*מידול מערכת:*

**גרסה 0:**

תוכן עניינים

[מודל מחלקות לבן: 2](#_Toc194947039)

[דיאגרמת ארכיטקטורה: 2](#_Toc194947040)

[תרחישי שימוש: 3](#_Toc194947041)

[מערכת: 3](#_Toc194947042)

[משתמשים: 6](#_Toc194947043)

[1. פעולות מבקר-אורח: 6](#_Toc194947044)

[2. פעולות קנייה של מבקר-אורח: 8](#_Toc194947045)

[.3 פעולות קנייה של מבקר-מנוי בשוק: 14](#_Toc194947046)

[.4פעולות של מבקר-מנוי בתפקידו כבעל חנות: 21](#_Toc194947047)

[.5 פעולות של מבקר-מנוי בתפקידו כמנהל חנות: 33](#_Toc194947048)

[.6 פעולות של מבקר-מנוי בתפקידו כמנהל מערכת המסחר: 37](#_Toc194947049)

[מבחני קבלה: 38](#_Toc194947050)

[מערכת: 38](#_Toc194947051)

[משתמשים: 40](#_Toc194947052)

[1. פעולות מבקר-אורח: 40](#_Toc194947053)

[2. פעולות קנייה של מבקר-אורח: 42](#_Toc194947054)

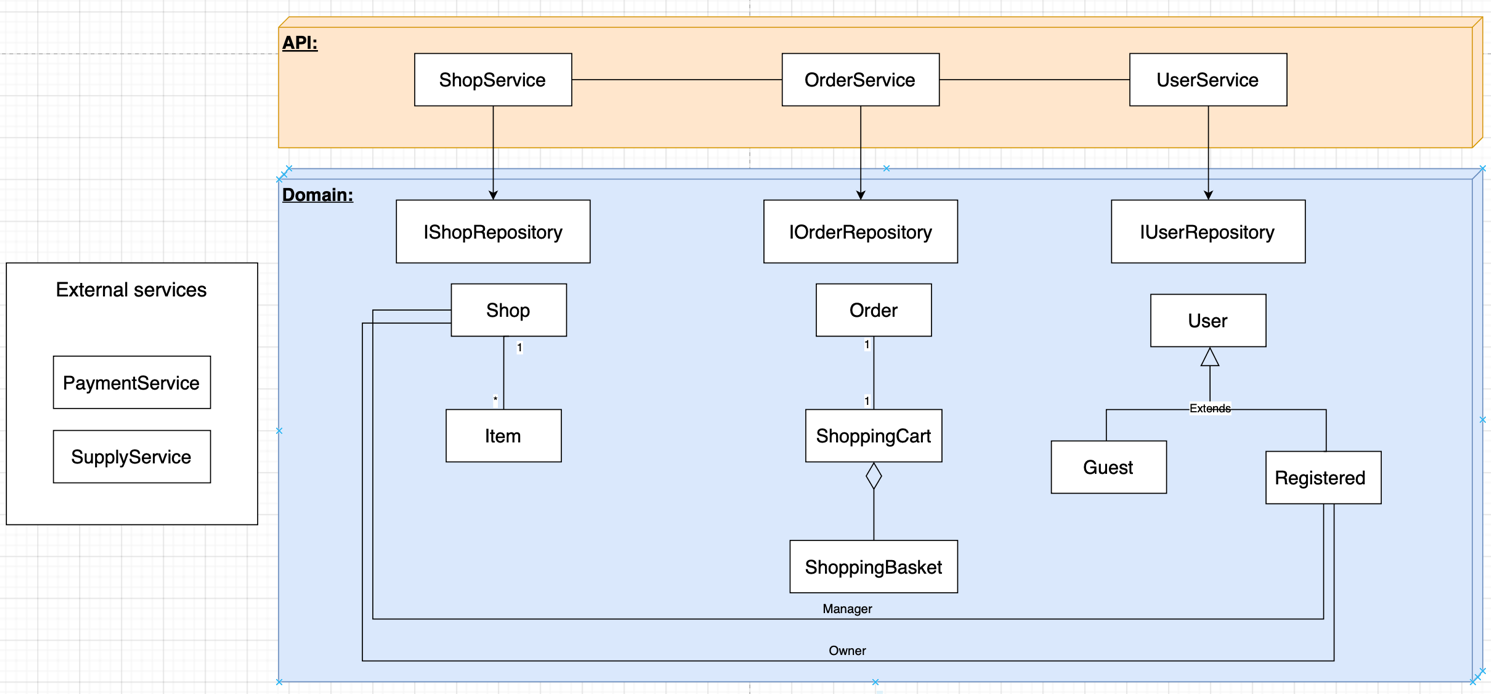
[3.פעולות קנייה של מבקר-מנוי בשוק: 44](#_Toc194947055)

[4.פעולות של מבקר-מנוי בתפקידו כבעל חנות: .................................................................................................47](#_Toc194947056)

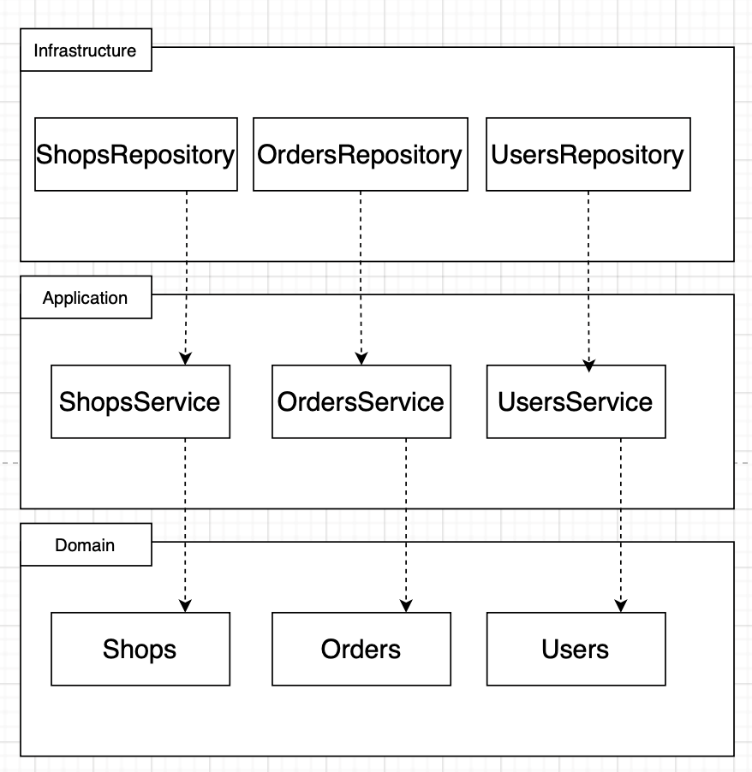
[5.פעולות של מבקר-מנוי בתפקידו כמנהל חנות: ...............................................................................................52](#_Toc194947057)

[6.פעולות של מבקר-מנוי בתפקידו כמנהל מערכת המסחר : 55](#_Toc194947058)

# מודל מחלקות לבן:

****

# דיאגרמת ארכיטקטורה:

****

# תרחישי שימוש:

## 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.

4. **Parameters:**

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:** initialize the trading system.
5. **System:** initialize connections to external services.

6. **Alternative Flows:**

1. the user failed to login.

>> System notifies the user that he failed to log in.

1. failed to connect to external services.

>> System notifies the user that external services are unavailable.

2.1  
Use-case: **update external services**

1. **Actor**: system manager

2. **Trigger:** the system manager updates the available external services.

3. **Precondition**:

There is a user with the system manager role.

4. **Parameters:** external service(optional).

5. **Main Scenario:**

1. **User:** logs in to the system
2. **System:** verify user successfully logged in.
3. **User:** add/remove external service.
4. **System:** verify user has system manager role.
5. **System:** sends confirmation message to user if update was successful.

6. **Alternative Flows:**

1. the user failed to login.

>> System notifies the user that he failed to log in.

1. the user doesn't have system manager permission .

>> System notifies the user that he doesn’t have permission.

1. failed to update external services.

>> System notifies the user that external services are unavailable.

2.2  
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. charge failed.

>> System notifies the user 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.

4.  
Use-case: **Real-time notification**

1. **Actor**: user

2. **Trigger:** the user receives a notification.

3. **Precondition**:

1. user is registered
2. user is logged in.

4. **Parameters:**

5. **Main Scenario:**

1. **System:** sends a real-time notification to a user about an action in the system.

6. **Alternative Flows:**

1. user is not logged in.

>> System will not send a real-time notification.

5.  
Use-case: **delayed notification**

1. **Actor**: user

2. **Trigger:** the user receives a notification.

3. **Precondition**:

1. user is registered
2. user isn’t logged in.

4. **Parameters:**

5. **Main Scenario:**

1. **System:** sends a notification to a user about an action in the system and saves the notification to appear when the user logs in.

6. **Alternative Flows:**

1. user is logged in.

>> System will send a real-time notification.

## 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 was not in the system

4. **Parameters**: void

5. **Main Scenario:**

**1. System**: Loads login page

**2. User**: Choosing guest

**3. System**: Register the active user as guest and gives him a temporary UserID

**4. System**: Creates an empty shopping cart

**5**. **System**: Creates session token for him

6. **Alternative flows**: Non

1.2  
Use case: **exit as Guest**

1. **Actors**: User

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

3. **Precondition**:

1. The user was in the system as guest

4. **Parameters**: Session token

5. **Main Scenario**:

**1. User**: logged in as guest

**2. User**: Choosing exit

**3. System**: delete his shopping cart, the temporary UserID and the session token.

6. **Alternative flows**: Non

1.3  
Use case: **Register**

1. **Actors**: User

2. **Trigger**: User Register to the system

3. **Precondition**:

1. The user was is in the system as guest

4. Parameters: session token, username, password, date of birth

5. **Main Scenario**:

**1. User**: Choosing Register

**2. System**: Loads register page

**3. User**: putting personal information

**4. System**: validate the username

**5**. **System**: Register a new user to the system

**6. System**: Transfer the guest cart to the registered shopping cart

6. **Alternative flows:**

**1. User**: Choosing Register

**2. System**: Loads register page

**3. User**: putting false personal information

**4. System**: Sends an error message since false information was inserted

**5. System**: waiting for the information to be corrected

1.4

Use case: **Login as Registered**

1. **Actors**: User

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

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

4. **Parameters**: Session token , Nickname, Password

5. **Main Scenario**:

**1. System**: checks if user is not logged in

**2. System**: Loads login page

**3. User**: Logs in as Registered

**4. User**: Enters nickname and password

**5. System**: Checks if the nickname matches the password

**6. System**: Confirm they match

**7. System**: Register the active user as the user linked to the nickname in the session token

6. **Alternative flows**:

**1. System**: checks if user is not logged in

**2. System**: Loads login page

**3. User**: Logs in as Registered

**4. User**: Enters nickname and password

**5. System**: Doesn’t find a match

**6. System**: Sends an error message and waits for next input

### 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**: void

5. **Main Scenario**:

**1. System**: get all the shops

**2. System**: get all the items

**3. System**: send the data to the user

6. **Alternative flows**: Non

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, KeyText, 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**: Non

b.

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

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

2.3

Use case**: Item adding to shop basket**

1. **Actors**: Guest

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

3. **Precondition**:

1. Shop exists

2. Items exists

3. Items available

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

5. **Main Scenario:**

**1. System**: search ShopID

**2. System**: validates that the shop exists

**3. System**: search all the mentioned items using ItemsID

**4. System**: validates that the items exist

**5. System**: validates that the items are available

**6. System**: search the respective basket for the given ShopID and UserID of the session

**7. System**: save the items in the user’s basket

**8. System**: send confirmation to user

6. **Alternative flows**:

Shop was not found:

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

Items were not found:

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

Items were not available:

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

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**: search CartID

**2. System**: validates that the cart exists

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

**4. System**: send the cart data to the user

6. **Alternative flows**:

Cart was not found:

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

Cart was not the user’s cart:

>>System notifies to the user that it’s not their cart

b.

Use case: **Change 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**: search CartID

**2. System**: validates that the cart exists

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

**4. System**: search the selected items with ItemsToDeleteID

**5. System**: validates that all the items exist in the cart

**6. System**: asks for user confirmation

**7. User**: approves deletion

**8. System**: delete the items from the user’s baskets

**9. System**: send confirmation to user

6. **Alternative flows:**

Cart was not found:

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

Cart was not the user’s cart:

>>System notifies to the user that it’s not the respective cart

Items were not found:

>>System notifies to the user that one or more items were not found in the cart

User does not approve purchase

>>System acknowledges

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**: search CartID

**2. System**: validates that the cart exists

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

**4. System**: validates that all the items in the cart available

**5. System**: Calculate the final price for all the items, after each item discount

**6. System**: asks for user confirmation

**7. User**: approves deletion

**8. System**: charge the user

**9. System**: send confirmation to user

6. **Alternative flows:**

Cart was not found:

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

Cart was not the user’s cart:

>>System notifies to the user that it’s not the respective cart

Items were not found:

>>System notifies to the user that one or more items were not found in the cart

Items were not available:

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

User does not approve purchase:

>>System acknowledges

**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, CartID

5. **Main Scenario**:

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

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

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

**4. System**: changes the user status to Guest

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

6. **Alternative flows:**

User not logged:

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

Cart was not the user’s cart:

>>System notifies to the user that it’s not their cart

Cart was not saved:

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

3.2

Use case: **Open shop**

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**: 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 and owner of the new shop

**System**: sends confirmation to the user

6. **Alternative flows**:

User not logged:

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

User lacks permission:

>>System notifies the user that they cannot open a shop

Shop name already exists:

>>System notifies the user to choose another name

Shop was not created:

>>System notifies to the user that the Shop was not created

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

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

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

**4. System**: updates the average rating

**5. 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

Rate was not saved:

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

b.

Use case: **Rate product**

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

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

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

**4. System**: updates the average rating

**5. 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

Rate was not saved:

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

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

**2. System**: verifies that the shop identified by ShopID exists

**3. System**: stores the message

**4. System**: forwards the message to the shop owner

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

6. **Alternative flows:**

User not logged:

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

Shop not found:

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

The message was not forwarded:

>>System notifies the user that the message was not forwarded

3.7

Use case: **View personal purchase history**

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

**2. System**: find all previous purchases made by the user

**3. System**: sends the data to the user

6. **Alternative flows:**

User not logged:

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

Couldn’t get the history:

>>System notifies the user that the request failed

3.9

Use case: **Submit bid offer**

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

**2. System**: verifies that the product is available for bidding

**3. System**: saves the bid offer

**4. System**: sends confirmation to the user

**5. System**: notifies the shop owner of the bid

6. **Alternative flows:**

User not logged:

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

Product not open for bidding:

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

Bid failed to save:

>>System notifies the user of the failure

3.10

Use case: **Direct product purchase**

1. **Actors**: Registered

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

3. **Precondition**:

1. User is logged in

2. Product exists and is available

3. Shop has defined purchase and delivery policies

4. **Parameters**: SessionToken, ProductID

5. **Main Scenario**:

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

**2. System**: verifies that the product exists and is available for purchase

**3. System**: fetches shop’s purchase and delivery policies

**4. System**: verifies product availability

**5. System**: calculates total cost based on policies

**6. System**: processes payment

**7. System**: confirms purchase to user

6. **Alternative flows**:

User not logged:

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

Product not available:

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

Payment failed:

>>System notifies the user of payment failure

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

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

**3. System**: fetches the lottery rules and product availability

**4. System**: registers the user for the lottery

**5. System**: sends confirmation of registration

6. **Alternative** **flows**:

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

### .4פעולות 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:** SessionToken, shopID, item

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 details, doesn’t belong to shop.
5. **System:** add item to shop.
6. **System:** send confirmation to user that the item was added.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

>> System notifies the user that don’t have permission to add item.

1. Item already belongs to shop.

>> System notifies the user that item already belongs to shop.

b.

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

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. **System:** send confirmation to user that the item was removed.

6. **Alternative Flows:**

1. Shop with shopID doesn’t exist.

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

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

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

1. Item doesn’t belong to the shop.

>> System notifies the user that can’t remove item that doesn’t belong to shop.

c.

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

1. **Actor**: shop owner

2. **Trigger:** shop owner requests to change item details

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 details, belongs to shop.
5. **User:** enter new item details.
6. **System:** change item details.
7. **System:** send confirmation to user that the item was changed.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

>> System notifies the user: don’t have permission to add item.

1. Item doen’t belongs to shop.

>> System notifies the user: can’t change item that doen’t belong to shop.

1. User enters non\_valid details.

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

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.
7. **System:** Sends confirmation to the user that the purchase/discount type was added.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

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

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

b.

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

1. **Actor**: Shop owner

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

3. **Precondition**:

1. Shop owner is logged in.
2. Shop exists.
3. 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. **System:** Sends confirmation to the user that the purchase/discount type was added.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. 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.

c.

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

1. **Actor**: Shop owner

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

3. **Precondition**:

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

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

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 policy doesn’t already belong to the shop.
5. **system:** creates purchase/discount policy with purchase/discountPolicyDetails.
6. **System:** adds purchase/discount policy to shop.
7. **System:** Sends confirmation to the user that the purchase/discount policy was added.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. purchase/discount policy already belongs to the shop.

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

d.

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

1. **Actor**: Shop owner

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

3. **Precondition**:

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

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

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 policy with purchase/discountPolicyId belongs to the shop.
5. **System:** remove purchase/discount policy from the shop.
6. **System:** Sends confirmation to the user that the purchase/discount policy was added.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. purchase/discountPolicy is not one of the purchase/discount policies of the store.

>> System notifies the user that the purchase/discount type 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:** verify user with NomineeId isn’t a shop owner of shopID.
6. **System:** sends a nomination message to the nominee .
7. **Nominee:** accepts the nomination message.
8. **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 shop doen’t exist.

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

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

1. User with NomineeId isn’t registered.

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

1. User with NomineeId is a shop owner of the shop.

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

1. User with NomineeId declined the nomination message .

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

4.4

Use-case: **Remove shop owner**

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 shop owner.
4. **System:** verify user with removeId is registered.
5. **System:** verify user with removeId is a shop owner of shopID.
6. **System:** verify user with removeId was promoted by user with UserID.
7. **System:** recursively removing all shop owners and managers promoted by removeId.
8. **System:** remove user with removeId as shop owner.
9. **System:** Sends confirmation to the shop owner that the user with removeId was removed as shop owner, and to all removed owners/managers that they have been removed from their role.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. User with removeId isn’t registered.

>> System notifies the user that a non-registered user cannot be a shop owner.

1. 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.

1. 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.

4.6

Use-case: **Add shop manager**

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

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

3. **Precondition**:

1. Shop owner is logged in.
2. Shop exists.
3. Nominee is a registered user.
4. 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.
9. **System:** Sends confirmation to the nominee and the shop owner that the nominee was added as shop manager.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. User with NomineeId isn’t registered.

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

1. 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.

1. User with NomineeId declined the nomination message .

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

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, newPermissions

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.
7. **System:** Sends confirmation to the shop manager and the shop owner that the manager permissions were defined.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

1. User with managerId isn’t a manager of the shop.

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

1. 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.

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.

1. User with founderId isn’t the shop founder.

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

1. 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 shop doen’t exist.

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

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

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 user with userId as a response.

6. **Alternative Flows:**

1. shop with shopID doesn’t exist.

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

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

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

4.13

Use-case: **Shop owner** **gets purchase history**

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.

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

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

### .5 פעולות של מבקר-מנוי בתפקידו כמנהל חנות:

5

a.

Use case: **View shop content**

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

b.

Use case: **Edit product inventory**

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

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

d.

Use case: **Edit discount policy**

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

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

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

### .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

# מבחני קבלה:

## מערכת:

|  |  |  |
| --- | --- | --- |
| **Test Name** | **Setup and Parameters** | **Expected Result** |
| **System initialization - Success** | 1. User u1 is created. 2. User u1 logs in. 3. User u1 is assigned as a system manager. 4. User u1 initializes the trading system. | Trading system is on. |
| **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. |
| **Update external service- Success** | 1.User u1 is created. 2.User u1 logs in. 3.User u1 is assigned as a system manager. 4.User u1 adds a PayPal payment service. | User u1 receives confirmation that PayPal payment service is added successfully. |
| **Update external service- Remove non-existing service** | 1.User u1 is created. 2.User u1 logs in. 3.User u1 is assigned as a system manager. 4.User u1 removes GooglePay. | User u1 receives notification that GooglePay payment service isn't removed. |
| **Update external service- add service that already exists** | 1.User u1 is created. 2.User u1 logs in. 3.User u1 is assigned as a system manager. 4.User u1 adds Fedex supply service. 5.User u1 tries to add Fedex supply service. | User u1 receives notification that Fedex supply service is already connected to the system. |
| **Payment/Supply- Success** | 1.User u1 is created. 2.User u1 logs in. 3.User u1 is assigned as a system manager. 4.User u1 adds a PayPal/Fedex payment/Supply service. 5.User u1 sets his payment/Delivery details. 6.User u1 proceeds order o1. 7.system sends info to payment/Delivery service. | User u1 receives confirmation from the system. |
| **Payment/Supply- invalid payment /Delivery details** | 1.User u1 is created. 2.User u1 logs in. 3.User u1 is assigned as a system manager. 4.User u1 adds a PayPal/Fedex payment/supply service. 5.User u1 sets invalid payment/delivery details. 6.User u1 proceeds order o1. 7.system sends info to payment/supply service. | User u1 receives notification from the system that the order cannot proceed due to invalid details. |
| **Real-time /delayed notification- Success** | 1.User u1 is created. 2.User u1 logs in/not logs in. | User u1 receive the real time/delayed message. |
| **Real-time /delayed notification-  user isn’t registered** | 1.system sends a notification to user u1 | notification does not get sent because user u1 doesn’t exist |

## משתמשים:

### 1. פעולות מבקר-אורח:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| **SuccessfulGuestLogsin** | 1. User logs in as guest 2. TempID1 is created and assign to the guest | 1. Guest is logged in into the system 2. TempID1 is assigned to the Guest 3. TempID1 has an empty shopping cart |
| **UserLogsinAsGuestBad** | 1. User 123 is created 2. User 123 logs in 3. User 123 attempts to logs in as guest | 1. User 123 Is still logs in as User 2. User 123 receives "Unauthorized Action" error 3. Guest’s TempID1 is deleted |
| **UserLogsinAsGuestGood** | 1. User 123 is created 2. User 123 logs in 3. User 123 logs out 4. User 123 attempts to logs in as guest | 1. User 123 Is logged in as guest 2. TempID1 is assigned to User 123 3. An empty shopping cart is created for TempID1 |

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| **SuccessfulGuest Exit** | 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 |
| **UserExitAsGuest** | 1. User 123 is created 2. User 123 logs in 3. User 123 attempts to exit as guest | 1. User 123 Is still logs in as User 2. User 123 receives "Unauthorized Action" error |

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| **SuccessfulGuestRegister** | 1. User logs in as guest 2. Guest chose register 3. Guest insert information correctly 4. User 123 is created and assigned to the active user | 1. User 123 is in the system 2. User 123 is logs in 3. The Guest cart is transferred to User 123 |
| **NotCompleteGuestRegister** | 1. User logs in as guest 2. Guest chose register 3. Guest insert not all the information correctly | 1. Guest receives "information not complete" error 2. User 123 is not created |
| **UserRegister** | 1. User 123 is created 2. User 123 logs in 3. User 123 attempts to register | 1. User 123 Is logs in 2. User 123 receives   "Unauthorized Action" error |

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| **SuccessfulUserLogsin** | 1. User enter the logs in page 2. User chose logs in as User 3. User enter information correctly | 1. User 123 is logged in |
| **NameWrongUserLogsin** | 1. User enter the logs in page 2. User chose logs in as User 3. User enter name not correctly | 1. User 123 Is not logged in 2. User receives "Logs in info wrong" error |
| **PasswordWrongUserLogsin** | 1. User enter the logs in page 2. User chose logs in as User 3. User enter password not correctly | 1. User 123 Is not logged in 2. User receives "Logs in info wrong" error |

### 2. פעולות קנייה של מבקר-אורח:

|  |  |  |
| --- | --- | --- |
| 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 |
| 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 |
| SystemErrorDuringSearch | 1. User is not in the system  2. User enters the system  3. User requests item search  4. Unexpected system error occurs | 1. System logs the error 2. System returns an error message to the user |
| 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 |
| AddItemsToCart | 1. User enters the system 2. User requests to add items to the cart with ShopID, ItemsID, SessionToken | 1. System searches for ShopID 2. System validates that the shop exists 3. System searches for ItemsID 4. System validates that items exist and are available 5. System adds items to the respective basket 6. System sends confirmation to the user |
| CartCheck | 1. User enters the system 2. User requests cart content with SessionToken, CartID | 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 |
| CartNotFound | 1. User enters the system 2. User requests cart content with a non-existent CartID | 1. System searches for CartID 2. System returns "Cart not found" message |
| UnauthorizedCartAccess | 1. User enters the system 2. User requests cart content that does not belong to them | 1. System verifies cart ownership 2. System returns "Unauthorized access" message |
| RemoveItemsFromCart | 1. User enters the system 2. User requests to remove items from cart with SessionToken, CartID, ItemsToDeleteID | 1. System searches for CartID 2. System validates cart existence 3. System verifies cart ownership 4. System searches for ItemsToDeleteID 5. System validates that items exist in the cart 6. System asks for user confirmation 7. User approves deletion 8. System removes items from cart 9. System sends confirmation to user |
| BuyCartContent | 1. User enters the system 2. User requests to buy cart content with SessionToken, CartID | 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 asks for user confirmation 7. User approves purchase 8. System charges the user 9. 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 |

### פעולות קנייה של מבקר-מנוי בשוק:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Test Name | Setup & Parameters | Expected Results | | SuccessfulLogout | 1. User 123 is created 2. User 123 logs in 3. User 123 adds items to CartID 4. User 123 requests logout | 1. System verifies user login 2. System verifies CartID ownership 3. System saves cart data 4. System changes user status to Guest 5. User 123 receives confirmation | | LogoutNotLoggedIn | 1. User 123 is not logged in 2. User 123 requests logout | 1. System returns "User not logged in" error | | CartOwnershipMismatch | 1. User 123 is created 2. User 123 logs in 3. User 456 adds items to CartID 4. User 123 requests logout | 1. System returns "Cart does not belong to user" error | | CartNotSaved | 1. User 123 is created 2. User 123 logs in 3. User 123 adds items to CartID 4. System encounters an error while saving cart 5. User 123 requests logout | 1. System returns "Cart data could not be saved" error | |
| |  |  |  | | --- | --- | --- | | Test Name | Setup & Parameters | Expected Results | | SuccessfulShopCreation | 1. User 123 is created 2. User 123 logs in 3. User 123 requests to open a shop "ABC" | 1. System verifies user login 2. System checks user permissions 3. System confirms unique shop name 4. System creates shop "ABC" 5. System assigns ownership to User 123 6. User 123 receives confirmation | | ShopCreationNotLoggedIn | 1. User 123 is not logged in 2. User 123 requests to open a shop "ABC" | 1. System returns "User not logged in" error | | NoPermissionToOpenShop | 1. User 123 is created 2. User 123 logs in 3. User 123 lacks permission 4. User 123 requests to open a shop "ABC" | 1. System returns "User lacks permission" error | | DuplicateShopName | 1. User 123 is created 2. User 123 logs in 3. Shop "ABC" already exists 4. User 123 requests to open shop "ABC" | 1. System returns "Shop name already exists" error | | ShopCreationFailed | 1. User 123 is created 2. User 123 logs in 3. User 123 requests to open a shop "ABC" 4. System encounters an error | 1. System returns "Shop creation failed" error | |
|  |
| |  |  |  | | --- | --- | --- | | Test Name | Setup & Parameters | Expected Results | | SuccessfulShopRating | 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 | | RateShopNotLoggedIn | 1. User 123 is not logged in  2. User 123 submits a rating | 1. System returns "User not logged in" error | | NoPurchaseHistoryForShop | 1. User 123 is created  2. User 123 logs in  3. User 123 submits a rating for ShopID without a purchase | 1. System returns "Purchase required before rating" error | | RatingNotSaved | 1. User 123 is created  2. User 123 logs in  3. User 123 submits a rating for ShopID  4. System encounters an error | 1. System returns "Rating not saved" error | |
| |  |  |  | | --- | --- | --- | | Test Name | Setup & Parameters | Expected Results | | SuccessfulMessageSend | 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 | | MessageSendNotLoggedIn | 1. User 123 is not logged in 2. User 123 sends a message | 1. System returns "User not logged in" error | | ShopNotFound | 1. User 123 is created  2. User 123 logs in  3. ShopID does not exist 4. User 123 sends a message | 1. System returns "Shop not found" error | | MessageNotForwarded | 1. User 123 is created  2. User 123 logs in  3. ShopID exists  4. User 123 sends a message  5. System encounters an error | 1. System returns "Message was not forwarded" error | |
| **View Personal Purchase History** |
| |  |  |  | | --- | --- | --- | | Test Name | Setup & Parameters | Expected Results | | SuccessfulPurchaseHistoryView | 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 | | ViewHistoryNotLoggedIn | 1. User 123 is not logged in  2. User 123 requests purchase history | 1. System returns "User not logged in" error | | HistoryRetrievalFailed | 1. User 123 is created  2. User 123 logs in  3. System encounters an error while retrieving history | 1. System returns "Request failed" error | |

### פעולות של מבקר-מנוי בתפקידו כבעל חנות:

|  |  |  |
| --- | --- | --- |
| **Add 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. 5. User u1 requests to add Item i1 to Shop s1. | User u1 receives confirmation that Item i1 was added to Shop s1. |
| **Add Item - Shop Does Not Exist** | 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. |
| **Add 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. 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. |
| **Add Item - duplicate item** | 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. |
| **Remove 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 remove Item i1 from Shop s1. | User u1 receives confirmation that Item i1 was removed from Shop s1. |
| **Remove 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 remove item i1 from shop s999 | User u1 is notified that Shop s999 does not exist. |
| **Remove 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 remove Item i1 to Shop s1. | User u2 is notified that they do not have permission to remove items to Shop s1. |
| **Remove Item - Item Does Not Belong to Shop** | 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 remove Item i1 from Shop s1. | User u1 is notified that Item i1 does not belong to Shop s1. |
| **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. |
| **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 |
| **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 |
| **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 |
| **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 |
| **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. |
| **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 |
| **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. |

### פעולות של מבקר-מנוי בתפקידו כמנהל חנות:

|  |  |  |
| --- | --- | --- |
| 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 |

|  |  |  |
| --- | --- | --- |
| 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  4. User 123 attempts to edit product | 1. System returns "Permission denied" error |

|  |  |  |
| --- | --- | --- |
| 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  4. User 123 submits an invalid policy | 1. System returns "Invalid policy format" error |

|  |  |  |
| --- | --- | --- |
| 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 |

|  |  |  |
| --- | --- | --- |
| 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 |

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| SuccessfulAddDiscountPolicy | 1. User 123 is created  2. User 123 logs in  3. User 123 has permission to add discount policies  4. User 123 submits NewDiscountPolicy | 1. System verifies session and permissions 2. System validates policy 3. System adds discount policy 4. 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 |

### פעולות של מבקר-מנוי בתפקידו כמנהל מערכת המסחר :

|  |  |  |
| --- | --- | --- |
| 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 |