Use Cases & Acceptance Tests

[USERS 3](#_Toc172138019)

[Use case: **Enter Market system** 3](#_Toc172138020)

[Use case**: Leave Market system** 4](#_Toc172138021)

[Use case: **Register** 5](#_Toc172138022)

[Use case: ***Login*** 6](#_Toc172138023)

[Use case: **Get information** 7](#_Toc172138024)

[Use case: **Search products** 8](#_Toc172138025)

[Use case: **Global product search** 9](#_Toc172138026)

[Use case: **Save product to basket** 11](#_Toc172138027)

[Use case: **Check shopping cart** 12](#_Toc172138028)

[Use case: **Shopping cart purchase** 13](#_Toc172138029)

[Use case: **Purchase cart - Missing product** 14](#_Toc172138030)

[Use case: **purchase cart- check policies** 16](#_Toc172138031)

[Use case: **purchase cart- check Discounts** 17](#_Toc172138032)

[Use case: **purchase cart - payment** 19](#_Toc172138033)

[Use case**: *Log out*** 20](#_Toc172138034)

[Use case**: *Open store*** 21](#_Toc172138035)

[Use case: **Edit product inventory** 22](#_Toc172138036)

[Use case: **Edit discount or buy policy** 23](#_Toc172138037)

[Use case: **Get purchase history** 24](#_Toc172138038)

[Store Owner 25](#_Toc172138039)

[Use case: **Inventory Management** 25](#_Toc172138040)

[Use case: **Add Product** 26](#_Toc172138041)

[Use case: **Remove Product** 27](#_Toc172138042)

[Use case**: Update Product** 29](#_Toc172138043)

[Use case**: Manage Purchase and Discount Policies on Products** 30](#_Toc172138044)

[Use case: **Add Purchase Policy on Product** 32](#_Toc172138045)

[Use case: **Add Discount Policy on Product** 33](#_Toc172138046)

[Use case: **Add Composite Policy** 34](#_Toc172138047)

[Use case: **Remove Purchase Policy on Product** 35](#_Toc172138048)

[Use case: **Remove Discount Policy on Product** 36](#_Toc172138049)

[Use case: **Update Purchase Policy on Product** 37](#_Toc172138050)

[Use case**: Update Discount Policy on Product** 38](#_Toc172138051)

[Use case**: Manage Purchase and Discount Policies on Category** 40](#_Toc172138052)

[Use case: **Add Purchase Policy on Category** 41](#_Toc172138053)

[Use case: **Add Discount Policy on Category** 42](#_Toc172138054)

[Use case: **Remove Purchase Policy on Category** 44](#_Toc172138055)

[Use case: **Remove Discount Policy on Category** 45](#_Toc172138056)

[Use case: **Update Purchase Policy on Category** 46](#_Toc172138057)

[Use case: **Update Discount Policy on Category** 48](#_Toc172138058)

[Use case: **Nomination of a Store Owner** 49](#_Toc172138059)

[Use case: **Nomination of a Store Manager** 51](#_Toc172138060)

[Use case: **Modifying Store Manager Privileges** 52](#_Toc172138061)

[Use case: **Closing a Store** 53](#_Toc172138062)

[Use case: **Request Information about Positions in the Store** 55](#_Toc172138063)

[Trading System Manager 56](#_Toc172138064)

[Use case: **Access Purchase History in the Trading System** 56](#_Toc172138065)

# USERS

### Use case: **Enter Market system**

* + **Actor:** User
  + **Precondition:** 
    1. User has access to the market system.
    2. User is not already in the system.
  + **Parameter:**
  + **Actions:**
    1. User enters the Market system.
    2. System present landing page with all available actions in the market.
    3. System allocates shopping cart to the user and mark it as Buyer.
  + **Post conditions:**
    1. User is in the market system, can preform actions.
    2. User gets identifier in the system.

Acceptance test:

* **Happy- Success entering Market system**
  + User enters the Market system.
  + System present landing page with relevant actions.
  + System allocates a shopping cart.
  + User can fill the cart with products.
  + User title is changed to "Buyer".
  + User has identifier in the system.
* **Bad- Fail entering Market system**
  + User enters the Market system.
  + System crash and doesn't present the landing page.
  + User doesn't get a shopping cart.
* Use case: ***Leave Market system***
  + **Actor:** guest user
  + **Precondition:** User is in the Market system.
  + **Parameter:**
  + **Actions:**
    1. User leave the Market system.
    2. System delete it's shopping cart and remove it’s mark.
  + **Post conditions:**
    1. User is outside the market system, can't preform actions.
    2. User's shopping cart delete.

Acceptance test:

* **Happy- Success leaving the system- cart deleted.**
  + User is in the Market system.
  + User leave the Market system.
  + System delete it's shopping cart
  + User is outside the market system.
* **Bad- Fail – cart saved**
  + User is in the Market system.
  + User leave the Market system.
  + System saves it's shopping cart
  + User is outside the market system.

### Use case**: Leave Market system**

* + **Actor:** logged in user
  + **Precondition:**
    1. subscribed and logged in
    2. User is in the Market system.
  + **Parameter:**
  + **Actions:**
    1. User leave the Market system.
    2. System saves it's shopping cart in the DB for future log in.
  + **Post conditions:**
    1. User is outside the market system, can't preform actions.
    2. User's shopping cart saved.

Acceptance test:

* **Happy-** **Success leaving the system- cart saved.**
  + User is in the Market system.
  + User logged in.
  + User leave the Market system.
  + System saves it's shopping cart
  + User is outside the market system.
* **Bad-** **Fail – cart deleted**
  + User is in the Market system.
  + User logged in.
  + User leave the Market system.
  + System deletes it's shopping cart
  + User is outside the market system.

### Use case: **Register**

* + **Actor:** User
  + **Precondition:**
    1. User is not already registered.
  + **Parameter:** *unique personal details*
  + **Actions:**
    1. User register the system with personal details asked by the system.
    2. System checks the correctness of the personal details.
    3. If wrong details: ask to enter again.
    4. System adds the user as a subscription to the Market system.
    5. System send the user success message.
  + **Post conditions:**
    1. User is registered to the market system.

Acceptance test:

* **Happy- Success registration**
  + User press register add enter correct personal details to register.
  + System checks the details and approve it.
  + System adds the user as a subscribed user.
  + System sends a confirmation message to the user.
* **Sad- Fail registration – problem with personal details**
  + User press register add enter incorrect personal details to register – wrong email address.
  + System checks the details and ask the user to correct the email.
  + User enter correct email address.
  + System checks the email address and approve it.
  + System adds the user as a subscribed user.
  + System sends a confirmation message to the user.
* **Sad - Fail registration – User is registered.**
* **Bad- Fail registration – can’t handle wrong email**
  + User press register add enter incorrect personal details to register – wrong email address.
  + System checks the details and approve it.
  + System adds the user as a subscribed user.
  + System try to send confirmation message to the user and fail due to incorrect email address, and show an error message. The user register with incorrect email

### Use case: ***Login***

* + **Actor:** User
  + **Precondition:**
    1. User registered
    2. User is not logged in (from any computer)
  + **Parameter:** *personal details*
  + **Actions:**
    1. System checks the correctness of the personal details.
    2. If wrong details: ask to enter again.
    3. User marked as subscribed.
    4. System send the user success message.
  + **Post conditions:**
    1. User is logged in to the market system.

Acceptance test:

* **Happy- Success log in**
  + Register user with correct personal details.
  + User press login add enter correct personal details to be identified.
  + System find the details in the registered users stock.
  + User get in the system.
  + System mark the user as subscribed user.
  + System sends a confirmation message to the user.
* **Sad- Fail log in – incorrect personal details.**
  + Register user with correct personal details.
  + User press login add enter incorrect personal details to be identified.
  + System doesn't find the details in the registered users stock.
  + System show an error message that can’t find the user in the registered users stock and ask to reenter the details.
* **Sad – Fail log in – User is already logged in from another computer.**
* **Bad- Fail log in – can’t handle wrong personal details.**
  + Register user with correct personal details.
  + User press login add enter incorrect personal details to be identified.
  + System doesn't find the details in the registered users stock.
  + System crash and the user stays logged out and without information.

### Use case: **Get information**

* + **Actor:** Subscribed visitor
  + **Precondition:**
    1. User is subscribed to the system.
  + **Parameter:** *store on which information is needed* and on products of wants.
  + **Actions:**
    1. User ask for information on specific store and products.
    2. System verify the store mentioned and if doesn’t find it – show error and ask the user to enter another store.
    3. If user specified products -> system shows information on the products if have, else show global information about the store.
    4. If user didn’t specify products -> system shows global information about the store.
  + **Post conditions:**
    1. Information required are visible to the user.

Acceptance test:

* **Happy- Success information request.**
  + Register user with correct personal details.
  + User press information and enter a store and a products.
  + System find the store.
  + System search for the products in the store- if find- return the information about those products. Else return global information about the store.
* **Sad- Fail information request – bad store provided.**
  + Register user with correct personal details.
  + User press information and enter a store and a products.
  + System doesn’t find the store.
  + System return an error message and ask for another store.
* **Sad – Fail information request – bad product provided.**
* **Bad- Fail information request – can’t handle wrong product/ store provided.**
  + Register user with correct personal details.
  + User press information and enter a store and a products.
  + System doesn’t find the store.
  + System crash and doesn’t return anything to the user.

### Use case: **Search products**

* + **Actor:** Subscribed visitor
  + **Precondition:**
    1. User is inside the system.
  + **Parameter:** *product name, category, store name or key words*
  + **Actions:**
    1. System suggest two options: "Global search" or "Store search"
    2. If user selects “Global search“ -> Use case "Global product search".
    3. If user selects "Store search":
       1. System ask for store.
       2. User enter store name.
       3. System checks validness of the store and show error of the store is invalid.
       4. If the store is valid - System search in the DB of the specified store according to the parameters provided.
       5. System ask if filter.
       6. If user select yes ->
          - System ask for filter parameters
          - User provide parameters
          - System checks validness of the parameters provided and if not valid- show error and ask filter again.
          - System filter the results and return the results to the user
       7. If user select no ->
          - System return the results to the user.
  + **Post conditions:**
    1. Relevant information about the products are visible to the user.
    2. Only information in the specific store are visible to the user.
    3. If asked- the information is filtered.

Acceptance test:

* **Happy- Success search**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose "store search".
  + System ask for store.
  + User enter valid store.
  + System gets information on the specific store.
  + System ask if filter.
  + User enter filter keys.
  + System filter the data and return filtered information on the store to the user.
* **Sad- Fail search – invalid store provided.**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose "store search".
  + System ask for store.
  + User enter invalid store.
  + System doesn’t find the store and return error of invalid store.
  + User enter valid store.
  + System find the store and gets information on the specific store.
  + System ask if filter.
  + User enter filter keys.
  + System filter the data and return filtered information on the store to the user.
* **Sad – Fail search – invalid filter parameters provided.**
* **Bad- Fail search – can’t handle invalid store or filter parameters.**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose "store search".
  + System ask for store.
  + User enter invalid store.
  + System doesn’t find the store, crash and doesn’t return anything to the user.

### Use case: **Global product search**

* + **Actor:** Subscribed visitor
  + **Precondition:**
    1. User is inside the system.
  + **Parameter:** *product name, category or key words*
  + **Actions:**
    1. System search in the DB of all stores according to the parameters provided.
    2. System ask if filter.
    3. If user select yes ->
       1. System ask for filter parameters
       2. User provide parameters
       3. System checks validness of the parameters provided and if not valid- show error and ask filter again.
       4. System filter the results and return the results to the user
    4. If user select no ->
       1. System return the results to the user.
  + **Post conditions:**
    1. Relevant information about the products are visible to the user.
    2. All information from all stores are visible to the user.
    3. If asked- the information is filtered.

Acceptance test:

* **Happy- Success search**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose " global search".
  + System gets information on all stores.
  + System ask if filter.
  + User enter valid filter keys.
  + System checks validness of the parameters provided.
  + System filter the data and return filtered information on the store to the user.
* **Sad- Fail search – invalid filter keys**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose " global search".
  + System gets information on all stores.
  + System ask if filter.
  + User enter invalid filter keys.
  + System checks validness of the parameters provided, show error and ask the user to enter filter again.
  + User enter valid filter keys.
  + System checks validness of the parameters provided.
  + System filter the data and return filtered information on the store to the user.
* **Bad- Fail search – can’t handle invalid filter keys.**
  + Register user with correct personal details.
  + User press search product.
  + System suggest "global search" or "store search".
  + User choose " global search".
  + System gets information on all stores.
  + System ask if filter.
  + User enter invalid filter keys.
  + System checks validness of the parameters provided, crash and doesn’t return anything to the user.

### Use case: **Save product to basket**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. User is inside the system.
  + **Parameter:** product, store
  + **Actions:**
    1. User press save for specific product in a specific store.
    2. System add the product to the basket of this store.
    3. If the user is logged in -> System saves changes in the DB.
    4. System show confirmation message to the user.
  + **Post conditions:**
    1. Product are saved in the basket of the specific store.
    2. If the user is logged in -> All changes saved in the DB.

Acceptance test:

* **Happy- Success saving product.**
  + Register user with correct personal details.
  + User get into a specific store and find a specific product.
  + User press save on the product.
  + System add the product to the user's basket.
  + System show confirmation message to the user.
* **Sad- Fail saving product – product is missing (someone just bought it)**
  + Register user with correct personal details.
  + User get into a specific store and find a specific product.
  + User press save on the product.
  + Product can't be saved.
  + System show error message to the user with the reason to the fail.
* **Bad- Fail saving product – can't handle missing product.**
  + Register user with correct personal details.
  + User get into a specific store and find a specific product.
  + User press save on the product.
  + Product can't be saved.
  + System crash and doesn't show error message with explanation to the user.

### Use case: **Check shopping cart**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. User is inside the system.
  + **Parameter:**
  + **Actions:**
    1. User press check shopping cart.
    2. System shows the shopping cart. If empty show empty.
  + **Post conditions:**
    1. Relevant information on the shopping cart is shown to the user.

Acceptance test:

* **Happy- success action.**
  + Register user with correct personal details.
  + User press show shopping cart.
  + System show the shopping cart to the user.
* **Sad- Fail action – nothing to show – empty shopping cart.**
  + Register user with correct personal details.
  + User press show shopping cart.
  + System doesn’t have anything to show, it shows an error message to the user.
* **Bad- Fail action – can't handle empty shopping cart.**
  + Register user with correct personal details.
  + User press show shopping cart.
  + System doesn’t have anything to show, it crashes as doesn’t show anything to the user.

### Use case: **Shopping cart purchase**

* + **Actor:** subscribed visitor
  + **Precondition:** User Logged in
  + **Parameter:** store
  + **Actions:**
    1. System checks the availability of each product in each store basket in the shopping cart.
    2. System update products quantity in each store.
    3. If missing product:
       1. Use case – "Purchase cart - Missing product".
    4. Use case – "Purchase cart – check policies".
    5. Use case - "Purchase cart – check discounts".
    6. Use case – "Purchase cart – Payment".
    7. Subscribed visitor enter paying method and payment details.
    8. Subscribed visitor approve payment details.
    9. Use case – "Purchase cart – Payment"
    10. Allert is being sent to the store owner about the purchase of the products.
  + **Post conditions:**
    1. A reservation has been made.
    2. Payment has been made.
    3. Purchase has been saved with all the relevant details.
    4. Products quantities updated.
    5. Allert has been sent to the store owner.

Acceptance test:

* **Happy- Success purchase**
  + Register user with correct personal details.
  + User add products to the shopping cart.
  + User press Purch.
  + System check the availability of each product in each store basket in the shopping cart.
  + System checks if the products contains alcoholic or cigarettes products.
  + There are no such products.
  + No product is missing.
  + System calculate the price of the shopping cart according to the discount and buying policy of the store.
  + Subscribed visitor enter paying method and payment details.
  + Subscribed visitor approve payment details.
  + System perform the payment using outside payment system.
  + Payment succeed.
  + System make the reservation and send receipt to the subscribed visitor.
  + Store owner got notification.
  + Purchase is saved in the purchase history DB.
* **Sad- Fail purchase – missing product.**
  + Register user with correct personal details.
  + User add products to the shopping cart.
  + User press Purch.
  + System check the availability of each product in each store basket in the shopping cart.
  + System checks if the products contains alcoholic or cigarettes products.
  + There are no such products.
  + One product is missing.
  + System cancels the purchase and send message to the subscribed visitor.
* **Bad- Fail purchase – can't handle missing product.**
  + Register user with correct personal details.
  + User add products to the shopping cart.
  + User press Purch.
  + System check the availability of each product in each store basket in the shopping cart.
  + System checks if the products contains alcoholic or cigarettes products.
  + There are no such products.
  + One product is missing.
  + System calculate the price of the shopping cart according to the discount and buying policy of the store.
  + Subscribed visitor enter paying method and payment details.
  + Subscribed visitor approve payment details.
  + System perform the payment using outside payment system.
  + System make the reservation and send receipt to the subscribed visitor.
  + A reservation made and the user pays for missing product that it won't get.

### Use case: **Purchase cart - Missing product**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. A purchase cart action has been started.
    2. A missing product has been found.
  + **Parameter:** product
  + **Actions:**
    1. System checks the availability of each product in each store basket in the shopping cart.
    2. System update products quantity in each store.
    3. If missing product:
       1. System restores the update of all products that where on the order.
       2. System cancels the order and sends message to the user.
    4. Else:
       1. System confirm the reservation and continue.
  + **Post conditions:**
    1. Products quantities updates according to missing products.
    2. User got message on failed order if needed.
    3. If needed- no reservation is being made.

Acceptance test:

* **Happy- Success – no missing products.**
  + System checks the availability of each product in each store basket in the shopping cart.
  + System updates quantities of the products in each store.
  + There are no missing products.
  + System confirm the reservation and continue.
* **Sad- Fail missing product** 
  + System checks the availability of each product in each store basket in the shopping cart.
  + System updates quantities of the products in each store.
  + One product is missing.
  + System cancels the update of all the products in the order.
  + System sends error message to the user and cancel the order.
* **Bad- Fail missing product – System doesn’t restore quantities.**
  + System check the availability of each product in each store basket in the shopping cart.
  + System updates quantities of the products in each store.
  + One product is missing.
  + System sends error message to the user and cancel the order.
  + Quantities in the stores are wrong.

### Use case: **purchase cart- check policies**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. A purchase cart action has been started.
    2. System confirmed the reservation.
  + **Parameter:**
  + **Actions:**
    1. System checks for products which require special policies.
    2. If there are such products ->
       1. System checks all rules of the policies.
       2. If fail in rule check ->
          - System cancel the reservation.
          - System restore quantities of all products.
          - System send error message to the user.
       3. Else –>
          - System confirm the policy.
    3. Else -> System confirm the policy.
    4. System calculate the purchase according to the discount policy of the store
    5. System show the user the total relevant price of the order and ask him to confirm.
  + **Post conditions:**
    1. Total price has been calculated.
    2. System checked for illegal purchases.
    3. All relevant discounts has been calculated.
    4. User got total price to approve.

Acceptance test:

* **Happy- Success policies check**
  + System checks for products which require special policy.
  + There are such products.
  + System checks the policy of the products.
  + Success in policy.
  + System confirm the buy policy.
  + System calculate the price of the shopping cart according to the discount and buying policy of the store.
  + System show the user the total relevant price of the order and ask him to confirm.
  + User confirm the reservation and continue.
* **Sad- Fail policies check – user buy alcoholic products and below 18- cancel order**
  + System checks for products which require special policy.
  + There are such products.
  + System checks the policy of the products.
  + Fail in policy.
  + System cancel the reservation.
  + System restore quantities of all products.
  + System send error message to the user.
* **Bad- Fail policies check – user buy alcoholic products and below 18- keep order**
  + System checks for products which require special policy.
  + There are such products.
  + System checks the policy of the products.
  + Fail in policy.
  + System confirm the buy policy.
  + System calculate the price of the shopping cart according to the discount and buying policy of the store.
  + System show the user the total relevant price of the order and ask him to confirm.
  + User confirm the reservation and continue.
  + A kid bought alcohol.

### Use case: **purchase cart- check Discounts**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. A purchase cart action has been started.
    2. System confirmed the reservation.
  + **Parameter:**
  + **Actions:**
    1. System checks for products which require special discounts.
    2. If there are such products ->
       1. System checks the conditions of the discounts.
       2. If stand in the conditions:
          - System apply the discounts according to the discounts rules.
          - System updates the products price.
       3. If fail in conditions ->
          - System doesn't update the products price.
    3. System calculate the price according to the discount policy of the store
    4. System show the user the total relevant price of the order and ask him to confirm.
  + **Post conditions:**
    1. Total price has been calculated.
    2. System checked for illegal purchases.
    3. All relevant discounts has been calculated.
    4. User got total price to approve.

Acceptance test:

* **Happy- Success policies check**
  + System checks for products which require special discount.
  + There are such products.
  + System checks the discount rules of the products.
  + Success in rules.
  + System update the products price according to the discount.
  + System calculate the price of the shopping cart according to the new prices.
  + System show the user the total relevant price of the order and ask him to confirm.
  + User confirm the reservation and continue.
* **Sad- Fail policies check – user buy alcoholic products and below 18- cancel order**
  + System checks for products which require special discount.
  + There are such products.
  + System checks the discount rules of the products.
  + Fail in rules.
  + System doesn’t update the products prices.
  + System calculate the price of the shopping cart according to the new prices.
  + System show the user the total relevant price of the order and ask him to confirm.
  + User confirm the reservation and continue.
* **Bad- Fail policies check – user buy alcoholic products and below 18- keep order**
  + System checks for products which require special discount.
  + There are such products.
  + System checks the discount rules of the products.
  + Fail in rules.
  + System update the products price according to the discount.
  + System calculate the price of the shopping cart according to the new prices.
  + System show the user the total relevant price of the order and ask him to confirm.
  + User confirm the reservation and continue.
  + The user pay less than he need although he doesn’t stand the rules.

### Use case: **purchase cart - payment**

* + **Actor:** subscribed visitor
  + **Precondition:**
    1. System approved the reservation.
    2. There is an external payment system.
    3. System approved the buying policy of the reservation.
    4. System calculated the total price of the reservation according to all the discounts.
  + **Parameter:** user's payment details, total price of the reservation.
  + **Actions:**
    1. Subscribed visitor enter paying method and payment details.
    2. Subscribed visitor approve payment details.
    3. System perform the payment using outside payment system.
    4. If payment fail ->
       1. System restores old products details.
       2. System sends error message to the user with the problem reason.
       3. System cancel the reservation.
       4. System sends error message to the user.
    5. Else -> System gets payment receipt and sends it to the user.
    6. If payment succeed - Purchase is saved in the purchase history DB.
  + **Post conditions:**
    1. Payment has been made.
    2. If payment failed ->
       1. Products quantities restored.
       2. Reservation has been cancelled.
    3. User got relevant message with the details of the payment.
    4. If payment succeed - Purchase is saved in the purchase history DB.

Acceptance test:

* **Happy- Success payment**
  + Subscribed visitor enter paying method and payment details.
  + Subscribed visitor approve payment details.
  + System perform the payment using outside payment system.
  + Payment succeed.
  + System send receipt to the subscribed visitor.
  + Purchase is saved in the purchase history DB.
* **Sad- Fail payment – fail in the payment system.**
  + Subscribed visitor enter paying method and payment details.
  + Subscribed visitor approve payment details.
  + System perform the payment using outside payment system.
  + Payment failed.
  + System restores old products details.
  + System sends error message to the user with the problem reason.
  + System cancel the reservation.
  + System sends error message to the user.
* **Bad- Fail payment – fail in the payment system and reservation continues.**
  + Subscribed visitor enter paying method and payment details.
  + Subscribed visitor approve payment details.
  + System perform the payment using outside payment system.
  + Payment failed.
  + System send receipt to the subscribed visitor.
  + Purchase is saved in the purchase history DB.
  + Reservation has been made without payment.

### Use case**: *Log out***

* + **Actor:** logged in user
  + **Precondition:**
    1. User is Logged in.
  + **Parameter:**
  + **Actions:**
    1. User press log out.
    2. System mark the user as subscribed Visitor and delete it’s shopping cart from the DB.
  + **Post conditions:**
    1. User no long a subscribed user.
    2. User's shopping cart is deleted.

Acceptance test:

* **Happy- success log out.**
  + Register user with correct personal details.
  + User log in.
  + User press log out.
  + System mark the user as subscribed Visitor and delete it’s shopping cart from the DB
* **Bad- Fail log out – can’t handle log out.**
  + Register user with correct personal details.
  + User log in.
  + User press log out.
  + System can’t log out the user and crash.

### Use case**: *Open store***

* + **Actor:** logged in user
  + **Precondition:**
    1. User is Logged in.
    2. The store is not already open.
  + **Parameter:** store details
  + **Actions:**
    1. User press open store and enter needed store details.
    2. System checks if a store with those details exists in the market.
    3. If so -> the system show error message and doesn’t open the store.
    4. Else, new store is added to the market with all the details provided by the user.
    5. System mark the user as the store owner and gives it all the permissions accordingly.
    6. System send confirmation message to the user.
  + **Post conditions:**
    1. New store exist in the market system.
    2. Users can buy products from the new store.
    3. The user has owner permission in the new store.

Acceptance test:

* **Happy- Success open store.**
  + Register user with correct personal details.
  + Log in with correct personal details.
  + User press add store with new store details.
  + System checks if a store with those details exists in the market.
  + No store with those details exists.
  + new store is added to the market with all the details provided by the user.
  + System mark the user as the store owner and gives it all the permissions accordingly.
  + System send confirmation message to the user.
* **Sad- Fail open store – exists store.**
  + Register user with correct personal details.
  + Log in with correct personal details.
  + User press add store with old store details.
  + System checks if a store with those details exists in the market.
  + There is a store with those details exists.
  + the system show error message and doesn’t open the store.
* **Bad- Fail open store – can't handle exists store.**
  + Register user with correct personal details.
  + Log in with correct personal details.
  + User press add store with old store details.
  + System checks if a store with those details exists in the market.
  + There is a store with those details exists.
  + new store is added to the market with all the details provided by the user.
  + System mark the user as the store owner and gives it all the permissions accordingly.
  + System send confirmation message to the user.
  + Two identical stores are open in the market.

### Use case: **Edit product inventory**

* + **Actor:** shop manager
  + **Precondition:** has edit product inventory permission.
  + **Parameter:** product
  + **Actions:**
    1. User logged in as shop manager.
    2. User press edit product inventory.
    3. User press add/ remove product from inventory.
    4. System check if product can be add or remove.
    5. If no- show error message.
    6. If yes add or remove the product and show success message.
  + **Post conditions:**
    1. All changes has been made and visible to the user.

Acceptance test:

* Happy-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press edit product inventory.
  + User press add/ remove product from inventory.
  + System check if product can be add or remove.
  + The product can be added or remove.
  + System add or remove the product and show success message.
* Sad-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press edit product inventory.
  + User press add/ remove product from inventory.
  + System check if product can be add or remove.
  + The product can't be added or remove.
  + System show error message and doesn't change the inventory.
* Bad-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press edit product inventory.
  + User press remove product from inventory.
  + System check if product can be remove.
  + The product can't be remove.
  + System remove the product and cause error because there is no product to remove.

### Use case: **Edit discount or buy policy**

* + **Actor:** shop manager
  + **Precondition:** has edit discount and buy policy permission.
  + **Parameter:** relevant details.
  + **Actions:**
    1. User logged in as shop manager.
    2. User press edit discount or buy policy.
    3. System check if new policy exists.
    4. If so -> show error message and doesn’t perform the change.
    5. Else -> add or change the policy and show success message.
  + **Post conditions:**
    1. All changes has been made and visible to the user.

Acceptance test:

* Happy-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press discount or buy policy.
  + User enter new policy.
  + System check if policy exists.
  + The new policy does not exists.
  + System add new policy and show success message.
* Sad-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press discount or buy policy.
  + User enter new policy.
  + System check if policy exists.
  + The new policy does exists.
  + System doesn’t add new policy and show error message.
* Bad-
  + Register user with correct personal details.
  + Log in as store manager with correct personal details.
  + User press discount or buy policy.
  + User enter new policy.
  + System check if policy exists.
  + The new policy does exists.
  + System add new policy and show success message.
  + Two same policies exists and cause problems.

### Use case: **Get purchase history**

* + **Actor:** market manager
  + **Precondition:** logged in as market manager.
  + **Parameter:** store name/ id or buyer id.
  + **Actions:**
    1. User logged in as market manager.
    2. User press get purchase history and enter store or buyer id of which history is required.
    3. System checks if store or buyer exists.
    4. If not -> system show error message and doesn’t show history.
    5. Else -> system search for history of purchases of the buyer or in the store and return it to the user.
  + **Post conditions:**
    1. Relevant information is shown to the user.

Acceptance test:

* Happy-
  + Register user with correct personal details.
  + Log in as market manager with correct personal details.
  + User press get purchase history.
  + User enter store or buyer id.
  + System check if id exists.
  + Id exists.
  + System return result of purchase history.
* Sad-
  + Register user with correct personal details.
  + Log in as market manager with correct personal details.
  + User press get purchase history.
  + User enter store or buyer id.
  + System check if id exists.
  + Id doesn't exist.
  + System doesn’t return result history and show error message.
* Bad-
  + Register user with correct personal details.
  + Log in as market manager with correct personal details.
  + User press get purchase history.
  + User enter store or buyer id.
  + System check if id exists.
  + Id doesn't exist.
  + System search history for the specific Id, fail and crash.

# Store Owner

### Use case: **Inventory Management**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product details (ID, parameters to modify)

**- Actions:**

1.System displays current inventory management dashboard.

1. System suggests 3 options: "Add Product", "Remove Product", “Update Product Details”.
2. If user selects:
3. “Add Product“-> Use case "“Add Product“ .
4. “Remove Product“-> Use case "Remove Product“ .
5. “Update Product“-> Use case "Update Product“ .

- **Postconditions:**

1. All changes are accurately and reflected in the inventory system and visible to the Store owner.

**Acceptance test:**

* **Happy- Store owner succeeds to add new product to store.**
  + - * Store Owner logs in and navigates to the inventory management dashboard.
      * Chooses “Add Product” and enters valid product details.
      * Submit the information.
      * User enters new policy.
      * System checks if the details, adds the product, and displays a success message.
      * Verify that the product appears in the inventory list.
      * System shows success message.
* **Sad – Store owner tries to add a product with incomplete details.**
  + - * Store Owner tries to add a product with incomplete details (e.g. missing price)
      * System detects the error and, display an error message and ask from the user to enter again the details.
* **Bad- Store owner tries to remove product that doesn’t exists- crash the system.**
  + - * Store owner tries to remove a product.
      * Store owner enter product ID.
      * System verifies that there is product with this ID.
      * System finds that the product doesn’t exists and crash.

### Use case: **Add Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product details (ID, parameters to modify)

**- Actions:**

1. System prompts to enter all required product details (name, price, quantity...)
2. Store Owner enters all the details and submit.
3. System validates the entered details are okay.
4. If validation fails ->

System displays an error message and ask him to enter again.

1. If validation succeeds ->

System adds the product to the inventory (and DB), displays a success message.

- **Postconditions:**

1. All changes are accurately and reflected in the inventory system and visible to the Store owner.

**Acceptance test:**

* **Happy- Store owner succeeds to add new product to store.**
  + - * Chooses “Add Product” and enters valid product details.
      * Submit the information.
      * User enters new policy.
      * System checks if the details, adds the product, and displays a success message.
      * Verify that the product appears in the inventory list.
      * System shows success message.
* **Sad- Store owner tries to add a product with incomplete details.**
  + - * Store Owner tries to add a product with incomplete details (e.g. missing price)
      * System detects the error and display an error message and ask from the user to enter again the details.
* **Bad- Store owner tries to add product while DB is down.**
  + - * Store owner adds a product when DB is temporarily down.
      * System attempts to add the product but fails and crash.

### Use case: **Remove Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product details (ID, parameters to modify)

**- Actions:**

1. System prompts to enter the product ID .
2. Store Owner enters the product ID.
3. System checks if the product ID exists:
4. If yes ->

System displays the product name and ask for confirmation.

1. If its confirmed ->

System removes the product from inventory and display a success message.

1. If it isn’t confirmed ->

System goes back to enter product ID screen.

5. if no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All changes are accurately and reflected in the inventory system and visible to the Store owner.

**Acceptance test:**

* **Happy- Store owner successfully removes product with ID.**
  + - * Store Owner selects “Remove Product” and enters product’s ID .
      * System verifies the product ID exists.
      * System display confirmation message of the removal.
      * System removes the product and displays a success message.
      * Verify that the product no longer appears in the inventory list.
* **Sad- Store owner tries removes product with ID that doesn’t exists.**
  + - * Store Owner selects “Remove Product” and enters product’s ID.
      * System verifies the product ID exists.
      * System display error message (product isn’t exists) and ask the user to enter new details.
* **Bad- Store owner tries to remove product while DB is down.**
  + - * Store owner remove a product when DB is temporarily down.
      * System attempts to remove the product but fails and crash.

### Use case**: Update Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product details (ID, parameters to modify)

**- Actions:**

1. System prompts to enter the product ID.
2. Store Owner enters the product ID.
3. System checks if the product ID exists:
4. If yes ->

System displays the product name and ask for confirmation.

1. If its confirmed ->

System removes the product from inventory and display a success message.

1. If it isn’t confirmed ->

System goes back to enter product ID screen.

5. if no ->

System displays an error message and ask him to enter again.

.

- **Postconditions:**

1. All changes are accurately and reflected in the inventory system and visible to the Store owner.

**Acceptance test:**

* **Happy - Store owner successfully update a product in the store.**
  + - * Store owner selects “Update Product”, chooses a product, and modifies some details.
      * Submits the updated information.
      * System checks the update, applies them, and displays a success message.
      * Verify that the product no longer appears in the inventory list.
* **Sad- Store owner tries update product with ID that doesn’t exists.**
  + - * Store Owner selects “Update Product” and enters product’s ID and all the parameters needed to be change.
      * System verifies the product ID exists.
      * System display error message (product isn’t exists) and ask the user to enter new details.
* **Bad- Store owner tries to update product while DB is down.**
  + - * Store owner update a product when DB is temporarily down.
      * System attempts to remove the product but fails and crash.

### Use case**: Manage Purchase and Discount Policies on Products**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1.System checks if product exists.

2. if no ->

System displays an error message and ask him to enter again.

3. if yes ->

* System suggests 6 options: "Add Purchase Policy on Product”, “Add Discount Policy on Product”, “Remove Purchase Policy on Product”, “Remove Discount Policy on Product”, “Update Purchase Policy on Product”, “Update Discount Policy on Product.”

1. If user selects:
2. “Add Product“-> Use case "“Add Product“ .
3. “Remove Product“-> Use case "Remove Product“ .
4. “Update Product“-> Use case "Update Product“ .

1.System displays current inventory management dashboard.

1. If user selects:
2. Add Purchase Policy on Product” -> Use case “Add Purchase Policy on Product”.
3. “Add Discount Policy on Product” -> Use case “Add Discount Policy on Product”.
4. “Remove Purchase Policy on Product” -> Use case "Remove Purchase Policy on Product” .
5. “Remove Discount Policy on Product” -> Use case "Remove Discount Policy on Product” .
6. “Update Purchase Policy on Product” -> Use case "Update Purchase Policy on Product” .
7. “Update Discount Policy on Product” -> Use case "Update Discount Policy on Product” .

- **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add purchase policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System add the policy to the product, applies them, and displays a success message.
* **Sad - Store owner tries to add a purchase policy that doesn’t exists.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to add purchase policy to a product whose already has purchase policy – crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System attempts to add the policy, but product has already a purchase policy and therefore crash.

### Use case: **Add Purchase Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System adds to product’s purchase policy the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add purchase policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System add the policy to the product, applies them, and displays a success message.
* **Sad - Store owner tries to add a purchase policy that doesn’t exists.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to add purchase policy to a product whose already has purchase policy – crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System attempts to add the policy, but product has already a purchase policy and therefore crash.

### Use case: **Add Discount Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (discount (hidden/ transparent).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System adds to product’s discount policy the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add discount policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System add the policy to the product, applies them, and displays a success message.
* **Sad - Store owner tries to add a discount policy that doesn’t exists.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to add discount policy to a product whose already has purchase policy – crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System attempts to add the policy, but product has already a purchase policy and therefore crash.

### Use case: **Add Composite Policy**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.
  + At least 2 Discount policies exist for this store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (composite (hidden/ transparent).
2. System asks to choose policies to contain (min 2)
3. System asks to enter operator between policies.
4. Store Owner enters policy.
5. System checks if policy exists.
6. If yes ->

System adds to store the composite policy.

1. If no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

### Use case: **Remove Purchase Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System checks if the product has purchase policy and if so, remove it, displays a success message.

1. if no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully remove purchase policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Purchase Policy”
      * System verifies that the product has a purchase policy.
      * System removes the policy to the product and displays a success message.
* **Sad - Store owner tries to remove a purchase policy that doesn’t exists.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Purchase Policy”
      * System verifies that product has purchase policy but fails to find to product.
      * System displays an error message that product isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to remove purchase policy but fails to update DB– crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Purchase Policy”
      * System verifies that product exists with purchase policy.
      * System removes the policy from the product but fails to update the DB.

### Use case: **Remove Discount Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (discount (hidden/ transparent)).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System checks if the product has discount policy and if so, remove it, displays a success message.

1. if no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully remove discount policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Discount Policy”
      * System verifies that the product has a discount policy.
      * System removes the policy to the product and displays a success message.
* **Sad - Store owner tries to remove a purchase policy – product doesn’t exists.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Discount Policy”
      * System verifies that product has discount policy but fails to find to product.
      * System displays an error message that product isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to remove discount policy but fails to update DB– crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Remove Discount Policy”
      * System verifies that product exists with discount policy.
      * System removes the policy from the product but fails to update the DB.

### Use case: **Update Purchase Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System modifies the product’s purchase policy to the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully update purchase policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Purchase Policy”
      * System verifies that the product has a purchase policy.
      * System updates the policy to the product and displays a success message.
* **Sad - Store owner tries to update a purchase policy – product doesn’t exist.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Purchase Policy”
      * System verifies that product has purchase policy but fails to find to product.
      * System displays an error message that product isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to update purchase policy but fails to update DB– crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Purchase Policy”
      * System verifies that product exists with purchase policy.
      * System updates the policy from the product but fails to update the DB.

### Use case**: Update Discount Policy on Product**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Product ID

**- Actions:**

1. System asks to enter policy type discount (hidden/ transparent)).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System modifies the product’s discount policy to the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All changes in the product policy are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully update discount policy.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Discount Policy”
      * System verifies that the product has a discount policy.
      * System updates the policy to the product and displays a success message.
* **Sad - Store owner tries to update a discount policy – product doesn’t exist.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Discount Policy”
      * System verifies that product has discount policy but fails to find to product.
      * System displays an error message that product isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to update discount policy but fails to update DB– crash.**
  + - * Store owner logs in and navigates to the product management section.
      * Enters product ID.
      * Choose “Update Discount Policy”
      * System verifies that product exists with discount policy.
      * System updates the policy from the product but fails to update the DB.

### Use case**: Manage Purchase and Discount Policies on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1.System checks if category ID exists.

2. if no ->

System displays an error message and ask him to enter again.

3. if yes ->

* System suggests 6 options: "Add Purchase Policy on Category”, “Add Discount Policy on Category”, “Remove Purchase Policy on Category”, “Remove Discount Policy on Category”, “Update Purchase Policy on Category”, “Update Discount Policy on Category.”

1. If user selects:
2. Add Purchase Policy on Category” -> Use case “Add Purchase Policy on Category”.
3. “Add Discount Policy on Product” -> Use case “Add Discount Policy on Category”.
4. “Remove Purchase Policy on Category” -> Use case " Category Purchase Policy on Category” .
5. “Remove Discount Policy on Category” -> Use case "Remove Discount Policy on Category” .
6. “Update Purchase Policy on Category” -> Use case "Update Purchase Policy on Category” .
7. “Update Discount Policy on Category” -> Use case "Update Discount Policy on Category” .

- **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add purchase policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System adds the policy to all the product in that category, applies them, and displays a success message.
* **Sad - Store owner tries to add a discount policy – policy doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to remove discount policy but fails to update DB– crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Discount Policy”
      * System verifies that product exists with discount policy.
      * System removes the policy from all the products in that category but fails to update the DB.

### Use case: **Add Purchase Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System adds to each product in that category product’s purchase policy the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

-**Postconditions:**

* + - * 1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add purchase policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System adds the policy to all the product in that category, applies them, and displays a success message.
* **Sad - Store owner tries to add a purchase policy – policy doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to add purchase policy but one of product in category has already purchase policy – fails and crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Purchase Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System attempts to add the policy to all the product in that category, but product some product there is already a purchase policy and therefore cause problems.

### Use case: **Add Discount Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type (discount (hidden/ transparent).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System adds to each product in that category product’s discount policy the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully add discount policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System adds the policy to all the product in that category, applies them, and displays a success message.
* **Sad - Store owner tries to add a discount policy – policy doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists but fails to find to policy.
      * System displays an error message that policy isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to add discount policy but one of product in category has already purchase policy – fails and crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Add Discount Policy”
      * Enters a policy type.
      * System verifies that the policy exists.
      * Submits the policy addition request.
      * System attempts to add the policy to all the product in that category, but product some product there is already a discount policy and therefore cause problems.

### Use case: **Remove Purchase Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System checks if any product in this category has purchase policy and if so, remove it, eventually displays a success message.

1. if no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully removes purchase policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Purchase Policy”
      * System verifies that the product has a purchase policy.
      * System removes the policy for all the products in that category and displays a success message.
* **Sad - Store owner tries to remove a purchase policy – product doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Purchase Policy”
      * System verifies that product has purchase policy but fails to find to product.
      * System displays an error message that category isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to remove purchase policy but fails to update DB –crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Purchase Policy”
      * System verifies that product exists with purchase policy.
      * System removes the policy from all the products in that category but fails to update the DB.

### Use case: **Remove Discount Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type (discount (hidden/ transparent)).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System checks if any product in this category has discount policy and if so, remove it, eventually displays a success message.

1. if no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully removes discount policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Discount Policy”
      * System verifies that the product has a discount policy.
      * System removes the policy for all the products in that category and displays a success message.
* **Sad - Store owner tries to remove a discount policy – product doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Discount Policy”
      * System verifies that product has discount policy but fails to find to product.
      * System displays an error message that category isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to remove discount policy but fails to update DB –crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Remove Discount Policy”
      * System verifies that product exists with discount policy.
      * System removes the policy from all the products in that category but fails to update the DB.

### Use case: **Update Purchase Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type (purchase type).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System modifies all in all the product in the category the purchase policy to the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

* **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully update purchase policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Purchase Policy”
      * System verifies that the category is exists.
      * System updates the policy to all the products in that category and displays a success message.
* **Sad - Store owner tries to update a purchase policy – category doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Purchase Policy”
      * System verifies that category exists.
      * System displays an error message that category isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to update purchase policy but fails to update DB –crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Purchase Policy”
      * System verifies that category exists with purchase policy.
      * System updates the policy for all the products in that category but fails to update the DB.

### Use case: **Update Discount Policy on Category**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.

- **Parameter:** Category ID

**- Actions:**

1. System asks to enter policy type discount (hidden/ transparent)).
2. Store Owner enters policy.
3. System checks if policy exists.
4. If yes ->

System modifies all in all the product in the category the discount policy to the given policy and displays a success message.

1. If no ->

System displays an error message and ask him to enter again.

- **Postconditions:**

1. All products in the category are updated to Store Owner’ modifications.

**Acceptance test:**

* **Happy - Store owner successfully update discount policy.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Discount Policy”
      * System verifies that the category is exists.
      * System updates the policy to all the products in that category and displays a success message.
* **Sad - Store owner tries to update a discount policy – category doesn’t exist.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Discount Policy”
      * System verifies that category exists.
      * System displays an error message that category isn’t exists and ask to enter again the details.
* **Bad - Store owner tries to update discount policy but fails to update DB –crash.**
  + - * Store owner logs in and navigates to the category management section.
      * Enters category ID.
      * Choose “Update Discount Policy”
      * System verifies that category exists with discount policy.
      * System updates the policy for all the products in that category but fails to update the DB.

### Use case: **Nomination of a Store Owner**

- **Actor:** Current Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.
  + The nominee is a register user and isn’t already store owner.

- **Parameter:** Nominee details

**- Actions:**

1.System displays store main dashboard.

2. Current Store Owner access to the store owner nomination section.

3. System asks for details to submit on the nominee

4. Current Store Owner enters the details.

5. System validates that the nominee is a subscribe user and not already store owner and that there is no more than one user that made him a nominee.

6. if validation fails, system display error message with the problem that occurred.

7. if validation succeeded, system send the current store owner a notification, asking them to approve or reject the nomination.

8.Current Store Owner needs to decide whether to approve or reject.

9. if approve ->

* System updates the store ownership.
* System grants the new owner all rights associated with the store ownership and management, including policy rights.
* System sends both old and new owners confirmation message.

10. if rejected ->

* System sends the nominee message that his nomination is rejected by the owner.

- **Postconditions:**

1. If the nomination accepted, the nominee is listed as the new store owner beside the first owner.
2. The nominee has full ownership rights.
3. Store ownerships are updated.

**Acceptance test:**

* Happy-
  + - * Current store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner.
      * Current store owner approves the request.
      * System updates the store ownership, grants to the new store owner the new permission.
      * System displays a success message.
* Sad-
  + - * Current store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner, but fails (he owns another store)
      * System displays an error message.
* Bad-
  + - * Current store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner.
      * System can’t send a notification to the current store owner and therefore the process is stuck and crash.

### Use case: **Nomination of a Store Manager**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.
  + The nominee is a register user and isn’t already store owner or store manager.

- **Parameter:** Nominee details, desired management privileges.

**- Actions:**

1.System displays store main dashboard.

2. Store Owner access to the store manager nomination section.

3. System prompts details to submit on the nominee.

4. Store Owner enters details.

5. Store Owner selects the specific management privileges to grant.

6. System validates that the nominee is a register user and not already store owner or manager.

7.if validation fails, system display error message with the problem that occurred.

7. if validation succeeded, system send the store owner a notification, asking them to approve or reject the nomination of the new store manager (including details on the nominee and privileges).

8. Store Owner needs to decide whether to approve or reject.

9. if approve ->

* System updates the store management records.
* System grants the new manager the selected rights management privileges.
* System sends both store owner and new store manager confirmation.

10. If rejected ->

* System sends the nominee message that his nomination is rejected by the owner.
* **Postconditions:**

1. If the nomination accepted, the nominee is listed as the new store manager.
2. The nominee has selected privileges.
3. Store management records are updated.

**Acceptance test:**

* **Happy - Store owner successfully nominate a member to be store owner.**
  + - * Store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner.
      * Store owner approves the request.
      * System updates the store ownership, grants to the new store owner the new permission.
      * System displays a success message.
* **Sad - Store owner tries to nominate a store owner to be store owner – fail.**
  + - * Store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner, but fails (he owns another store)
      * System displays an error message.
* **Happy - Store owner tries to nominate a member to be store owner but system doesn’t send notification to store owner to confirm.** 
  + - * Store owner logs in and navigates to the nomination section.
      * Enters nominee details.
      * System verifies that nominee isn’t already a store owner.
      * System can’t send a notification to the store owner and therefore the process is stuck and crash.

### Use case: **Modifying Store Manager Privileges**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store.
  + The store manager whose privileges are being adjusted is already a store manager with management privileges.
  + The store manager whose privileges are being adjusted was nominated by the Store Owner.

- **Parameter:** Store manager ID, new privileges

**- Actions:**

1.System displays store main dashboard.

2. Store owner selects the section of managing store manager privileges.

3. System displays the store manager privileges/

4. Store Owner modifies the privileges to the given privileges.

5. System validates the change.

6. System send message to the store manager that his privileges have been modify.

**-Postconditions:**

1. Store manager privileges has change.

**Acceptance test:**

* **Happy - Store owner successfully modifying another store owner’s permissions.**
  + - * Store owner logs in and navigates to the nomination section.
      * Enters store manager ID, new privileges.
      * System verifies there is manager in the store with this ID and that new privileges are compatible.
      * System updates the manager’s privileges and displays a success message.
* **Sad- Store owner tries to modify another store owner’s permissions but the ID doesn’t fit to store owner.**
  + - * Store owner logs in and navigates to the nomination section.
      * Enters store manager ID, new privileges.
      * System verifies there is manager in the store with this ID and that new privileges are compatible, but fails there isn’t manager with this ID.
      * System displays an error message.
* **Bad- Store owner tries to modify another although he doesn’t have the permission to do that.**
  + - * Store owner logs in and navigates to the nomination section.
      * Enters store manager ID, new privileges.
      * System verifies there is manager in the store with this ID and that new privileges are compatible.
      * System didn’t find out that the manager is manager of different store but continue the process.

### Use case: **Closing a Store**

- **Actor:** First Store Owner (founder)

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has ownership permissions for the store and he’s the first owner.

- **Parameter:** Store ID

**- Actions:**

1.System displays store main dashboard.

2. Store owner selects the option close the store.

3. System prompts a confirmation message to close the store.

4. If Store Owner confirmed ->

a. System changes the status of the store to “inactive”.

b. System updates the active stores (“hide” the products in this store)

Store Owner modifies the privileges to the given privileges.

5. System sends to all the owners and managers of this store a notification that the store is closed until farther notice.

**-Postconditions:**

1. The store is now “inactive” in DB.

2. Users can no longer see information about product in the store.

3. The store’s product are excluded from appearing in the market’s product search result.

**Acceptance test:**

* **Happy - Founder successfully close a store.**
  + - * Founder is logged in.
      * User enters store ID.
      * System verifies that there is active store with this ID.
      * System displays a confirmation message that the store will be “inactive.”
      * System updates the active stores and send notification to the store owners and managers and eventually displays a success message.
* **Sad- Founder tries to close a store- store is already deactivate.**
  + - * Founder is logged in.
      * User enters store ID.
      * System verifies that there is active store with this ID but fails because the store with this ID is already “inactive”.
      * System displays an error message.
* **Bad- Founder successfully close a store but system continue to display store’s products in system.**
  + - * Founder is logged in.
      * User enters store ID.
      * System verifies that there is active store with this ID.
      * System displays a confirmation message that the store will be “inactive.”
      * System updates the active stores and send notification to the store owners and managers but doesn’t hide the project in the store from the search system and allows buyer see the product in the “inactive” store.

### Use case: **Request Information about Positions in the Store**

- **Actor:** Store Owner

- **Precondition:**

* + Store Owner is logged in.
  + Store Owner has the necessary permissions to access to managerial information.

- **Parameter:** Store ID

**- Actions:**

1.System displays store main dashboard.

2. Store owner selects the option managerial information.

3. System displays information about each employee of the store and shows the privileges of each manger.

**-Postconditions:**

**Acceptance test:**

* **Happy – Store Owner successfully gets information about positions in store.**
  + - * Store Owner is logged in.
      * User enters store ID.
      * System verifies that there is a store with this ID and user as necessary permissions.
      * System displays all managerial information within the store (store managers, owners, employees...)
* **Sad – Store Owner tries to get information about positions in store without permission to do so.**
  + - * Store Owner is logged in.
      * User enters store ID.
      * System verifies that there is a store with this ID and user as necessary permissions System displays an error message but fails the user doesn’t have the necessary permission.
      * System displays an error message.
* **Bad – Store Owner tries to get information about positions in store – DB is down.**
  + - * Store Owner is logged in.
      * User enters store ID.
      * System verifies that there is a store with this ID and user as necessary permissions System displays an error message but fails the user doesn’t have the necessary permission.
      * System tries to pull the data from DB, but DB is temporality down and therefore system crash.

# Trading System Manager

### Use case: **Access Purchase History in the Trading System**

- **Actor:** Administrator

- **Precondition:**

* + Administrator is logged in.

- **Parameter:**

**- Actions:**

1.System displays trading system main dashboard.

2. Administrator enters to purchase history.

3. System displays a list of all the Purchase history sorting by each store ID

**-Postconditions:**

**Acceptance test:**

* **Happy – Administrator successfully access purchase history.**
  + - * Administrator is logged in.
      * Administrator navigates to purchase history dashboard.
      * System verifies administrator logged in.
      * System displays of all the Purchase history sorting by each store ID .
* **Sad – Administrator tries access purchase history – administrator logout.**
  + - * Administrator is logged in.
      * Administrator navigates to purchase history dashboard.
      * System verifies administrator logged in but fails.
      * System displays an error message that user isn’t logged in.
* **Bad – Administrator tries access purchase history , no purchase made- system return empty page instead of message.**
  + - * Administrator is logged in.
      * Administrator navigates to purchase history dashboard.
      * System verifies administrator logged in.
      * No purchase made in the trading system, system tries to display the purchase but can’t find any and return empty page instead of message.