System use cases:

Use Case I.1 – Market activation:

A user activates the market system for the first time, and becomes the first system manager.

Actors: user (and he will be also the system manager).

Precondition: system must be off.

Parameters: initial authentication details (username and password) and details about one payment service and one supply service.

Postcondition: system must be on.

Main Scenario:

1. user: turn on the system.
2. system: asks the user for initial authentication details.
3. user: enters its (new) authentication details.
4. system: verifies the validity of the given details.
5. system: opens the system, with one member and system manager that both represent the user, with the given external services.
6. system: responses a success to the user.
7. system: start serving users.

Alternative Scenarios:

- User's initial authentication details not verified successfully (4), system asks for the details again.
- Payment service or supply service are not verified successfully (4), the system asks for the details again.
- System fails preparation (5), the system cancels the action and waits for System Manager action.

Use Case I.2 - change/switch/add an external service:

A system manager changes/switches/adds an external service

Actors: system manager.

Precondition: system must be on, the system manager must be login.

Parameters: details of the new service (payment or stock)

Postcondition: the system must continue to work the same as before the change of the service, with the change of the service.

Main scenario:

1. system manager: chooses the service he wants to edit/add/switch
2. system manager: enters the details of the service
3. system: authenticates the details the user entered and his permission.
4. system: changes/add/switches the service for the new one
5. system: send positive response to the user

Alternative scenarios:

- User's not have permission to do this act successfully (3)
- service details are incorrect and requested to enter again.

Use Case I.3 : Using the services of an external payment system

the system contacts a payment system which the market is familiar with and tries to perform a payment and receive payment validation

Actors: System

Precondition: the system is activated

Parameters: details of the transaction

Postcondition: the system must get payment validation, and must continue to work the same as before the change of the service.

Main scenario:

1. the system contacts the external payment system with transaction details to perform payment
2. the external payment sends back a positive validation for the transaction the system tried to perform

Alternative scenarios:

- the transaction details the system sent are invalid
- the payment service returned a negative confirmation for the payment the system tried to do because of problem with the transaction (not because of the details)

Use Case I.4 - Using the services of an external supply service:

The system contacts a supply system which the market is familiar with and asks for approval of a valid supply.

Actors: System

Precondition: System is activated

Parameters: details of the supply

Postcondition: the system must get approval of a valid supply, and continue to work the same as before the request of service

Main Scenario:

1. system: contacts an external supply system it is familiar with and sending it the details of the required package and client information
2. external service: sends back a positive answer that the request was approved

Alternative Scenarios:

- the supply details the system sent are invalid
- The external service sends back a negative answer (2) that the request was declined, and the reason for the decline of the request.

Use Case I.5.1 real time notifications to store owner:

System must give real time notifications to logged in store owners in one of the following scenarios:

1. A product in one of the store owner’s stores is bought.
2. One of the store owner’s stores is being opened.
3. One of the store owner’s stores is being closed.
4. One of the store appointments as a store owner is being removed.

Actors: A store owner.

Precondition: System must be active, and the store owner is logged in.

Precondition: the store owner must get the notification, and system continue to work the same as before the request of service.

Main Scenario:

1. One of the scenarios that described earlier happens.
2. The system notifies the store owner about the event that occurred.

Use Case I.5.2 real time notifications to member:

System must give real time notifications to logged in member in case of receiving message

Actors: A member.

Precondition: System must be active, and the member is logged in.
Precondition: the member must get the notification, and system continue to work the same as before the request of service.
Main Scenario:

1. The member receive message.
2. The system notifies the member about the event that occurred.

Use Case I.6 - Member notifications show up when logging in:
A member logs in to the system, and the notifications that were meant for them during the time they were logged off are shown.
Actors:. Member
Precondition: System must be active, Member must be logged in.
Postcondition: System must continue to work normally as it should after any login, without considering the notifications.
Main Scenario:

1. member: logs in to the system.
2. system: shows up all notifications of the member to them.

Alternative Scenarios:

- No new notifications to show (2) so the system show to the member " There are no new notifications".

Guest User:

II.1.1 – Guest Entrance

A user enters the system as a guest, the system defines him as guest and assigns him his own shopping-cart.

Actors: User

Pre-Conditions: System server is turned on

Post-Conditions: the user is connected to the system as a guest
Parameters: None
Actions Scenario:
1. User: requests to enter the system
2. System: accepts the user
3. System: defines the user as a guest
4. System: creates a shopping cart assigned to the user
5. System: alerts the user with a response that the operation succeeded
Alternative Steps: If the system loses its connection (whether because of disconnection of the user, or any other reason) with the user during steps 2-5, it cancels the previous actions and ends the communication.

<u>II.1.2 – Guest Disconnection</u>

A guest disconnects the system, the system removes his assigned shopping cart.
Actors: Guest
Pre-Conditions: System server is turned on, the guest user is connected to the system
Post-Conditions: the user is no longer connected to the system a guest
Parameters: None
Actions Scenario:
1. Guest: requests the system to log out
2. System: Unassigns the shopping cart of the user and deletes it
3. System: Closes the connection with the user
Alternative Steps: If the user closes his connection during this action, if it happens before (2) the system deletes his shopping cart. The system then skips step 3 and acts as if the request succeeded.

<u>Use Case II.1.2.2 - Member disconnection:</u>

A Member can disconnect from the system.
<u>Actors:</u> Logged in member.
<u>Precondition:</u> System must be active, member must be logged in.

Parameters: None
Postcondition: System must continue to work normally as it should after any disconnection, member must be logged out.
Main Scenario:

1. member: send a logout request.
2. system: saves the member's cart.
3. system: disconnects the member and closes the connection.

Alternative Scenarios:

- The member closes the connection (1) without ask to logout, the system acts as a disconnection request has been received.

II.1.3 – Guest Registration

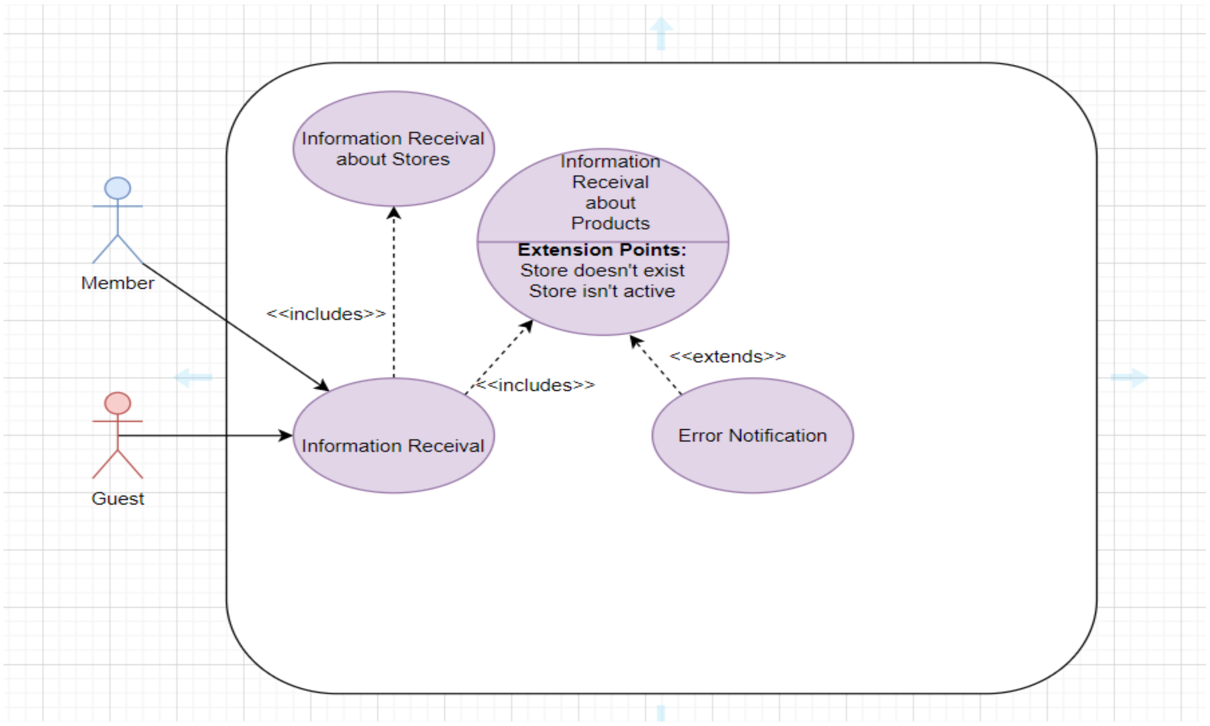
A guest registers the system giving unique authentication details, later the system recognizes him as a member when logging in using these details.
Actors: Guest
Pre-Conditions: System server is turned on, the guest user is connected to the system.
Post-Conditions: A member account with the given authentication details is registered to the system and can log in using them.
Parameters: Authentication Details
Actions Scenario:
1. Guest: requests the system to register, specifying authentication details
2. System: verifies the authentication details and checks that they are unique among the registered members of the system
3. System: Stores a new member with the given authentication details
4. System: alerts the user the action has succeeded
Alternative Steps: If the user closes his connection during this session – the system undoes the actions that were done previously and does not register the user. If the details given by the user are not unique – the system alerts the user the operation has failed.

II.1.4 – Guest Login as a Member

A guest who has registered the system as a member, can use his unique authentication details to log into the system as a member.

Actors: Guest
Pre-Conditions: System server is turned on, the guest user is connected to the system
Post-Conditions: The system changes the user state from guest to a member.
Parameters: Authentication Details
Actions Scenario:
1. Guest: requests the system to login, specifying authentication details
2. System: verifies that the specified authentication details exist in the stored member details.
3. System: changes the user state from a guest to a member
4. System: alerts the user the action has succeeded
Alternative Steps: If the user closes his connection during this session – the system undoes the actions that were done previously and does not log in the user. If the details given by the user do not exist in the system – the system alerts the user that the operation has failed.

II.2.1 – Guest Information Receival



A guest can access information about stores and products in the stores.

<u>II.2.1.A – Guest Information Receival about Stores</u>

A guest can access the information about all the active stores in the system.

Actors: Guest

Pre-Conditions: System server is turned on

Parameters: None

Actions Scenario:

1. Guest: requests information about the stores in the system
2. System: returns to the user a list of details of all the active stores in the system
3. System: alerts the user the action has succeeded

<u>II.2.1.B – Guest Information Receival about Products in a Store</u>

A guest can access the information about all the products in an active store in the system.

Actors: Guest

Pre-Conditions: System server is turned on

Parameters: Store identifier

Actions Scenario:

1. Guest: requests information about the products in a specific store, and specifies its identifier
2. System: verifies that there exists a store with this identifier in the system
3. System: returns to the user a list of details all the products in the desired store
4. alerts the user the action has succeeded
Alternative Steps: if the verification in (2) fails – the system alerts the user the action has failed.

<u>II.2.2 – Guest Search Products</u>

A guest can search product regardless of a specific store by mandatory fields (name, category, or keywords) and filter the results according to several attributes.
Actors: Guest
Pre-Conditions: System server is turned on.
Parameters: Mandatory search fields, additional filtering attributes
Actions Scenario:
1. Guest: request details of products relevant to the specified search fields and filtering attributes
2. System: verifies that the guest filled at least one of the mandatory fields
3. System: returns to the user a list of details of all the products relevant to the given search fields and filtering attributes.
4. alerts the user the action has succeeded
Alternative Steps: if the verification in (2) fails – alert the user the action has failed

<u>II.2.3 – Guest Add Products</u>

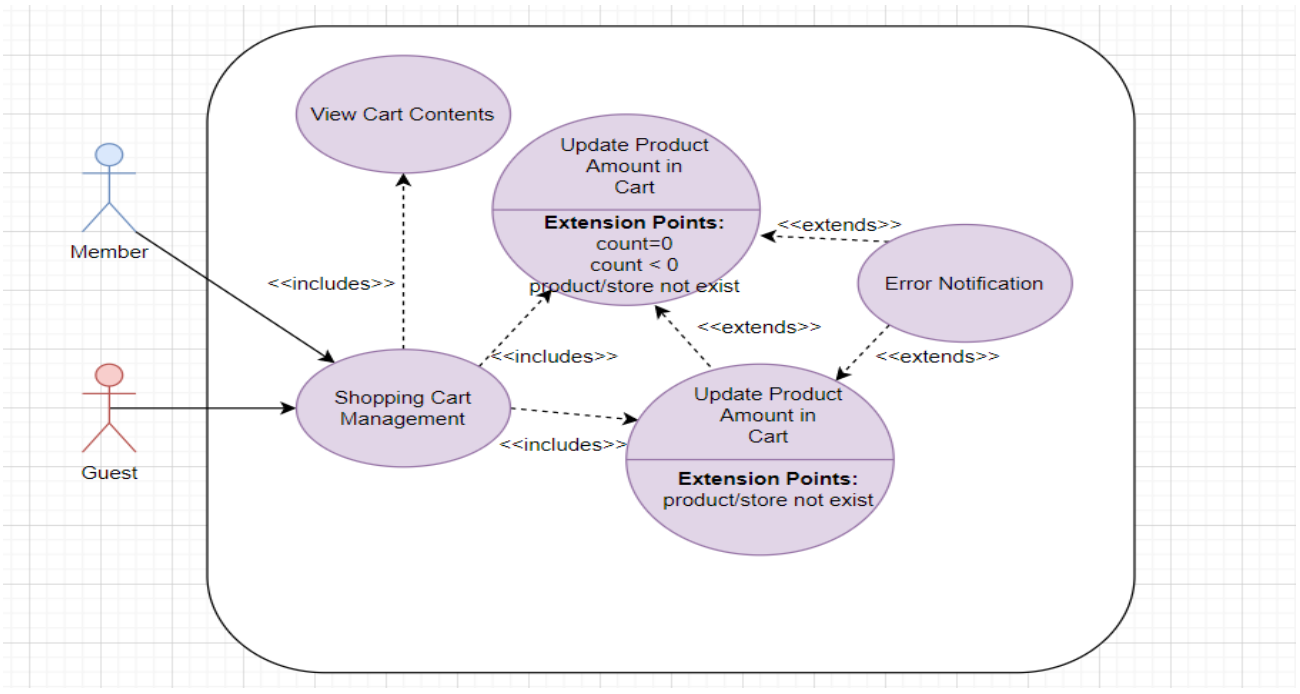
A guest can add products to his store bag, as a part of his own shopping cart.
Actors: Guest
Pre-Conditions: System server is turned on
Post-Conditions: the guest's shopping cart contains a store bag of the specified store which contains the specified amount of the specified product
Parameters: store id, product id, count

Actions Scenario:

1. Guest: request to add product from a store, also specifying the amount of such products
2. System: verifies that - store id exists in the system, product id is available in the desired store, count is a positive integer and does not exceed the number of products in the store stock.
3. System: adds the amount of the desired product to the guest's store bag, and updates his own shopping cart respectively.
4. alerts the user the action has succeeded

Alternative Steps: if the verification in (2) fails – the system alerts the user the action has failed

<u>II.2.4 – Guest Shopping Cart Management</u>



A guest can watch his shopping cart and change its contents -either remove products or edit amounts of products.

<u>II.2.4.A – Guest View Cart Contents</u>

A guest can watch his shopping cart contents.

Actors: Guest

Pre-Conditions: System server is turned on.

Parameters: None

Actions Scenario:

1. Guest: requests to watch his cart contents

2. System: returns the user a list of his car contents - its products details (including the stores they are from).
3. alerts the user the action has succeeded

<u>**II.2.4.B – Guest Remove Product from Cart**</u>

A guest can remove products from his store bag, as a part of his own shopping cart.
Actors: Guest
Pre-Conditions: System server is turned on.
Post-Conditions: the guest's shopping cart no longer contains the specified product in the specified store's bag.
Parameters: store id, product id
Actions Scenario:
1. Guest: request to remove product from a store bag in his own shopping cart
2. System: verifies that – a relevant store bag exists in the user's shopping cart and some products with the specified id appears in the desired store bag.
3. System: removes the specified product from the specified store bag in the guest's shopping cart.
4. alerts the user the action has succeeded
Alternative Steps: if the verification in (2) fails – the system alerts the user the action has failed

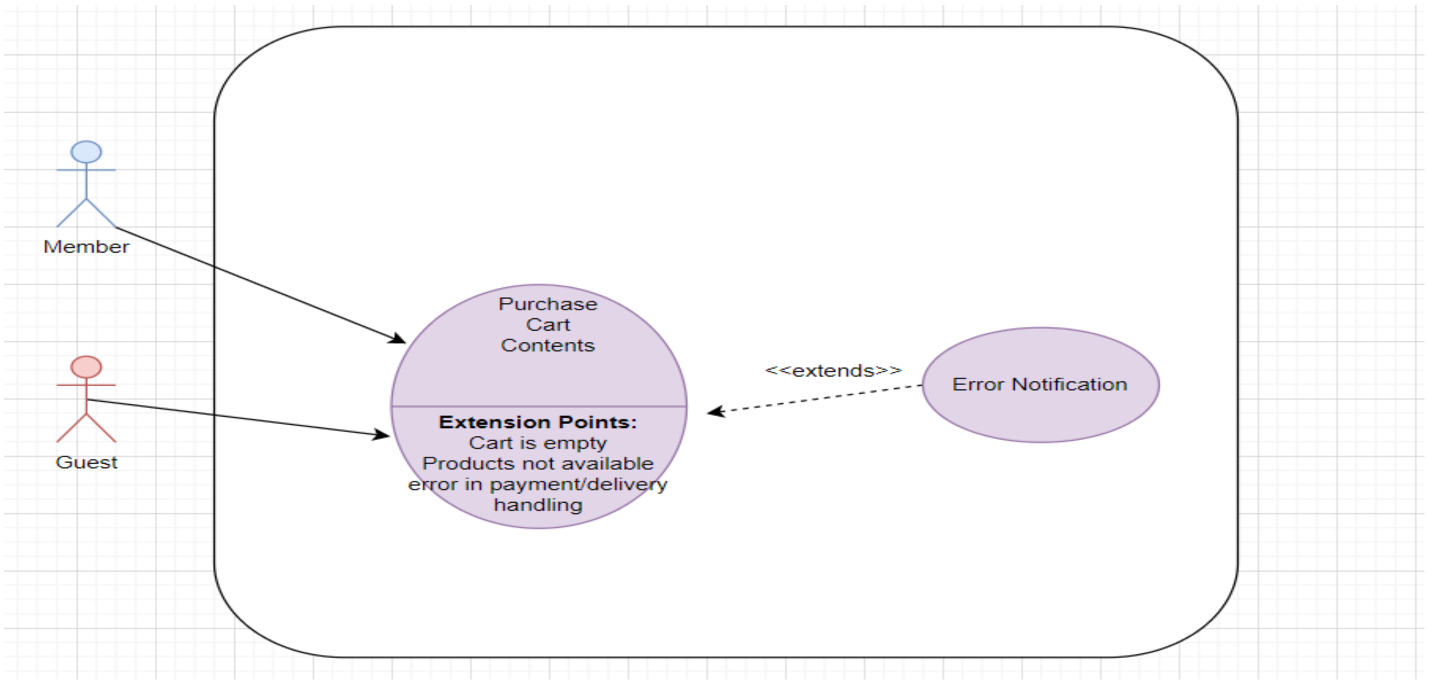
<u>**II.2.4.C – Guest Update Product Amount in Cart**</u>

A guest can change the amount of a specific product in his store bag, as a part of his own shopping cart.
Actors: Guest
Pre-Conditions: System server is turned on.
Post-Conditions: the guest's shopping contains the specified amount of the specified product in the specified store's bag.
Parameters: store id, product id, count
Actions Scenario:
1. Guest: request to change the amount of a specific product from a store bag in his own shopping cart

2. System: verifies that – a relevant store bag exists in the user's shopping cart, count is positive integer and some products with the specified id appears in the desired store bag.
3. System: sets the amount of the specified product in the specified store bag in the guest's shopping cart to be count.
4. alerts the user the action has succeeded
Alternative Steps: if the verification in (2) fails – the system alerts the user the action has failed

Use Case II.2.5 - Purchase Cart Contents:

A Guest can purchase his cart contents if it obeys the purchase and discount policies for guests, and only if all the cart contents are available in the appropriate stores.



Use Case II.2.5.a - Purchase cart content (all products are available, delivery details and payment details are valid):

A Guest can purchase his personal cart contents if all the products are available in the appropriate store.

Actors: Guest.

Precondition: System must be active, cart is non empty.

Parameters: payment method, payment details, delivery details

Post Condition: The cart is empty, the products stock quantity in the appropriate stores is decreased respectively.

Main Scenario:

1. guest: sends a purchase request.
2. system: verifies that all products are available in stores (with respect to the desired quantities).
3. system: verifies payment method and details with the payment service
4. system: verifies delivery details with the delivery service.
5. system: makes the purchase.
6. system: adds new order details to the system.
7. system: sends order details to the delivery service.
8. system: empties the user's cart
9. system: sends a response to the user with the order approval.

Alternative Scenarios:

- payment verification failed (3), the system should cancel the action and send an error response.
- delivery verification failed (4), the system should cancel the action and send an error response.

Use Case II.2.5.b - Purchase cart content (not all products available):

A Guest cannot purchase his personal cart contents if some products in it are not available.

Actors: Guest.

Precondition: System must be active, non-empty cart.

Parameters: payment method, payment details, delivery details

Main Scenario:

1. guest: send a purchase request.
2. system: verifies that all products are available in stores (with respect to the desired quantities).
3. system: sends an error response telling which products are not available

Use Case II.2.5.c - Purchase cart content (Invalid Payment details):

A Guest cannot purchase his cart content in case that the payment details he supplied are invalid.

Actors: Guest.

Precondition: System must be active, cart is non empty.

Parameters: payment method, payment details, delivery details

Main Scenario:

1.guest: sends a purchase request.

2.system: verifies that all products are available in stores (with respect to the desired quantities).
3.system: verifies payment method and details with the payment service
4.system: sends an error response telling the action has failed due to invalid payment details.

Use Case II.2.5.d - Purchase cart content (Invalid delivery details):
A Guest cannot purchase his cart content in case that the delivery details he supplied are invalid.
Actors: Guest.
Precondition: System must be active, cart is non empty.
Parameters: payment method, payment details, delivery details
Main Scenario:
1.guest: sends a purchase request.
2.system: verifies that all products are available in stores (with respect to the desired quantities).
3.system: verifies payment method and details with the payment service
4.system: verifies delivery details with the delivery service.
5.system: sends an error response telling the action has failed due to invalid delivery details.

After Supporting bids, we add the next use cases:
*** The bidding is part of the purchase requirement, but it is worth mentioning that its tests aren't too much related to the purchase as the purchase is only the final part of the bidding process that is actually unrelated to the purchase but to the displayed price of a product to the user.**
Use Case II.2.5.a_Bid- customer bids and his bit is approved
A Guest can bid on a products price, asking for a different price from the one listed by the store. The store owners can choose to accept it or not. The price of the product changes(for the user only) if all owners accepted it.
Actors: Guest and all owners of the store.
Precondition: System must be active, there is a store with a at least one product.
Parameters: productId, bid price

<u>Post Condition:</u> The product’s price changes only for the user who bidded. The bid is not active and cant be approved later on.
<u>Main Scenario:</u>
1. guest: bids on an item in a store
2. all owners: approve the bid
3. system: the price of the product changes for the user.

<u>Use Case II.2.5.b_Bid- customer bids and his bit is not approved as a result of a rejection</u>

A Guest can bid on a products price, asking for a different price from the one listed by the store. The store owners can choose to accept it or not. The price of the product changes(for the user only) if all owners accepted it.
<u>Actors:</u> Guest and all owners of the store.
<u>Precondition:</u> System must be active, there is a store with a at least one product.
<u>Parameters:</u> productId, bid price
<u>Post Condition:</u> The product’s price doesnt change. The bid is not active and cant be approved later on.

<u>Main Scenario:</u>
1. guest: bids on an item in a store
2. one of the owners: rejects the bid

<u>Use Case II.2.5.c_Bid- customer bids and his bit is not approved as a result of no approvment</u>

A Guest can bid on a products price, asking for a different price from the one listed by the store. The store owners can choose to accept it or not. The price of the product changes(for the user only) if all owners accepted it.
<u>Actors:</u> Guest and all owners of the store.
<u>Precondition:</u> System must be active, there is a store with a at least one product.
<u>Parameters:</u> productId, bid price
<u>Post Condition:</u> The product’s price doesnt change. The bid is still active and can be approved later on.
<u>Main Scenario:</u>
1. guest: bids on an item in a store
2. one of the owners: doesnt approve

<u>Use Case II.2.5.d_Bid- customer bids and his bit is countered. Then the user decides to accept the counter bid, all the owners accept his new bid.</u>

A Guest can bid on a product's price, asking for a different price from the one listed by the store. The store owners can choose to counter it. Now the user can decide to approve it or reject it.
If he approves the counter, approval of all owners for his new bid(as part of the approval) is required for the price to change(as before)
<u>Actors:</u> Guest and all owners of the store.
<u>Precondition:</u> System must be active, there is a store with a at least one product.
<u>Parameters:</u> productId, bid price, counter bid price
<u>Post Condition:</u> The product’s price changes.
<u>Main Scenario:</u>

1. guest: bids on an item in a store(with price Y)
2. one of the owners: counters the offer with a new proposal of a bid(with price X)
3. guest: approves the bid proposal(this sends a new bid with price X)
4. all owners: approve the bid
5. system: the price of the product changes(to X) for the user.

<u>Use Case II.2.5.e_Bid- customer bids and his bit is countered. Then user decides to reject the counter bid</u>

A Guest can bid on a products price, asking for a different price from the one listed by the store. The store owners can choose to accept it or not. The price of the product changes(for the user only) if all owners accepted it.
<u>Actors:</u> Guest and all owners of the store.
<u>Precondition:</u> System must be active, there is a store with a at least one product.
<u>Parameters:</u> productId, bid price, counter bid price
<u>Post Condition:</u> The product’s price doesnt change.
<u>Main Scenario:</u>

1. guest: bids on an item in a store(with price Y)
2. one of the owners: counters the offer with a new proposal of a bid(with price X)
3. guest: rejects the bid proposal.

The members are able to do **II.2.1-II.2.5** exactly the same way as guests except that now new precondition is that the user must be logged in.

Use Case II.3.1 - Member logout:

A Member can logout from the system.

Actors: Logged in member.

Precondition: System must be active, member must be logged in.

Parameters: None

Postcondition: System must continue to work normally as it should after any logout, member must be logged out and be treated as a guest.

Main Scenario:

1. member: send a logout request.
2. system: saves the member's cart.
3. system: logs out the member.
4. system: change the user state to be guest.
5. system: sends a successful response with a new empty cart.

Alternative Scenarios:

- The member closes the connection (1) without ask to logout, the system acts as a disconnection request has been received.

Use Case II.3.2 - Create new store:

A Member can create a new store and become the first store owner.

Actors: Logged in member.

Precondition: System must be active, member must be logged in.

Parameters: store information

Postcondition: System must continue to work normally, a new store whose founder is this user exists in the system.

Main Scenario:

1. member: sends request for creating new store with store information.
2. system: checks the validity of the new store details.
3. system: creates a new store with given details.

4. system: sets the member to be the founder of the store.
5. system: sends a successful response with store id.

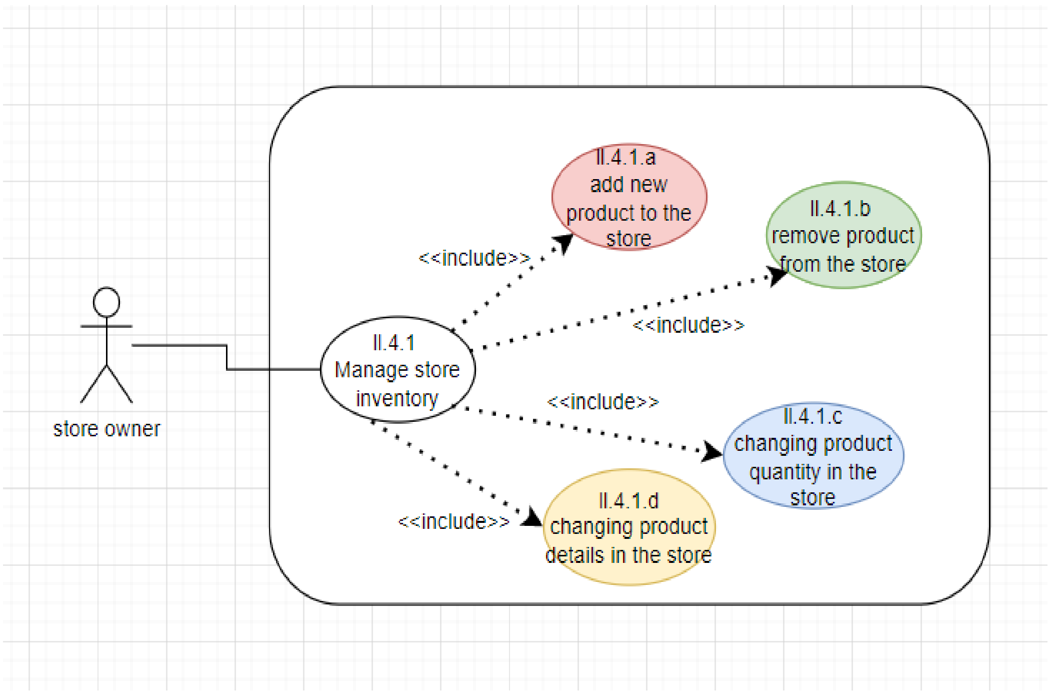
Alternative Scenarios:

- The member gives not valid details (one or more) (2), the system asks to enter the information again.

Use Case II.4.1 - Inventory management:

An owner can manage the store inventory: adding and remove products, changing the products' details.

A manager can also do all these things if his assigner gives him the permission.


II.4.1.a
add new product to the store
II.4.1.b
remove product from the store
<<include>>
<<include>>
store owner
II.4.1
Manage store inventory
<<include>>
II.4.1.c
changing product quantity in the store
<<include>>
II.4.1.d
changing product details in the store

Owner

Use Case II.4.1.a - Adding new product to the store:

The store owner can add product to the store.

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in.

Parameters: new product details.

<u>Postcondition:</u> System must continue to work normally, a new product appears in the relevant store.
<u>Main Scenario:</u>

1. owner: sends request for adding new product to the store.
2. system: creates new product collection using the given details.
3. system: sends a successful response.

<u>Use Case II.4.1.b - Removing products from the store:</u>
<u>Actors:</u> Store owner.
<u>Precondition:</u> System must be active, store must be active, store owner must be logged in, product id must be of an active product in store.
<u>Parameters:</u> product id.
<u>Main Scenario:</u>

1. owner: sends request for removing existing product from the store.
2. system: remove product from the store.
3. system: sends a successful response.

<u>Use Case II.4.1.c - Changing product’s quantity in inventory:</u>
<u>Actors:</u> Store owner.
<u>Precondition:</u> System must be active, store must be active, store owner must be logged in, product id must be of an active product in the store.
<u>Parameters:</u> product id, new quantity.
<u>Main Scenario:</u>

1. owner: sends request for updating product’s quantity in the store(with a legal(positive) quantity).
2. system: update product’s quantity in store inventory.
3. system: sends a successful response.

Alternative Scenario:

1. <u>Illigeal Input</u>
    a. owner: sends request for updating product’s quantity in the store(with a not legal(negative) quantity).
    b. system:return an error response and dont change quantity.

<u>Use Case II.4.1.d - Changing product’s price:</u>

<u>Actors:</u> Store manager.

<u>Precondition:</u> System must be active, store must be active, store manager must be logged in, product id must be of an active product in the store.

<u>Parameters:</u> product id, new price.

<u>Main Scenario:</u>

1. manager: sends request for updating product’s price in the store(with a legal(positive) price).
2. system: update product’s price in store inventory.
3. system: sends a successful response.

Alternative Scenario:

1. <u>Illigeal Input</u>
    1. manager: sends request for updating product’s price in the store(with a not legal(negative) price).
    2. system:return an error response and dont change price.

<u>Use Case II.4.1.e - Changing product’s details in the store:</u>

<u>Actors:</u> Store manager.

<u>Precondition:</u> System must be active, store must be active, store manager must be logged in, product id must be of an active product in the store.

<u>Parameters:</u> product id, new product’s details.

<u>Main Scenario:</u>

1. manager: sends request for updating product’s details in the store.
2. system: update product’s details in store.
3. system: sends a successful response.

Manager

Use Case II.4.1.a - Adding a new product to the store:

The store manager can add a product to the store.
Actors: Store manager.
Precondition: System must be active, store must be active, store manager must be logged in.
Parameters: new product details.
Postcondition: The system must continue to work normally, and a new product appears in the relevant store.
Main Scenario:

4. Manager: sends a request for adding a new product to the store.
5. system: creates a new product collection using the given details.
6. system: sends a successful response.

Alternative Scenario:

1. No Permission:
    a. manager(without permission to add a new product): sends a request for adding a product to the store
    b. system: returns an error response indicating the manager doesn't have permission for adding products and doesn't add the product requested.

Use Case II.4.1.b - Removing products from the store:

Actors: Store manager.
Precondition: System must be active, store must be active, store manager must be logged in, and product id must be of an active product in store.
Parameters: product id.
Main Scenario:

4. Manager: requests to remove an existing product from the store.
5. system: removes the product from the store.
6. system: sends a successful response.

Alternative Scenario:

2. No Permission:
    a. manager(without permission to add a new product): sends a request for removing a product from the store

b. system: returns an error response indicating the manager doesn't have permission for removing products and doesn't remove the product requested.

Use Case II.4.1.c - Changing product's quantity in inventory:

Actors: Store manager.

Precondition: The system must be active, the store must be active, the store manager must be logged in, and the product id must be of an active product in the store.

Parameters: product id, new quantity.

Main Scenario:

4. Manager: sends a request for updating the product's quantity in the store(with a legal(positive) quantity).
5. system: update product's quantity in-store inventory.
6. system: sends a successful response.

Alternative Scenario:

2. Illigeal Input
    c. Manager: sends a request for updating the product's quantity in the store(with a not legal(negative) quantity).
    d. system: return an error response and don't change quantity.
2. No Permission:
    a. manager(without permission to edit quantity): sends a request for updating product's quantity in the store
    b. system: returns an error response indicating the manager doesn't have permission for editing the details of the product and doesn't change quantity.

Use Case II.4.1.d - Changing product's price:

Actors: Store manager.

Precondition: The system must be active, the store must be active, the store manager must be logged in, product id must be of an active product in the store.

Parameters: product id, new price.

Main Scenario:

4. manager: sends a request for updating product’s price in the store(with a legal(positive) price).
5. system: update product’s price in store inventory.
6. system: sends a successful response.

Alternative Scenario:

2. Illigeal Input
    3. manager: sends a request for updating product’s price in the store(with a not legal(negative) price).
    4. system: returns an error response and doesnt change the price.
3. No Permmision:
    1. manager(without permission to edit price): sends a request for updating product’s price in the store
    2. system: return an error response indicating the manager doesnt have the permission for editing the products details and dont change the price.

Use Case II.4.1.e - Changing product’s details in the store:

Actors: Store manager.

Precondition: System must be active, store must be active, store manager must be logged in, product id must be of an active product in the store.

Parameters: product id, new product’s details.

Main Scenario:

4. manager: sends request for updating product’s details in the store.
5. system: update product’s details in store.
6. system: sends a successful response.

Alternative Scenario:

3. No Permmision:
    a. manager(without permission to edit details): sends request for updating product’s details in the store

b. system:return an error response indicating the manager doesnt have the permission for editing the products details and dont change details.

II. 4.2 - Change type of purchases and discount policy

The store owner can change the purchases and discount policy of the .store

Actors: Owner

Pre-Conditions: System must be active, store must be active, store owner must be logged in

Parameters: store id, purchases and discount policy

:Actions Scenario

1. owner: sends request for changing policy in the store.
2. system: sets new policy for the store.
3. system: sends a successful response.

Post-condition: New policy should apply

Alternative Steps: The parameter that given is not valid id, the system .returns error message with the specific information

II. 4.4 – Make Store Owner

.The store owner can make other member to be also the store owner

Actors: Owner

Pre-Conditions: System must be active, store must be active, store owner must be logged in

Parameters: member id

:Actions Scenario

1. owner: sends request for making new owner in the store.
2. system: verify member exists and not already owner.
3. system: sets new owner under the actor’s (owner) in the hierarchy.
4. system: sends a successful response

(Post-condition: Store owners hierarchy is a tree (no cycles

Alternative Steps: The member does not exist (2), the system cancels .the action

The member is already an owner of the store (2), the system cancels .the action

<u>II. 4.5 – Remove Store Manager</u>

The store owner can remove a user from being a store manager if he .is the one who added him

Actors: Owner

Pre-Conditions: System must be active, store must be active, store owner must be logged in

Parameters: member id, store id

:Actions Scenario

1. owner: sends request for removing a store manager.
2. system: verify member exists and is a manager.
3. system: verify the owner is the one who added the chosen manager.
4. system: remove the store manager
5. system: sends a successful response

Post-condition: selected member is no longer a manager of the store

Alternative Steps: The member does not exist (2), the system cancels .the action

The member is not a manager of the store (2), the system cancels the action

The member was not added by the actor owner (3), the system cancels the action

<u>II. 4.6 – Make Store Manager</u>

.The store owner can make other member to be a store owner

Actors: Owner

Pre-Conditions: System must be active, store must be active, store owner must be logged in

Parameters: member id

:Actions Scenario

1. owner: sends request for making new manager in the store.
system: verify member exists and not already manager. 2.
.system: sets the new manager with the owner as appointor.3
system: sends a successful response.4

Post-condition: The member role is assigned to be new store .manager, and the assigner is the actor

Alternative Steps: The member does not exist (2), the system cancels .the action
The member is already a manager of the store (2), the system cancels .the action

II.4.7 – Owner Update Store Manager Permission
.Store can edit the permissions each his store's managers has
Actors: Owner
Pre-Conditions: System server is turned on, store owner is logged in, .store is active
.Post-Conditions: the manager's new permissions are updated
Parameters: store id, manager's id, new permissions
:Actions Scenario
Owner: requests to edit store manager's permission .1
.System: verifies that the store id exists in the system and is active .2
.System: verifies that the owner owns this store .3
.System: verifies that the manager manages this store .4
.System: verifies that the manager was appointed by this owner .5
.System: edits the manager permissions respectively .6
System: alerts the owner the action has succeeded .7
Alternative Steps: if any of the verifications (2) - (5) fail, the system .alerts the user the action has failed

II.4.9 – Founder Close Store
.A founder can make his store inactive
Actors: Founder
Pre-Conditions: System server is turned on, founder is logged in, the store owners and managers are available in the system
Post-Conditions: the products in the newly closed store won't appear in the product search results, the store owners and managers are notified about their newly closed store
Parameters: founder's member identifier, the identifier of the store desired to be closed
:Actions Scenario

Founder: requests to close one of his stores .1
.System: verifies that the store id exists in the system .2
.System: verifies that the founder id exists .3
.System: verifies that the store was indeed founded by this founder .4
.System: verifies that the store is active .5
.System: closes the store .6
System: notify the store owners and managers that this store was .7
closed
System: alerts the founder the action has succeeded .8
Alternative Steps: if any of the verifications (2) - (5) fail, the system
.alerts the user the action has failed

<u>II.4.11 – Owner Get Store Roles Details</u>

Store owner can access the information about the roles holders in his
.store, as well as the permissions his store's managers have

<u>II.4.11.A- Get Roles Holders Details</u>

Actors: Owner
Pre-Conditions: System server is turned on, store owner is logged in
Parameters: store owner's member id, store id
:Actions Scenario
Owner: requests to get information about role holders in his store .1
.System: verifies that the store id exists in the system .2
.System: verifies that the owner owns this store .3
System: returns a list with data about each role holder in this shop .4
to the owner
alerts the owner the action has succeeded .5
Alternative Steps: if any of the verifications (2) - (3) fail, the system
.alerts the user the action has failed

<u>II.4.11.B - Get Managers' Permissions</u>

Actors: Owner
Pre-Conditions: System server is turned on, store owner is logged in
Parameters: store owner's member id, store id
:Actions Scenario
Owner: requests to get managers' permissions in his store .1
.System: verifies that the store id exists in the system .2
.System: verifies that the owner owns this store .3

System: returns a list with the permissions for each manager in this .4 shop to the owner
System: alerts the owner the action has succeeded .5
Alternative Steps: if any of the verifications (2) - (3) fail, the system .alerts the user the action has failed

II.4.13 - Owner Get Purchase History in the Store

.Store owner can get the purchase history in his stores
Actors: Owner
Pre-Conditions: System server is turned on, store owner is logged in
Parameters: store owner's member id, store id
:Actions Scenario
Owner: requests to get the purchase history in one of his stores .1
.System: verifies that the store id exists in the system .2
.System: verifies that the owner owns this store .3
System: returns a list with the all the purchases happened in this .4 store to the owner
System: alerts the owner the action has succeeded .5
Alternative Steps: if any of the verifications (2) - (3) fail, the system .alerts the user the action has failed

(II.6.2 - Remove system member (by system manager

The system manager can cancel the membership of an existing .member in the system as long as he's not holding a special role
.Actors: System manager
Pre-Conditions: System must be active, system manager must be logged in
Parameters: Admin id, member id
:Actions Scenario
1. The admin requests to remove the membership of a member with the specified id as well as specifying his own credentials.
2.The system verifies that the admin of the given id exists in the system.
3.The system verifies that the member of the given id exists in the system.
4.The system verifies that the member of the given id exists does not hold a special role - i.e, not a founder/owner or manager of some store.
5. The system removes the membership of the specified member.

Alternative Steps: The admin id entered by the system manager is for an admin not currently existing in the system so the action can't be performed so an error message is returned via a notification
The member id entered by the system manager is does not exist in the system so the action can't be performed so an error message is returned via a notification
The member id entered by the system manager corresponds to a store founder/owner or manager so the action can't be performed so an error message is returned via a notification

(II.6.4 - Get Purchase History of a Store/Buyer (by system manager

.The system manager can get the purchase history of a store/buyer
.Actors: System manager
Pre-Conditions: System must be active, system manager must be logged in
Parameters: Admin id, store/buyer id
:Actions Scenario
1. The admin requests the purchase history of the store/buyer and inputs the required credentials(admin id) and the information regarding the store/buyer(store/buyer id).
2.The system verifies that the admin of the given id exists in the system, and that the store/buyer id corresponds to an existing store/buyer in the system.
The system outputs the purchase history of the entered store/buyer.3
Alternative Steps: The admin id entered by the system manager is for an admin not currently existing in the system so the action can't be .performed so an error message is returned via a notification
The store/buyer id does not exist in the system so the action can't be .performed so an error message is returned via a notification
.

II.6.6 - Get Information about Logged in/out members (by system (manager

The system manager can get information about logged in and logged .out system members
.Actors: System manager
Pre-Conditions: System must be active, system manager must be logged in
Parameters: Admin id, logged in/out boolean

:Actions Scenario

1. The admin requests information about logged in/out members as well as his credentials.
2.The system verifies that the admin of the given id exists in the system.
3. The system returns information about logged in/out members respectively to the parameter supplied.

Alternative Steps: The admin id entered by the system manager is for an admin not currently existing in the system so the action can’t be ..performed so an error message is returned via a notification