**Table of Contents- Use Cases:**

[Guest entry into the system: (1.1)……… 3](#_Toc196123472)

[Guest exits from the system: (1.2)……… 3](#_Toc196123473)

[Guest registration: (1.3)……………………. 3](#_Toc196123474)

[User login: (1.4)………………………………….. 4](#_Toc196123475)

[Guest- get information about stores and products: (2.1) 4](#_Toc196123476)

[Guest- search for products across all stores: (2.2a) 4](#_Toc196123477)

[Guest- search for products inside a specific store: (2.2b) 5](#_Toc196123478)

[Guest- add product to storeBag: (2.3) 5](#_Toc196123479)

[Guest- view and edit shoppingCart contents: (2.4) 6](#_Toc196123480)

[Guest- purchase cart (2.5)………………… 6](#_Toc196123481)

[Subscriber- get information about stores and products (3) 7](#_Toc196123482)

[Subscriber- search for products across all stores: (3) 7](#_Toc196123483)

[Subscriber- search for products inside a specific store: (3) 7](#_Toc196123484)

[Subscriber- add product to storeBag: (3) 8](#_Toc196123485)

[Subscriber- View and edit shoppingCart contents: (3) 8](#_Toc196123486)

[Subscriber- purchase cart (3)………….. 9](#_Toc196123487)

[Subscriber exits from the system: (3). 9](#_Toc196123488)

[Subscriber- Logout (3.1)…………………… 10](#_Toc196123489)

[Subscriber- Open a store (3.2)………….. 10](#_Toc196123490)

[Subscriber- Rate product and store (3.4) 10](#_Toc196123491)

[Subscriber- Send a message to a store (3.5) 11](#_Toc196123492)

[Subscriber- View personal purchase history (3.7) 11](#_Toc196123493)

[Subscriber- Submit bid for a product (3.9) 12](#_Toc196123494)

[Subscriber- Purchase a product after bid approval (3.9) 13](#_Toc196123495)

[Subscriber- Purchase a product via auction (3.10) 13](#_Toc196123496)

[Subscriber- Participate in a raffle for a product (3.11) 14](#_Toc196123497)

[Subscriber- Win a product raffle and complete purchase (3.11) 15](#_Toc196123498)

[Owner- Add new product to store (4.1) 16](#_Toc196123499)

[Owner- Remove a product from the store (4.1) 16](#_Toc196123500)

[Owner- Edit product from the store (4.1) 17](#_Toc196123501)

[Owner- Edit store purchase policy (4.2) 17](#_Toc196123502)

[Owner- Edit store discount policy (4.2) 18](#_Toc196123503)

[Owner- Appoint additional store owner (4.3) 19](#_Toc196123504)

[Owner- Remove store owner (4.4)……. 19](#_Toc196123505)

[Owner- Appoint store manager (4.6)… 20](#_Toc196123506)

[Owner- Edit store manager permissions (4.7) 21](#_Toc196123507)

[Founder- Close store (4.9)……………….. 21](#_Toc196123508)

[Owner- Request information about store roles (4.11) 22](#_Toc196123509)

[Owner- Respond to user messages (4.12) 22](#_Toc196123510)

[Owner- View store purchase history (4.13) 23](#_Toc196123511)

[System Administrator- Close store by system admin (6.1) 23](#_Toc196123512)

System Administrator – Suspend User (6.2)……………………………………………23

System Administrator – Revoke User Suspension (6.3)………………………………..24

System Administrator – View User Suspensions (6.4)………………………………………24

**UseCases**

# **Guest entry into the system: (1.1)**

1. **Actor:** Guest (unregistered user).
2. **Trigger:** User opens the store system.
3. **Precondition:**
4. User is not logged in.
5. **Input Parameters:** None
6. **Main Scenario:**
7. System identifies the user as a guest.
8. System creates a temporary session for the guest to manage actions during the visit.
9. A new empty shopping cart is initialized in memory for the session.
10. **Alternative Flows:** None

# **Guest exits from the system: (1.2)**

1. **Actor:** Guest (unregistered user).
2. **Trigger:** Guest closes browser.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** Session ID
6. **Main Scenario:**
7. System deletes the in-memory cart associated with that session.
8. System ends the guest’s session.
9. **Alternative Flows:**
10. Unexpected disconnect:

Session times out, cart is also deleted.

# **Guest registration: (1.3)**

1. **Actor:** Guest (unregistered user).
2. **Trigger:** Guest requests to register.
3. **Precondition:**
4. Guest is not a registered user.
5. **Input Parameters:** Username, password, emailAddress, shippingAddress
6. **Main Scenario:**
7. User fills the registration details- username, password, emailAddress, shippingAddress.
8. System checks that the username and emailAddress doesn't exist.
9. If valid, creates a new user record.
10. A new shoppingCart will be created and saved into the user’s record
11. Guest is now a registered user (subscriber), but is still not authenticated (no token is issued yet)- because still didn’t log in.
12. **Alternative Flows:**
13. Username or emailAddress already exists:

System shows error.

# **User login: (1.4)**

1. **Actor:** Registered user.
2. **Trigger:** Guest requests to login.
3. **Precondition:**
4. User is registered in the system.
5. User is not logged in.
6. **Input Parameters:** Username, password
7. **Main Scenario:**
8. User fills the login details- username, password.
9. System checks that the username exists.
10. If exists, checks that the password is correct.
11. The system defines the user as a Subscriber.
12. The system generates a token that identifies the user and saves it in the system (this token is used to recognize the user in future actions).
13. **Alternative Flows:**
14. Username doesn't exist:

System shows error.

1. Wrong password:

Access denied.

# **Guest- get information about stores and products: (2.1)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests for information of stores and products.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** None
6. **Main Scenario:**
7. System displays active stores and their available products.
8. **Alternative Flows:** None

# **Guest- search for products across all stores: (2.2a)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests to search products across all stores.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** productName, category, keyNames, filters (priceRange, productRating, storeRating)
6. **Main Scenario:**
7. Guest enters search keywords or selects category.
8. Guest applies optional filters (priceRange, productRating, storeRating).
9. System performs search across all active stores.
10. System returns a list of matching products and their store info.
11. **Alternative Flows:**
12. No matches found:

System shows "no results" message.

# **Guest- search for products inside a specific store: (2.2b)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests to search products inside a specific store.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** storeName, productName, category, keyNames, filters (priceRange, productRating)
6. **Main Scenario:**
7. Guest searches for a specific store.
8. System checks that the store is valid.
9. If valid, guest enters search keywords or selects category.
10. Guest applies optional filters (priceRange, productRating).
11. System performs search across the requested store.
12. System returns a list of matching products.
13. **Alternative Flows:**
14. StoreName is not valid:

system blocks search and displays message

1. No matches found:

System shows "no results" message.

# **Guest- add product to storeBag: (2.3)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests to add a product to a storeBag.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** productID, quantity
6. **Main Scenario:**
7. Guest selects product and quantity.
8. Product is added to the storeBag.
9. **Alternative Flows:** None

# **Guest- view and edit shoppingCart contents: (2.4)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests to view and edit shoppingCart contents.
3. **Precondition:**
4. Guest session is active.
5. **Input Parameters:** for view: none, for edit: productID, newQuantity.
6. **Main Scenario:**
7. System displays all products currently in the cart.
8. Guest chooses to change quantity or to remove a specific product from the cart.
9. System updates the shoppingCart accordingly.
10. **Alternative Flows:**
11. Guest tries to set quantity to zero:

system removes the item.

# **Guest- purchase cart (2.5)**

1. **Actor:** Guest.
2. **Trigger:** Guest requests to purchase the shoppingCart.
3. **Precondition:**
4. Guest session is active.
5. Shopping cart is not empty.
6. **Input Parameters:** deliveryAddress, paymentInfo: creditCardNumber, expDate, cvv.
7. **Main Scenario:**
8. System validates stock.
9. If valid, the system updates the prices according to the discountPolicy.
10. System makes the purchase according to the purchasePolicy.
11. User fills the deliveryAddress.
12. User fills the paymentInfo: creditCardNumber, expDate, cvv.
13. System contacts payment service.
14. System contacts delivery service.
15. Upon success:

* system completes purchase.
* System clears shoppingCart.
* System updates product stock in the relevant stores.
* System stores this purchase in the purchase history of the relevant stores.

1. **Alternative Flows:**
2. Payment fails:

system blocks the action.

1. Product unavailable:

system blocks the action.

1. Delivery service fails:

system blocks the action.

1. Purchase does not comply with purchase policy:

System blocks the action and shows explanation.

# **Subscriber- get information about stores and products (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests for information of stores and products.
3. **Precondition:**
4. User is logged in.
5. User is authenticated and has a valid token.
6. **Input Parameters:** None
7. **Main Scenario:**
8. System displays active stores and their available products.
9. **Alternative Flows:** None

# **Subscriber- search for products across all stores: (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to search products across all stores.
3. **Precondition:**
4. User is logged in.
5. User is authenticated and has a valid token.
6. **Input Parameters:** productName, category, keyNames, filters (priceRange, productRating, storeRating)
7. **Main Scenario:**
8. Subscriber enters search keywords or selects category.
9. Subscriber applies optional filters (priceRange, productRating, storeRating).
10. System performs search across all active stores.
11. System returns a list of matching products and their store info.
12. **Alternative Flows:**
13. No matches found:

System shows "no results" message.

# **Subscriber- search for products inside a specific store: (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to search products inside a specific store.
3. **Precondition:**
4. User is logged in.
5. User is authenticated and has a valid token.
6. **Input Parameters:** storeName, productName, category, keyNames, filters (priceRange, productRating)
7. **Main Scenario:**
8. Subscriber searches for a specific store.
9. System checks that the store is valid.
10. If valid, Subscriber enters search keywords or selects category.
11. Subscriber applies optional filters (priceRange, productRating).
12. System performs search across the requested store.
13. System returns a list of matching products.
14. **Alternative Flows:**
15. StoreName is not valid:

system blocks search and displays message

1. No matches found:

System shows "no results" message.

# **Subscriber- add product to storeBag: (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to add a product to a storeBag.
3. **Precondition:**
4. User is logged in.
5. User is authenticated and has a valid token.
6. **Input Parameters:** productID, quantity
7. **Main Scenario:**
8. Subscriber selects product and quantity.
9. Product is added to the storeBag.
10. System saves the shoppingCart to cookies.
11. **Alternative Flows:** None

# **Subscriber- View and edit shoppingCart contents: (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriberrequests to view and edit shoppingCart contents.
3. **Precondition:**
4. User is authenticated and has a valid token.
5. **Input Parameters:** for view: none, for edit: productID, newQuantity.
6. **Main Scenario:**
7. System loads the subscriber’s cart.
8. System displays all products currently in the cart.
9. Subscriber chooses to change quantity or to remove a specific product from the cart.
10. System updates the shoppingCart accordingly and saves to cookies if necessary.
11. **Alternative Flows:**
12. Subscriber tries to set quantity to zero:

system removes the item.

# **Subscriber- purchase cart (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to purchase the shoppingCart.
3. **Precondition:**
4. Shopping cart is not empty.
5. User is authenticated and has a valid token.
6. **Input Parameters:** deliveryAddress, paymentInfo: creditCardNumber, expDate, cvv.
7. **Main Scenario:**
8. System validates stock.
9. If valid, the system updates the prices according to the discountPolicy.
10. System makes the purchase according to the purchasePolicy.
11. Subscriber fills the deliveryAddress.
12. Subscriber fills the paymentInfo: creditCardNumber, expDate, cvv.
13. System contacts payment service.
14. System contacts delivery service.
15. Upon success:

* system completes purchase.
* System clears shoppingCart.
* System updates product stock in the relevant stores.
* System stores the purchase in the subscriber’s personal purchase history.
* System stores the purchase in the purchase history of the relevant stores.

1. **Alternative Flows:**
2. Payment fails:

system blocks the action.

1. Product unavailable:

system blocks the action.

1. Delivery service fails:

system blocks the action.

1. Purchase does not comply with purchase policy:

System blocks the action and shows explanation.

# **Subscriber exits from the system: (3)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber closes browser.
3. **Precondition:**
4. User is authenticated and has a valid token.
5. **Input Parameters:** Authentication token
6. **Main Scenario:**
7. System retains the subscriber’s cart, associating it with their username.
8. **Alternative Flows:**
9. Unexpected disconnect:

Token expires, but cart is preserved.

# **Subscriber- Logout (3.1)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to logout.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. **Input Parameters:** None
7. **Main Scenario:**
8. System preserves the subscriber’s shopping cart (associated with the user).
9. System transitions the user into guest mode.
10. System creates a new empty shopping cart for the guest.
11. **Alternative Flows:** None

# **Subscriber- Open a store (3.2)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to open a new store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Subscriber does not currently own a store with the same name.
7. **Input Parameters:** storeName, storeType.
8. **Main Scenario:**
9. Subscriber enters the required store information (name and type).
10. System checks that the store name is available.
11. System creates a new store.
12. System assigns the subscriber as the store foundsers.
13. System makes the store visible and active.
14. **Alternative Flows:**
15. Store name already exists:

System displays error message and blocks creation.

1. Required fields are missing:

System prompts user to complete missing data.

# **Subscriber- Rate product and store (3.4)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to rate a product from their purchase history.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Subscriber has successfully purchased the product from the store.
7. **Input Parameters:** productID**,** productRate.
8. **Main Scenario:**
9. Subscriber selects a product that he purchased.
10. System verifies that the purchase of this product exists for the user.
11. Subscriber enters product rate.
12. System saves the rating and associates it with the product/store.
13. **Alternative Flows:**
14. No matching purchase found:

System blocks the action and shows error.

1. Rating value is invalid (not in range 0-5):

System prompts user to enter a valid rating.

# **Subscriber- Send a message to a store (3.5)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to send a message/ question to a specific store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Target store is active.
7. **Input Parameters:** storeID, message.
8. **Main Scenario:**
9. System verifies the store is active.
10. Subscriber writes and submits the message.
11. System stores the message and notifies the store owner.
12. **Alternative Flows:**
13. Store is inactive:

System blocks the action and shows error.

1. Message is empty:

System prompts user to enter a valid message.

# **Subscriber- View personal purchase history (3.7)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to view their purchase history.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. **Input Parameters:** None
7. **Main Scenario:**
8. System retrieves all past purchases of the subscriber across all stores.
9. For each purchase the system displays: Purchase date, product name and quantity, store name, price paid at the time of purchase, delivery status.
10. **Alternative Flows:**
11. Subscriber has no previous purchases:

System displays an empty history with the relevant message.

# **Subscriber- Submit bid for a product (3.9)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to submit a bid on a product.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. The product is available in stock.
7. The product/ store allows bidding according to its purchase policy.
8. **Input Parameters:** storeID,productID, bidAmount.
9. **Main Scenario:**
10. System verifies the store is active.
11. System verifies the product is in stock.
12. System verifies that the store’s purchase policy allows bidding for this product.
13. Subscriber enters a bid amount.
14. System validates the bid amount (positive number and not 0).
15. System sends notifications to all store owners and authorized store managers.
16. System stores the bid.
17. Subscriber receives confirmation that the bid was submitted.
18. **Alternative Flows:**
19. Store is inactive:

System blocks the action and shows error.

1. Product is out of stock:

System blocks the bid and shows a message.

1. Bidding is not allowed on this product:

System disables the bid option or shows an error.

1. Bid amount is invalid (negative or 0):

System shows validation error and blocks submition.

# **Subscriber- Purchase a product after bid approval (3.9)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber receives bid approval from all store owners and managers and chooses to complete the purchase.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Subscriber has previously approved a bid for the product.
7. Product is still available in stock.
8. Store purchase policy allows completing the transaction.
9. All store owners and managers approved the bid.
10. **Input Parameters:** storeID, productID, bidAmount, delieveryAddress, paymentInfo (creditCardNumber, expDate, cvv).
11. **Main Scenario:**
12. System verifies the product is still available.
13. Subscriber enters delivery and payment details.
14. System processes payment for the approved bid amount.
15. Subscriber contacts delivery service.
16. On success:

* System confirms the purchase.
* System updates the stock.
* System stores the purchase in the subscriber’s personal purchase history.
* System stores this purchase in the purchase history of the relevant stores.

1. **Alternative Flows:**
2. Product is no longer available:

System cancels the purchase and notifies the user.

1. Payment fails:

System blocks the action and prompts retry.

1. Deliver service failure:

System blocks the purchase and notifies the user.

# **Subscriber- Purchase a product via auction (3.10)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber requests to place an offer on a product sold via auction.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Product is still available in stock.
7. Product is currently offered for sale in an active auction.
8. Store purchase policy allows auction based purchases.
9. **Input Parameters:** storeID,productID, offerAmount, paymentDetails, deliveryAddress.
10. **Main Scenario:**
11. System displays starting price and auction duration.
12. Subscriber submits an offer that is higher than the starting price and all previous offers.
13. System validates the offer.
14. System saves the offer and updates the highest offer.
15. When the auction ends, system checks for valid auctions.
16. The subscriber who submitted the highest offer is declared the winner.
17. System notifies the winning subscriber.
18. Subscriber confirms the purchase.
19. Subscriber provides delivery and payment information.
20. System processes the payment and initiates delivery.
21. On success:

* System confirms the purchase.
* System updates the stock.
* System stores the purchase in the subscriber’s personal purchase history.
* System stores this purchase in the purchase history of the relevant stores.

1. **Alternative Flows:**
2. Offer is too low:

System blocks participation.

1. Auction already ended:

Offer not accepted.

1. Payment or delivery fails:

System cancels the transaction and notifies the subscriber.

# **Subscriber- Participate in a raffle for a product (3.11)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber chooses to participate in a raffle for a product.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Product is still available in stock.
7. Raffle for the product is active (time window is still open)
8. Store purchase policy allows raffle participation for this product.
9. **Input Parameters:** storeID,productID, paymentDetails.
10. **Main Scenario:**
11. System displays raffle details (end date and price).
12. System checks that the raffle is still active and that the product is in stock.
13. User pays to enter the raffle.
14. System validates that all payments from all users doesn’t pass the product price.
15. If the total payments reach the product price, the systems closes the raffle, and the raffle is carried out and each participant's chances of winning are according to the amount he paid when entering the lottery.
16. **Alternative Flows:**
17. Raffle is already closed:

System blocks participation.

1. Product is no longer available:

System cancels the entry to the raffle.

# **Subscriber- Win a product raffle and complete purchase (3.11)**

1. **Actor:** Subscriber.
2. **Trigger:** Subscriber is selected as the winner of a product raffle.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. Subscriber previously participated in a valid raffle.
7. Raffle has ended successfully.
8. Subscriber was randomly selected as the winner according to the amount he paid when entering the lottery.
9. Product is still available in stock.
10. **Input Parameters:** storeID,productID, deliveryAddress.
11. **Main Scenario:**
12. Subscriber is notified of their win.
13. Subscriber confirms intent to complete the purchase.
14. System validates product availability.
15. Subscriber enters delivery address.
16. System contacts delivery service.
17. On success:

* Purchase is finalized.
* Product is marked as sold.
* System updates the stock.
* System stores the purchase in the subscriber’s personal purchase history.
* System stores this purchase in the purchase history of the relevant stores.

1. **Alternative Flows:**
2. Product is no longer available:

System cancels the purchase and notifies the user.

1. Delivery fails:

System blocks the purchase and notifies the user.

# **Owner- Add new product to store (4.1)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to add a product to his store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or a manager with the appropriate permissions.
7. Store is active.
8. **Input Parameters:** storeID, name, category, price, quantity.
9. **Main Scenario:**
10. System validates that the store is active and the user is his owner or his manager with permission.
11. User fills in product details and initial quantity.
12. System validates input (non-negative price and quantity, store is active and he is the owner).
13. System creates a new product and adds it to the store.
14. System confirms successful addition.
15. **Alternative Flows:**
16. Invalid input (negative price/quantity):

System shows error.

1. Store is inactive:

System blocks the action.

1. Product with the identical name is already in store:

System blocks the action.

# **Owner- Remove a product from the store (4.1)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to remove a product from his store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or a manager with the appropriate permissions.
7. Store is active.
8. The product exists in the store.
9. **Input Parameters:** storeID, name.
10. **Main Scenario:**
11. System validates that the store is active and the user is his owner or his manager with permission.
12. System verifies that the product exists in the store.
13. System removes the product from the store.
14. System confirms successful remove.
15. **Alternative Flows:**
16. Product does not exists:

System shows error.

1. Store is inactive:

System blocks the action.

# **Owner- Edit product from the store (4.1)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to edit a product’s details.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or a manager with the appropriate permissions.
7. Store is active.
8. The product exists in the store.
9. **Input Parameters:** storeID, name, fieldName, fieldUpdate (can be price and quantity).
10. **Main Scenario:**
11. System validates that the store is active and the user is his owner or his manager with permission.
12. System verifies that the product exists in the store.
13. System validates the new values.
14. System updates the product with the new information.
15. System confirms successful update.
16. **Alternative Flows:**
17. Product does not exists:

System shows error.

1. Store is inactive:

System blocks the action.

1. Invalid input (negative price or quantity):

System blocks action.

# **Owner- Edit store purchase policy (4.2)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to edit the store’s purchase policy.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or a manager with the appropriate permissions.
7. Store is active.
8. **Input Parameters:** storeID, productName/userName, newDetails (allowedCustomers, purchaseLanes, activeRules).
9. **Main Scenario:**
10. System validates that the store is active and the user is his owner or his manager with permission.
11. System verifies that the product exists in the store/username exists.
12. System validates the new details.
13. System applies the new purchase policy to the selected product/user.
14. System confirms successful update.
15. **Alternative Flows:**
16. Store is inactive:

System blocks the action.

1. Invalid type:

System blocks action.

# **Owner- Edit store discount policy (4.2)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to edit the store’s discount policy.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or a manager with the appropriate permissions.
7. Store is active.
8. **Input Parameters:** storeID, productName, discountDetails (type, rules, appliedUsers).
9. **Main Scenario:**
10. System validates that the store is active and the user is his owner or his manager with permission.
11. System verifies that the product exists in the store.
12. System validates the new discount details.
13. System applies the new discount policy for the product/user.
14. System confirms successful update.
15. **Alternative Flows:**
16. Store is inactive:

System blocks the action.

1. Invalid type:

System blocks action.

1. Invalid discount details :

System blocks the update.

# **Owner- Appoint additional store owner (4.3)**

1. **Actor:** Store owner (the one appointing the new owner).
2. **Trigger:** Store owner requests to appoint a new owner to the store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store.
7. Store is active.
8. The target user is a registered subscriber and not currently an owner of the same store.
9. **Input Parameters:** storeID, username.
10. **Main Scenario:**
11. System validates that the user is his owner.
12. System verifies that the target user is not already an owner of this store.
13. System creates a pending appointment request to the target user.
14. The selected subscriber receives a notification and must approve or decline.
15. If approval, the subscriber is added as a store owner and add the subscriber to the appointedOwnerList of the owner that appointed him.
16. **Alternative Flows:**
17. Target user already owns the store:

The system blocks the appointment.

1. Approval is declined:

Appointment request canceled.

# **Owner- Remove store owner (4.4)**

1. **Actor:** Store owner (who originally appointed the target owner).
2. **Trigger:** Store owner requests to remove a store owner they add previously appointed.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store.
7. Store is active.
8. The target user is an owner of the same store.
9. Target user was appointed by the acting user.
10. **Input Parameters:** storeID, username.
11. **Main Scenario:**
12. System validates that the acting user is his owner.
13. System verifies that the target user was appointed by the acting user.
14. System removes the target user from the store’s ownersList.
15. System recursively updates all users (owners and managers) that were appointed by the removed owner to be regular subscribers (remove them from the appointedOwnerList and appointedManagerList).
16. System sends notifications to all affected users.
17. **Alternative Flows:**
18. Target user was not appointed by the acting user:

The system blocks the action.

# **Owner- Appoint store manager (4.6)**

1. **Actor:** Store owner (the one appointing the new manager).
2. **Trigger:** Store owner requests to appoint a new manager to the store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store.
7. Store is active.
8. The target user is a registered subscriber who is not currently a manager or owner of the store.
9. **Input Parameters:** storeID, username, listOfPermissions.
10. **Main Scenario:**
11. System validates that the acting user is his owner.
12. System validates that the target user is a subscriber and not already a manager/owner.
13. System creates a pending appointment request.
14. The target user receives a notification and must approve or decline the appointment. If approval, the user is added as a manager and the selected permissions are assigned.
15. System will add the target user to the appointedManagerList.
16. **Alternative Flows:**
17. Target user is already a manager or owner:

The system blocks the action.

1. Target user declines the appointment:

Request canceled.

# **Owner- Edit store manager permissions (4.7)**

1. **Actor:** Store owner (who originally appointed the manager).
2. **Trigger:** Store owner requests to update the permissions of a store manager that he appointed.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store.
7. Store is active.
8. The target user is a store manager who was appointed by the acting user.
9. **Input Parameters:** storeID, username, listOfPermissions.
10. **Main Scenario:**
11. System validates that the acting user is his owner.
12. System validates that the target user is a manager appointed by the acting user.
13. Owner selects the new set of permissions.
14. System updates the manager’s permissions accordingly.
15. System confirms the update and notifies the target manager.
16. **Alternative Flows:**
17. Target user is not a manager:

The system blocks the action.

1. Target manager was not appointed by the acting user:

The system blocks the action.

# **Founder- Close store (4.9)**

1. **Actor:** Store founder (who originally created the store).
2. **Trigger:** Store founder requests to close the store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the original founder of the target store.
7. Store is active.
8. **Input Parameters:** storeID.
9. **Main Scenario:**
10. System validates that the store is active and the acting user is his founder.
11. System updates the store status to inactive.
12. System sends notifications to all store owners and managers about the closure.
13. Appointments are preserved but no management actions can be taken while the store is inactive.
14. **Alternative Flows:**
15. Store is already inactive:

System blocks the action.

1. Acting user is not the founder of the store:

The system blocks the action.

# **Owner- Request information about store roles (4.11)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to view store roles.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or the manager with permissions.
7. Store is active.
8. **Input Parameters:** storeID.
9. **Main Scenario:**
10. System validates that the store is active and the acting user is his owner or his manager with permissions.
11. System retrieves and displayes a list of: all store owners, all store managers and their current permissions.
12. **Alternative Flows:**
13. Store is inactive:

System blocks the action.

1. User is not an owner of the store:

The system blocks the action.

# **Owner- Respond to user messages (4.12)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to respond to a user’s message.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or the manager with permissions.
7. Store is active.
8. A user has previously submitted a message to this store.
9. **Input Parameters:** storeID, messageID, responseText.
10. **Main Scenario:**
11. System validates that the store is active and the acting user is his owner or his manager with permissions.
12. Owner writes a response.
13. System stores the response and links it to the original message.
14. System sends a notification to the user who submitted the question.
15. **Alternative Flows:**
16. Store is inactive:

System blocks the action.

1. User is not an owner of the store:

The system blocks the action.

1. Message was already responded:

The system blocks the action.

1. Empty response submitted:

The system blocks the action.

# **Owner- View store purchase history (4.13)**

1. **Actor:** Store owner or store manager (with permission to this action).
2. **Trigger:** Store owner/manager requests to view purchase history of the store.
3. **Precondition:**
4. User is logged in as subscriber.
5. User is authenticated and has a valid token.
6. User is the owner of the target store or the manager with permissions.
7. Store is active.
8. **Input Parameters:** storeID.
9. **Main Scenario:**
10. System validates that the store is active and the acting user is his owner or his manager with permissions.
11. System retrieves and displays a list of all purchases made from the store.
12. For each purchase the system shows: date, product name, quantity, price, buyer username, delivery status.
13. **Alternative Flows:**
14. Store is inactive:

System blocks the action.

1. No purchases exist yet:

System shows an appropriate message.

# **System Administrator- Close store by system admin (6.1)**

1. **Actor:** System administrator.
2. **Trigger:** System administrator requests to close a store.
3. **Precondition:**
4. User is logged in as system administrator.
5. User is authenticated and has a valid token.
6. Store is exist and active.
7. **Input Parameters:** storeID.
8. **Main Scenario:**
9. System validates that the store is exist and active and the acting user is the system administrator.
10. System updates the store status to inactive.
11. System removes all ownership and management roles associated with the store- all of them become regular subscribers.
12. System sends notifications to all affected users (owners and managers) about the closure and the role termination.
13. **Alternative Flows:**
14. Store is inactive:

System blocks the action.

# **System Administrator – Suspend User (6.2)**

**1.Actor:** System administrator.

**2.Trigger:** System administrator requests to suspend a user.

**3.Precondition:**

1. User is logged in as system administrator.
2. User is authenticated and has a valid token.
3. Target user exists in the system.

**4.Input Parameters:** targetUsername, suspensionDuration (optional; null for permanent).

**5.Main Scenario:**

1. System validates the existence of the target user and the administrator’s identity.
2. System applies suspension to the user account for the specified duration or permanently.
3. System marks the user as suspended and limits their actions to viewing-only mode.
4. System stores the suspension record with date, duration, and end date (if applicable).

**6.Alternative Flows:**

1. Target user does not exist:  
   System blocks the action.
2. Target user is already suspended:  
   System updates the existing suspension if duration differs.
3. Suspension duration is invalid (e.g., negative):  
   System blocks the action.

# **System Administrator – Revoke User Suspension (6.3)**

1. **Actor:** System administrator.
2. **Trigger:** System administrator requests to revoke a user suspension.
3. **Precondition:**
   1. User is logged in as system administrator.
   2. User is authenticated and has a valid token.
   3. Target user exists and is currently suspended.
4. **Input Parameters:** targetUsername.
5. **Main Scenario:**
   1. System verifies the target user is suspended and the requester is a valid administrator.
   2. System lifts the suspension from the user account.
   3. System restores the user's full permissions.
   4. System logs the revocation event.
6. **Alternative Flows:**
   1. Target user is not suspended:  
      System blocks the action.
   2. Target user does not exist:  
      System blocks the action.

**System Administrator – View User Suspensions (6.4)**

1. **Actor:** System administrator.
2. **Trigger:** System administrator requests to view all user suspensions.
3. **Precondition:**
   1. User is logged in as system administrator.
   2. User is authenticated and has a valid token.
4. **Input Parameters:**
5. **Main Scenario:**
   1. System validates the administrator’s credentials.
   2. System retrieves all suspension records
   3. System displays each record with: targetUsername, startDate, duration, and endDate.
   4. System marks expired suspensions accordingly.
6. **Alternative Flows:**
   1. No suspensions found (with or without filters):  
      System shows an appropriate message.