop owner.

      >> System notifies the user doesn’t have permission to change the purchase/discount type.

   3. purchase/discount type already belongs to the shop.

      >> System notifies the user that the purchase/discount type already belongs to the shop.

Test Name	Setup & Parameters	Expected Results
Add Purchase/Discount Type - Success	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. Purchase/discount type does not exist in Shop s1 6. User u1 requests to add a purchase/discount type	Purchase/discount type is successfully added to Shop s1, and confirmation is sent

Add Purchase/Discount Type - Shop Not Found	1. User u1 is created 2. User u1 logs in 3. User u1 requests to add a purchase/discount type to non-existing shop	System notifies that the shop does not exist
Add Purchase/Discount Type - Unauthorized	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created by User u2 4. User u1 requests to add a purchase/discount type to Shop s1	System notifies that User u1 does not have permission

b.

Use-case: **Remove purchase/discount Type**

1. **Actor**: Shop owner
2. **Trigger:** Shop owner requests to remove purchase/discount type.
3. **Precondition**:
    4. Shop owner is logged in.
    5. Shop exists.
    6. purchase/discount type belongs to the shop.
4. **Parameters:** SessionToken, shopID, purchase/discountTypeId
5. **Main Scenario:**
    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** verify purchase/discount type with purchase/discountTypeId belongs to the shop.
    5. **System:** remove purchase/discount type from the shop.
6. **Alternative Flows:**
    1. shop with shopID doesn’t exist.

       >> System notifies the user that shop doesn't exist.
    2. User with UserID isn’t the shop owner.

       >> System notifies the user that they don't have permission to change the purchase/discount type.
    3. purchase/discountType is not one of the purchase/discount types of the store.

>> System notifies the user that the purchase/discount type doesn't belong to the shop.

Test Name	Setup & Parameters	Expected Results
Remove Purchase/Discount Type - Success	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. Purchase/discount type is added to Shop s1 6. User u1 requests to remove the purchase/discount type	Purchase/discount type is removed, and confirmation is sent
Remove Purchase/Discount Type - Not Found	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. User u1 requests to remove a non-existing purchase/discount type	System notifies that the purchase/discount type does not exist

c.

Use-case: **Add purchase/discount rules to policy**

1. **Actor**: Shop owner
2. **Trigger:** Shop owner requests to add purchase/discount rules to the shop's policy.
3. **Precondition**:
    1. Shop owner is logged in.
    2. Shop exists.
    3. purchase/discount roles does not belong to the shop's policy.
4. **Parameters:** SessionToken, shopID, purchase/discountPolicyRule.
5. **Main Scenario:**
    1. **System:** verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** verify purchase/discount rule doesn't already belong to the shop's policy.
    5. **system:** add purchase/discount rule to the shop policy.
6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

   >> System notifies the user that the shop doesn't exist.

2. User with UserID isn’t the shop owner.

   >> System notifies the user that he doesn't have permission to change the purchase/discount policy.

3. purchase/discount rule already belongs to the shop policy.

   >> System notifies the user that the purchase/discount rule already belongs to the shop’s policy.

Test Name	**Setup & Parameters**	**Expected Results**
AddPolicyRuleSuccessfully	1. Shop owner is logged in 2. Shop with ID exists 3. Rule is not part of the shop’s policy	1. System adds the rule to the policy 2. System confirms success
AddPolicyRule_RuleAlreadyExists	1. Shop owner is logged in 2. Rule already exists in the policy	1. System notifies that the rule already belongs to the policy

d.

Use-case: **Remove purchase/discount rule from policy**

1. **Actor**: Shop owner

2. **Trigger:** Shop owner requests to remove purchase/discount rule.

3. **Precondition**:

   1. Shop owner is logged in.
   2. Shop exists.
   3. purchase/discount rule belongs to the shop’s policy.

4. **Parameters:** SessionToken, shopID, purchase/discountPolicyRule

5. **Main Scenario:**

   1. **System:** verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with shopID exists.
   3. **System:** verify user with UserID is shop owner.
   4. **System:** verify purchase/discountPolicy rule belongs to the shop’s policy.
   5. **System:** remove purchase/discount rule from the shop’s policy.

6. **Alternative Flows:**

   1. shop with shopID doesn’t exist.

>> System notifies the user that the shop doesn't exist.

2. User with UserID isn’t the shop owner.

   >> System notifies the user that he doesn't have permission to change the purchase/discount policy.

3. purchase/discount rule is not one of the purchase/discount policies of the store.

   >> System notifies the user that the purchase/discount rule doesn’t belong to the shop.

Test Name	**Setup & Parameters**	**Expected Results**
RemovePolicyRuleSuccessfully	1. Shop owner is logged in 2. Shop exists 3. Rule exists in the policy	1. System removes the rule from the policy 2. System confirms success
RemovePolicyRule_RuleNotExists	1. Shop owner is logged in 2. Rule does not exist in the policy	1. System notifies that the rule doesn’t belong to the shop

4.3

Use-case: **Add shop owner**

1. **Actor**: Shop owner, Nominee

2. **Trigger:** Shop owner requests to add nominee as shop owner.

3. **Precondition**:

   1. Shop owner is logged in.
   2. Shop exists.
   3. Nominee is a registered user.
   4. Nominee isn’t already a shop owner of the shop.

4. **Parameters:** SessionToken, shopID, NomineeId

5. **Main Scenario:**

   1. **System**: verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with shopID exists.
   3. **System:** verify user with UserID is shop owner.
   4. **System:** verify user with NomineeId is registered.
   5. **System:** sends a nomination message to the nominee .

6. **Nominee:** accepts the nomination message.
7. **System:** Sends confirmation to the nominee and the shop owner that the nominee was added as shop owner.

6. **Alternative Flows:**

1. shop with shopID doesn't exist.

   >> System notifies the user that the shop doesn't exist.

2. User with UserID isn't the shop owner.

   >> System notifies the user that he doesn't have permission to change the purchase/discount policy.

3. User with NomineeId isn't registered.

   >> System notifies the user that can't add a non-registered user as a shop owner.

4. User with NomineeId declined the nomination message .

   >> System notifies the user that the nominee declined the shop owner's nomination .

Test Name	**Setup & Parameters**	**Expected Results**
AddShopOwnerSuccessfully	1. Shop owner is logged in 2. Shop exists 3. Nominee is registered 4. Nominee is not already an owner 5. Nominee accepts nomination	1. System adds the nominee as owner 2. System confirms to both parties
AddShopOwner_NomineeDeclined	1. Shop owner is logged in 2. Nominee declines nomination	1. System notifies that the nomination was declined
AddShopOwner_NomineeNotRegistered	1. Shop owner is logged in 2. Nominee is not a registered user	1. System notifies that nominee is not registered

4.4

<u>Use-case: **Remove shop owner**</u>

1. **Actor**: Shop owner

2. **Trigger:** Shop owner requests to remove a different shop owner.

3. **Precondition**:

   1. Shop owner is logged in.

2. Shop exists.
3. Dismissed owner is a registered user.

4. **Parameters:** SessionToken, shopID, removeId

5. **Main Scenario:**

1. **System**: verifies that the user with UserID of the session is logged in
2. **System:** verify shop with shopID exists.
3. **System:** verify user with removeId is a shop owner of shopID.
4. **System:** verify user with removeId was promoted by user with UserID.
5. **System:** recursively removing all shop owners and managers promoted by removeId.
6. **System:** remove user with removeId as shop owner.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

   >> System notifies the user that the shop doesn't exist.

2. User with UserID isn’t the shop owner.

   >> System notifies the user doesn’t have permission to remove owner

3. User with removeId isn’t a shop owner of the shop.

   >> System notifies the user that the user with removeId isn’t a shop owner.

4. User with removeId wasn’t promoted by the user with ownerId .

   >> System notifies the user that he cannot remove user with removeId from shop owner role because he wasn’t promoted by him.

Test Name	**Setup & Parameters**	**Expected Results**
RemoveShopOwnerSuccessfully	1. Shop owner is logged in 2. Shop exists 3. removeId is a shop owner promoted by the user	1. System recursively removes the owner and all sub-promotions
RemoveShopOwner_NotPromotedByUser	1. Shop owner is logged in 2. removeId was not promoted by the user	1. System notifies that the user cannot remove this owner

4.6

Use-case: **Add shop manager**

1. **Actor**: Shop owner

2. **Trigger:** Shop owner requests to add nominee as shop manager.

3. **Precondition**:

    5. Shop owner is logged in.
    6. Shop exists.
    7. Nominee is a registered user.
    8. Nominee isn’t already an owner or manager of the shop.

4. **Parameters:** SessionToken, shopID, NomineeId, permissions

5. **Main Scenario:**

    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** verify user with NomineeId is registered.
    5. **System:** verify user with NomineeId isn’t a shop owner or manager of shopID.
    6. **System:** sends a nomination message to the nominee .
    7. **Nominee:** accepts the nomination message.
    8. **System:** set user with nomineeId to have the permissions defined by his appointee.

6. **Alternative Flows:**

    1. shop with shopID doesn’t exist.

       >> System notifies the user that shop doen’t exist.

    2. User with UserID isn’t the shop owner.

       >> System notifies the user doesn’t have permission to add a manager.

    3. User with NomineeId isn’t registered.

       >> System notifies the user that can’t add a non-registered user as a shop owner.

    4. User with NomineeId is a shop owner or manager of the shop.

       >> System notifies the user that the nominee is already a shop owner or manager.

    5. User with NomineeId declined the nomination message .

       >> System notifies the user that the nominee declined the shop manager’s nomination .

Test Name	Setup & Parameters	Expected Results
Add Shop Manager - Success	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. User u2 is created and registered 6. User u1 nominates User u2 as a manager 7. User u2 accepts the nomination	User u2 is assigned as manager with defined permissions, confirmation sent
Add Shop Manager - Nominee Not Registered	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. User u2 does not exist 6. User u1 nominates User u2 as manager	System notifies that the nominee is not a registered user

4.7

Use-case: **Set manager permissions**

1. **Actor**: Shop owner
2. **Trigger:** Shop owner requests to change shop manager permissions.
3. **Precondition**:
    1. Shop owner is logged in.
    2. Shop exists.
    3. Shop manager is a manager of the shop.
4. **Parameters:** SessionToken, shopID, managerId, newPermission
5. **Main Scenario:**
    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** verify user with managerId is a shop manager of shopID.
    5. **System:** verify user with ownerId is appointee of user with managerId.
    6. **System:** set user with managerId to have the permissions defined by his appointee.
6. **Alternative Flows:**
    1. shop with shopID doesn't exist.

       >> System notifies the user that shop doen't exist.

2. User with UserID isn't the shop owner.

   >> System notifies the user doesn't have permission to change managers permissions.

3. User with managerId isn't a manager of the shop.

   >> System notifies the user that the manager isn't a shop manager.

4. User with ownerId isn't the appointee of the user with managerId .

   >> System notifies the user that he cannot set permissions to a manager that is not his appointee.

Test Name	**Setup & Parameters**	**Expected Results**
Set Manager Permissions - Success	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. User u2 is assigned as manager 6. User u1 updates User u2's permissions	Permissions are updated, confirmation sent

4.9

Use-case: **Close shop by founder**

1. **Actor**: Shop Founder

2. **Trigger:** Shop Founder requests to close a shop.

3. **Precondition**:

   1. Shop founder is logged in.
   2. Shop exists.
   3. Shop is open.

4. **Parameters:** SessionToken, shopID

5. **Main Scenario:**

   1. **System**: verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with shopID exists.
   3. **System:** verify user with UserID is shop founder.
   4. **System:** verify shop with shopID is open.
   5. **System:** closes shop with shopID.

6. **System:** Sends confirmation to the shop manager and owners that the shop was closed.

6. **Alternative Flows:**

1. shop with shopID doesn't exist.

   >> System notifies the user that shop doen't exist.

2. User with founderId isn't the shop founder.

   >> System notifies the user that it doesn't have permission to close the shop.

3. The shop with shopID is already closed.

   >> System notifies the user that the shop is already closed.

4.11

Use-case: **Get shop members permission info**

1. **Actor**: Shop Owner

2. **Trigger:** Shop Owner requests to get all members permission info.

3. **Precondition**:

   1. Shop owner is logged in.
   2. Shop exists.
   3. Shop owner is an owner of the shop.

4. **Parameters:** SessionToken, shopID

5. **Main Scenario:**

   1. **System**: verifies that the user with UserID of the session is logged in
   2. **System:** verify shop with shopID exists.
   3. **System:** verify user with UserID is shop owner.
   4. **System:** show the owner all the members info(including permissions for managers).

6. **Alternative Flows:**

   1. shop with shopID doesn't exist.

      >> System notifies the user that the shop doesn't exist.

   2. User with UserID isn't the shop owner.

      >> System notifies the user that it doesn't have permission to watch members' info.

Test Name	Setup & Parameters	Expected Results

Get Shop Members Permission Info - Success	1. User u1 is created 2. User u1 logs in 3. Shop s1 is created 4. User u1 is assigned as owner 5. User u1 requests shop members permission info	Shop members' information (including manager permissions) is displayed

4.12

Use-case: **Shop owner Responds to message**

1. **Actor**: Shop Owner
2. **Trigger:** Shop Owner requests to respond to a message.
3. **Precondition**:
    1. Shop owner is logged in.
    2. Shop exists.
    3. Shop owner is an owner of the shop.
    4. Message was sent to shop inbox by userId
4. **Parameters:** SessionToken, shopID, userId, message
5. **Main Scenario:**
    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** sends the given message to the user with userId as a response.
6. **Alternative Flows:**
    1. shop with shopID doesn’t exist.

       >> System notifies the user that the shop doesn't exist.
    2. User with UserID isn’t the shop owner.

       >> System notifies the user that it doesn't have permission to respond to users messages.

Test Name	**Setup & Parameters**	**Expected Results**

Shop Owner Responds to Message - Success	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. User u2 sends a message to Shop s1. 5. User u1 responds to the message.	User u2 receives a response from User u1.

4.13

<u>Use-case: **Shop owner gets purchase history**</u>

1. **Actor**: Shop Owner
2. **Trigger:** Shop Owner requests to shop purchase history.
3. **Precondition**:
    1. Shop owner is logged in.
    2. Shop exists.
    3. Shop owner is an owner of the shop.
4. **Parameters:** SessionToken, shopID.
5. **Main Scenario:**
    1. **System**: verifies that the user with UserID of the session is logged in
    2. **System:** verify shop with shopID exists.
    3. **System:** verify user with UserID is shop owner.
    4. **System:** returns shop purchase history.
6. **Alternative Flows:**
    1. shop with shopID doesn't exist.

        >> System notifies the user that shop doen't exist.
    2. User with UserID isn't the shop owner.

        >> System notifies the user that it doesn't have permission to watch the purchase history.

Test Name	Setup & Parameters	Expected Results

Get Shop Purchase History - Success	1. User u1 is created. 2. User u1 logs in. 3. Shop s1 is created by User u1. 4. Shop s1 has purchase history. 5. User u1 requests purchase history of Shop s1.	User u1 receives the purchase history of Shop s1.

5. פעולות של מבקר-מנוי בתפקידו כמנהל חנות:

5

a.

<u>Use case: **View shop content**</u>

1. **Actors**: Shop Manager

2. **Trigger**: User accesses the shop management dashboard

3. **Precondition**:

    1. User is logged in

    2. User is a manager of the shop with view permissions

4. **Parameters**: SessionToken, ShopID

5. **Main Scenario**:

    1. **System**: verifies the user is logged, by UserID by the SessionToken
    2. **System**: verifies the user is the shop manager of ShopID
    3. **System**: verifies that the user has view permission
    4. **System**: retrieves shop data

5. **System**: sends the information to the user

6. **Alternative flows**:

User not logged:

>>System notifies the user that they are not logged in

User not a manager in this shop:

>>System denies access

User lacks permission to view data:

>>System denies access and notifies the user

Test Name	Setup & Parameters	Expected Results
SuccessfulViewShopContent	1. User 123 is created 2. User 123 logs in 3. User 123 is a manager of ShopID with view permission 4. User 123 accesses the shop dashboard	1. System verifies user session 2. System confirms the user is a manager 3. System checks view permissions 4. System retrieves and sends shop data to user
ViewShopNotLoggedIn	1. User 123 is not logged in 2. User 123 accesses the shop dashboard	1. System returns "User not logged in" error
UserNotManagerOfShop	1. User 123 is created 2. User 123 logs in 3. User 123 is not a manager of ShopID 4. User 123 accesses the shop dashboard	1. System returns "Access denied" error
UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 is a manager of ShopID but lacks view permission 4. User 123 accesses the shop dashboard	1. System returns "Permission denied" error

b.

<u>Use case: **Edit product inventory**</u>

1. **Actors**: Shop Manager

2. **Trigger**: User requests to edit product data

3. **Precondition**:

1. User is logged in
2. User is a manager of the shop

3. User has permission to edit product inventory

Parameters: SessionToken, ShopID, ProductID, UpdatedProductData

5. **Main Scenario**:

1. **System**: verifies that the session belongs to a manager of the shop
2. **System**: verifies that the user has permission to edit inventory
3. **System**: verifies that the product exists in the shop
4. **System**: updates the product data
5. **System**: sends confirmation to the user

6. **Alternative flows:**

User not logged:

>>System notifies the user that they are not logged in

Product not found:

>>System notifies the user that the product doesn’t exist

User lacks permission:

>>System denies the request

Test Name	Setup & Parameters	Expected Results
SuccessfulEditProduct	1. User 123 is created 2. User 123 logs in 3. User 123 is a manager of ShopID with inventory edit permissions 4. ProductID exists 5. User 123 updates product data	1. System verifies session and permissions 2. System verifies product existence 3. System updates product data 4. System confirms update to user
EditProductNotLoggedIn	1. User 123 is not logged in 2. User 123 requests to edit ProductID	1. System returns "User not logged in" error
ProductNotFound	1. User 123 is created 2. User 123 logs in 3. ProductID does not exist 4. User 123 attempts to edit product	1. System returns "Product not found" error
UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 is a manager but lacks inventory edit permissions	1. System returns "Permission denied" error

	4. User 123 attempts to edit product	

c.

Use case: **Edit purchase policy**

1. **Actors**: Shop Manager

2. **Trigger**: User requests to modify the shop’s purchase policy

3. **Precondition**:

   1. User is logged in
   2. User has edit permission for purchase policies

4. **Parameters**: SessionToken, ShopID, UpdatedPurchasePolicy

5. **Main Scenario**:

   1. **System**: verifies session and permissions
   2. **System**: updates the purchase policy for the shop
   3. **System**: sends confirmation to the user

6. **Alternative flows:**

User lacks permission:

>>System denies the request

Invalid policy format:

>>System notifies the user

Test Name	Setup & Parameters	Expected Results
SuccessfulEditPurchasePolicy	1. User 123 is created 2. User 123 logs in 3. User 123 has purchase policy edit permissions 4. User 123 submits UpdatedPurchasePolicy	1. System verifies session and permissions 2. System updates purchase policy 3. System confirms update to user
EditPolicyNotLoggedIn	1. User 123 is not logged in 2. User 123 submits UpdatedPurchasePolicy	1. System returns "User not logged in" error
UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 lacks purchase policy edit permissions 4. User 123 submits UpdatedPurchasePolicy	1. System returns "Permission denied" error
InvalidPolicyFormat	1. User 123 is created 2. User 123 logs in 3. User 123 has permission	1. System returns "Invalid policy format" error

	4. User 123 submits an invalid policy	

d.

<u>Use case: **Edit discount policy**</u>

1. **Actors**: Shop Manager

2. **Trigger**: User requests to modify the shop’s discount policy

3. **Precondition**:

    1. User is logged in

    2. User has edit permission for discount policies

4. **Parameters**: SessionToken, ShopID, UpdatedDiscountPolicy

5. **Main Scenario**:

    1. **System**: verifies session and permissions
    2. **System**: updates the discount policy
    3. **System**: sends confirmation to the user

6. **Alternative flows:**

User lacks permission:

    >>System denies the request

Invalid policy format:

    >>System notifies the user

Test Name	Setup & Parameters	Expected Results
SuccessfulEditDiscountPolicy	1. User 123 is created 2. User 123 logs in 3. User 123 has discount policy edit permissions 4. User 123 submits UpdatedDiscountPolicy	1. System verifies session and permissions 2. System updates discount policy 3. System confirms update to user
EditDiscountPolicyNotLoggedIn	1. User 123 is not logged in 2. User 123 submits UpdatedDiscountPolicy	1. System returns "User not logged in" error
UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 lacks discount policy edit permissions 4. User 123 submits UpdatedDiscountPolicy	1. System returns "Permission denied" error

InvalidDiscountPolicyFormat	1. User 123 is created 2. User 123 logs in 3. User 123 has permission 4. User 123 submits an invalid policy	1. System returns "Invalid policy format" error

e.

Use case: **Add purchase policy**

1. **Actors**: Shop Manager

2. **Trigger**: User requests to add a new purchase policy

3. **Precondition**:

    1. User is logged in

    2. User has permission to add purchase policies

4. **Parameters**: SessionToken, ShopID, NewPurchasePolicy

5. **Main Scenario**:

System: verifies session and permissions

System: validates the new policy

System: adds the purchase policy to the shop

System: sends confirmation to the user

6. **Alternative flows:**

Policy invalid:

    >>System notifies the user

User lacks permission:

    >>System denies the request

Test Name	Setup & Parameters	Expected Results
SuccessfulAddPurchasePolicy	1. User 123 is created 2. User 123 logs in 3. User 123 has permission to add policies 4. User 123 submits NewPurchasePolicy	1. System verifies session and permissions 2. System validates policy 3. System adds purchase policy 4. System confirms update to user
AddPurchasePolicyNotLoggedIn	1. User 123 is not logged in 2. User 123 submits NewPurchasePolicy	1. System returns "User not logged in" error

UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 lacks purchase policy add permissions 4. User 123 submits NewPurchasePolicy	1. System returns "Permission denied" error
InvalidPurchasePolicy	1. User 123 is created 2. User 123 logs in 3. User 123 submits an invalid policy	1. System returns "Invalid policy format" error

f.

Use case: **Add discount policy**

1. **Actors**: Shop Manager

2. **Trigger**: User requests to add a new discount policy

3. **Precondition**:

    1. User is logged in

    2. User has permission to add discount policies

4. **Parameters**: SessionToken, ShopID, NewDiscountPolicy

5. **Main Scenario**:

    1. System: verifies session and permissions

    2. System: validates the new policy

    3. System: adds the discount policy to the shop

    4. System: sends confirmation to the user

6. **Alternative flows**:

Policy invalid:

    >>System notifies the user

User lacks permission:

    >>System denies the request

Test Name	Setup & Parameters	Expected Results
SuccessfulAddDiscountPolicy	1. User 123 is created 2. User 123 logs in	1. System verifies session and permissions 2. System validates policy 3. System adds discount policy 4.

	3. User 123 has permission to add discount policies 4. User 123 submits NewDiscountPolicy	System confirms update to user
AddDiscountPolicyNotLoggedIn	1. User 123 is not logged in 2. User 123 submits NewDiscountPolicy	1. System returns "User not logged in" error
UserLacksPermission	1. User 123 is created 2. User 123 logs in 3. User 123 lacks discount policy add permissions 4. User 123 submits NewDiscountPolicy	1. System returns "Permission denied" error
InvalidDiscountPolicy	1. User 123 is created 2. User 123 logs in 3. User 123 submits an invalid policy	1. System returns "Invalid policy format" error

6. פעולות של מבקר-מנוי בתפקידו כמנהל מערכת המסחר:

6.1

Use case: **Closing shop as System manager**

1. **Actors**: System manager, shop related users

2. **Trigger**: System manager choosing to close his shop

3. **Precondition**: The user was in the system as user and the shop is in the system

4. **Parameters**: ShopID, UserID

5. **Main Scenario**:

    1. System manager: closing a shop

    2. System: Delete the subscription of the users who are related to the shop

    3. System: Sends a message to all the deleted subscribers about the closing of the shop

6. **Alternative flows**:

System manager: attempting to close a closed shop

    >>System: send an “unauthorized action” error to the system manager

Test Name	Setup & Parameters	Expected Results
SuccessfulShopClosing	1. System manager exist in the system 2. System manager logs in 3. System manager choses an online shop 4. System manager closing the shop	1. The subscribed users are deleted from the subscription to the shop 2. A message is sent to all the subscribed users for the shop
ClosingClosedShop	1. System manager exist in the system 2. System manager logs in 3. System manager choses a closed shop 4. System manager attempting to close the shop	1. A "unauthorized action" error is sent to the system manager

6.6 Use-case: **Suspend User**

1. **Actor:** System Administrator
2. **Trigger:** Administrator requests to suspend a user
3. **Precondition:**
   1. system manager is logged in
   2. Target user exists and is currently active
   3. Suspension rule is not already active for the user
4. **Parameters:** SessionToken, TargetUserID, SuspensionDuration
5. **Main Scenario:**
   1. **System:** Authenticates the admin using the authentication service
   2. **System:** Verifies that the target user exists
   3. **System:** Applies the suspension for a limited or permanent duration
   4. **System:** Restricts the user to view-only actions
6. **Alternative Flows:**

   User does not exist:

   >> System notifies the admin that the user does not exist

   Admin authentication failed:

   >> System denies action and logs the failure

Test Name	**Setup & Parameters**	**Expected Results**

SuspendUserSuccessfully	1. Admin is logged in 2. Target user exists and is active 3. Admin selects suspension duration	1. User is suspended 2. User can only view content
SuspendUser_UserNotExist	1. Admin is logged in 2. Target user does not exist	1. System notifies admin that user does not exist

6.7 Use-case: **Unsuspend User**

1. **Actor:** System Administrator
2. **Trigger:** Administrator requests to lift the suspension on a user
3. **Precondition:**
   1. Administrator is logged in
   2. Target user is currently suspended
4. **Parameters:** SessionToken, TargetUserID
5. **Main Scenario:**
   1. **System:** Authenticates the admin using the authentication service
   2. **System:** Verifies that the target user is suspended
   3. **System:** Lifts the suspension
   4. **System:** Restores full user permissions
6. **Alternative Flows:**

   User is not suspended:

   >> System notifies the admin that the user is not suspended

   Admin authentication failed:

   >> System denies action

Test Name	**Setup & Parameters**	**Expected Results**
UnsuspendUserSuccessfully	1. Admin is logged in 2. Target user is suspended	1. System removes suspension 2. User regains full access
UnsuspendUser_UserNotSuspended	1. Admin is logged in 2. Target user is not suspended	1. System notifies admin that user is not suspended

6.8 Use-case: **View Suspended Users**

1. **Actor:** System Administrator
2. **Trigger:** Administrator views the list of suspended users
3. **Precondition:**
    1. Administrator is logged in
4. **Parameters:** SessionToken
5. **Main Scenario:**
    1. **System:** Authenticates the admin
    2. **System:** Retrieves suspended users from the system
    3. **System:** Displays suspension information including start date, duration, and end date
6. **Alternative Flows:**
   Admin authentication failed:
   >> System denies access to the suspension list

Test Name	**Setup & Parameters**	**Expected Results**
ViewSuspendedUsersSuccessfully	1. system manager is logged in	1. System displays list of suspended users with suspension details
ViewSuspendedUsers_AuthFails	1. Admin is not logged in or session is invalid	1. System denies access to suspension list