* Use case: **Register to system**
  + **Actor: Guest User**
  + **Precondition:** Guest is not already registered to the system
  + **Parameter:** Member information
  + **Actions:**
    1. **Guest** enters the system
    2. **System** presents the option to signup
    3. The **Guest** provides the **Member** **information**
    4. **System** returns indication that the user is now a **member**

|  |  |
| --- | --- |
| Action | Expected Result |
| Provide valid member information that isn't currently in the system. | A new member has been added to the system withthe provided member information. |
| Provide member information that is currently in the system. | Error message |
| Provide invalid member information. | Error message |

* Use case: **Login**
  + **Actor: Guest User**
  + **Precondition:** User is a member of the system.
  + **Parameter:** Email and password
  + **Actions:**
    1. **Guest** enters the system
    2. **System** presents the option to enter email address and password and to login
    3. **Guest User** enters his email address and password
    4. If provided **Guest User**'s information is in the system and his password is valid
       1. **System** identifies the **Guest User** as member
       2. **System** informs that the member successfully logged in

|  |  |
| --- | --- |
| Action | Expected Result |
| Provide valid login information that is currently in the system. | The user is being logged in the system. |
| Provide login information that isn't currently in the system. | Error message |
| Provide invalid login information. | Error message |

* Use case: **Gather information about store/product**
  + **Actor: User**
  + **Precondition:** User opened the system
  + **Parameter:** String (presenting a product or a store)
  + **Actions:**
    1. **User** provides the product search information of the **product/store** he wants information of
    2. **System** searches through all it's system database for:
       1. **Products** that its name matches the provided information
       2. **Stores** that its name matches the provided information.
    3. **System** presents all the matching stores and products.

|  |  |
| --- | --- |
| Action | Expected Result |
| User Provides information about product that is currently in the system. | The system shows the relevant products matching to the provided information |
| Provide information that isn't currently in the system. | Message that indicates there are no matching products. |
| User Provides information about a store that is currently in the system. | The system shows the relevant stores matching to the provided information. |

* Use case: **Search for products**
  + **Actor: User**
  + **Precondition:** There are products in the system
  + **Parameter:** String to search
  + **Actions:**
    1. **User** types in a string representing what he looks for
    2. **System** searches through all the **products** in the data base- by its name and description
    3. **System** presents all the best matching results in a list
    4. **User** (optional) chooses Parameters to filter the products listed (such as price, category and so on)
    5. **System** filters the products and presents the new results.

|  |  |
| --- | --- |
| Action | Expected Result |
| User enters valid string of product name to search in the system, and such product exists in the system | The system shows the relevant products matching to the provided string |
| User enters valid string of product name to search in the system, and no such product exists in the system | Message that indicates there are no matching products. |
| User enters invalid string | **Error message** |

* Use case: **Save products in a shopping cart**
  + **Actor: User**
  + **Precondition:** There are available products to buy
  + **Parameter:** Products
  + **Actions:**
    1. **User** can choose a **product** to buy
    2. If the **product** already exists in the shopping cart
       1. **System** updates the amount of the **product** appearances in the **shopping cart**
    3. Else
       1. **System** adds **item** representing the product to the **shopping cart**

|  |  |
| --- | --- |
| Action | Expected Result |
| User chooses product to buy | The system adds this product to the user's shopping cart. |
| User chooses product to buy and the owner just removed the product | The system shows message indicates that the product is no longer available |
| User enters a negative amount of the product. | The system shows message indicates that the amount is no valid. |

* Use case: **View shopping cart**
  + **Actor: User**
  + **Precondition:**
  + **Parameter:**
  + **Actions:**
    1. **User** asks to present the shopping cart
    2. **System** presets all the products in the shopping cart, their amount and total amount and price in the cart

|  |  |
| --- | --- |
| Action | Expected Result |
| User request to present shopping cart- when the shopping cart isn't empty. | The system presents the requested shopping cart. |
| User request to present shopping cart- when the shopping cart is empty. | The system presents the requested shopping cart. |
| User request to present shopping cart- when the shopping cart is empty. | The system shows an empty list of products and the total price will be 0. |

**\*\*think if more acceptance tests are needed (bad)**

* Use case: **Edit shopping cart-**
  + **Actor: User**
  + **Precondition:** Shopping cart not empty
  + **Parameter:** Product to update and chosen change to make
  + **Actions:**
    1. **System** presents the existing **shopping cart**
    2. **User** choose a **product** from the **cart**
    3. If the **user** choose changing the amount
       1. If the new requested amount is 0, the **product** I removed from the **cart**
       2. Else the amount is updated as requested

|  |  |
| --- | --- |
| Action | Expected Result |
| User adds a valid amount of a product (i.e less then what the Store is offering) | The system adds the required products and presents the requested shopping cart. |
| User reduces the amount of a product to a number above 0 | The system the reduces the amount in the cart as requested |
| User reduces the amount of a product to 0 | The system removes the product from the user’s cart |
| User reduces the amount of a product to a negative number | The system removes the product from the user’s cart |

* **Use case: Buy a product in an immediate sell**
  + **Actor: User**
  + **Precondition:** The product exists in the shopping cart
  + **Parameter:** Product to buy and amount
  + **Actions:**
    1. **User** choose a product from the shopping cart and the amount, and asks to buy it
    2. **System** checks if there is a discount to the purchase
       1. **Systems** checks with the **store** if the **product's** price with the requested amount has discount
          - Discount for amount
          - Hidden discount
       2. If there is a discount
          - The **system** informs the **user** and updates the purchase total price
    3. **System** request information about the shipping information
    4. If user is a **~~member~~** **User** and is logged in
       1. **System** asks the user rather to use the address exists in the **system** or to ship to a different address
    5. Else
       1. **System** asks for basic information about the **user**
       2. **System** asks for shipping information to ship the products to
    6. **System** asks for paying information from the user
    7. **User** fills in the paying information
    8. **System** checks validity of all the information
       1. If all is ok, **system** sends the request to the **store owner** and informs the **user**
          - **System** updates the **products** amount in the **store** accordingly and informs the user the purchase succeeded and shows him the purchase details
       2. If something went wrong, the purchase is cancelled and the status restores as before and shows to the user message accordingly.

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Relevant use case action |
| The product exists in the shopping cart and User choose a product and the amount and there is a discount for this product in the specific store. | The system reduces the price of the product and updated the total price accordingly. | 2 |
| The product exists in the shopping cart and User choose a product and the amount and the user is logged in | The system asks the user rather to use the exist address or use another | 5 |
| The product exists in the shopping cart and the User choose a product and the amount and the user isn't logged in | The system asks the user for an address to send the product. | 5 |
| The product exists in the shopping cart and User choose a product and the amount and the user enters a valid payment information(and all the previous information is valid also) | the **products** amount in the **store** have been updated accordingly and informs the user the purchase succeeded and shows him the purchase details | All |
| The product exists in the shopping cart and User choose a product and the amount and the user enters a non-valid payment information(or one of the previous information is non valid ) | the purchase has been cancelled and the status restores as before and message has been showed to the user accordingly. | All |
| The product doesn't exist in the shopping cart | The user has been informed the product isn't in the user's shopping cart. | All |

* Use case: **Buy a product by offering a price**
  + **Actor: User**
  + **Precondition:** The product exists in the shopping cart
  + **Parameter:** Product to buy, amount and offered price
  + **Actions:**
    1. **User** choose a product from the shopping cart and asks to buy it, by offering a price
    2. **System** sends the offer to the store owner
    3. **Store owner** checks the offer and decides to accept or not to accept the offer
       1. If the **store owner** accepts the offer
          - **System** informs the **user** that the offer was accepted
          - The **user** can use **Buy a product in an immediate sell** use case with the new offered price
       2. If the **store owner** does not accept the offer
          - **System** informs the **user** that his offer was declined
          - The **System** offers the **user** to start **Buy a product by offering a price** again

|  |  |
| --- | --- |
| Action | Expected Result |
| The product exists in the shopping cart, the user offer a price for this product and the store owner accepts the offer. | The user redirects to immediate sell option with the offered price. |
| The product exists in the shopping cart, the user offer a valid price for this product and the store owner rejects the offer. | The user informed that the offer was declined and user get the option to offer a price again. |
| The product exists in the shopping cart, the user offer a non valid price for this product and the store owner rejects the offer. | The user informed the offer price is not valid |
| The product doesn't exist in the shopping cart | The user informed the product isn't in the user's shopping cart. |

* Use case: **Buy a product by auction**
  + **Actor: User**
  + **Precondition:** The **product** exists in the **shopping cart**
  + **Parameter: Product** to buy, amount and offered price
  + **Actions:**
    1. **System** presents the bottom price to start from, the current highest price and last date to offer
    2. **User** offers a price to buy the product
    3. **System** checks if the price is valid
       1. If price valid and is higher than the highest price offered until now by any user
          - **System** updates the **product's** highest price
          - **System** informs all the other users that offered a price in the auction.
       2. If price is lower than the highest price the **user** is informed, and he can choose to start **Buy a product by auction** use case again
    4. **System** collects the user's offered prices
    5. **System** closes the option to offer prices on the last date
    6. If there is an offered price
       1. **System** informs the **user** that offered the highest price that he can buy the **product** with the offered price
       2. The user uses the **Buy a product in an immediate sell** use case to buy the **product** in the offered price
    7. Else
       1. The **product** is not being sold

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Relevant stage |
| The product exists in the shopping cart, the user offers a valid price and he is not the highest so far. | the user informed that his offer is not the highest and he got another option to offer an higher price. | 3 |
| The product exists in the shopping cart, the user offers a valid price and he is not the highest. | The product exists in the shopping cart, the user offers a valid price and he is not the highest. | All |
| The product doesn't exist in the shopping cart | The user informed the product isn't in the user's shopping cart. | All |
| The product exists in the shopping cart, the user offered an non valid price | The product exists in the shopping cart, the user offered an non valid price | All |

* Use case: **Buy a product by lottery purchase**
  + **Actor: User**
  + **Precondition:** The **product** exists in the **shopping cart**
  + **Parameter: Product** to buy, amount and number of "chances" to win
  + **Actions:**
    1. **System** presents the last date to buy chances to win
    2. **User** asks to buy the product
    3. **User** chooses amount of "chances" to buy (each "chance" at the same price?)
       1. If the "chances" price is lower than the total price of the **product**
          - **System** asks the user to purchase the "chances"

**User** use **Buy a product in an immediate sell** use case, which the **product** to buy is the "chances" he chose to buy and the total price of all the chances he wants to buy

* + - 1. Else
         * **System** informs the **user** that the amount is not ok
    1. If the total price of the **product** is reached by the total price of chances sold to all users combined
       1. The **system** performs the lottery, with the suitable amount of chances to each user according to what he bought
       2. The **system** informs the **user** that won the lottery and sends the **product** to be shipped to him according to the shipping information he used in the **Buy a product in an immediate sell** use case
       3. The **system** informs all the other **users** that they lost the lottery and will not win the product
    2. If the last date to buy the product is reached
       1. **System** informs all the **users** that the total chances price, bought by users, have not reached the product's price, and the lottery is cancelled
       2. The **system** refunds the money people paid using the information they used in option 3.a

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Relevant stage |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

* Use case: **Logout**
  + **Actor: ~~Member~~,User**
  + **Precondition: Member** is logged in to the **system**
  + **Parameter:**
  + **Actions:**
    1. **User** asks to logout from the **system**
    2. **System** changes **user** status from logged in **~~member~~** to **guest**
    3. **User** can keep use the **system** as a **~~user~~** guest(not logged in member)

**Acceptance Tests:**

1. **Action:** The user logging out from the system

**Expected Result:** The user status changed to guest status, and he can use the system as a guest

**\*\*think if more acceptance tests are needed (sad,bad)**

* Use case: **Open a store**
  + **Actor: ~~Member~~,User**
  + **Precondition: ~~Member~~** **User** is logged in to the **system**
  + **Parameter:** New **store** information
  + **Actions:**
    1. **User** asks to open a new store in the **system**
    2. **System** asks for information about the **store** from the **user**
    3. **User** provides information
    4. **System** adds the store to the **system**
       1. **System** signs the **user** as the founder of the **store**

**Acceptance Tests:**

1. **Action:** The user is logged to the system and provides valid information while opening the store

**Expected Result:** The store is added to the system with the provided information and the user is signed as co-founder of the store

2. **Action:** The user is logged to the system and provides non valid information while opening the store

**Expected Result:** Message indicates the information provided for creating a store is invalid showed to the user.

3. **Action:** The user is not logged to the system and provides valid information while opening the store

**Expected Result:** Message indicates that creating a store required the user to be logged in.

* Use case: **Review purchase history**
  + **Actor: ~~Member~~,User**
  + **Precondition: ~~Member~~** **User** is logged in to the **system**
  + **Parameter:**
  + **Actions:**
    1. **System** presents the option to view personal purchase history
    2. User asks to view the history
    3. System presents details about every previous the user performed in the past

**Acceptance Tests:**

1. **Action:** The user is logged to the system and asks to watch his personal purchase history

**Expected Result:** A list of all the user's previous purchases.

* Use case: **Add new product to store**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition:** 
    1. **~~Member~~** **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
    3. **Same product does not already exist in the store**
  + **Parameter:** Store identification and new product
  + **Actions:**
    1. Store owner and seller asks to add a new product to a store he owns
    2. The system asks for store identification
    3. Store owner and seller enters store id
    4. System asks for product's details
    5. **Store owner and seller** provides required details- product details
    6. System checks if such a program already exist in the system
    7. If the product with the same product id exists already in the stor
       1. System informs Store owner and seller that a product already exists
    8. Else
       1. System adds the product to the store's products resource
       2. System updates product's current stock to 0
       3. System informs the Store owner and seller that the product was added successfully

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and valid new product details

**Expected Result:**

* The system added the new product to the store
* The product can be found when searching for products in the store
* The product can be found when searching for products in all the system
* The product's stock is 0

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that already exists in the store

**Expected Result:** System informs the **Store owner and seller that the products he asks to add already exits in the store**

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists, and with invalid product details

**Expected Result:** System informs the **Store owner and seller** that the product's details are invalid

* Use case: **Remove product from store**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition: ~~Member~~**
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** Store identification and product id
  + **Actions:**
    1. Store owner and seller asks to remove a product from a store he owns
    2. The system asks for store identification
    3. Store owner and seller enters store id
    4. System asks for product's id
    5. **Store owner and seller** provides required details- product id
    6. System checks if such a program already exist in the system
    7. If the product with the same product id exists in the store
       1. System deletes product from the store resources
       2. System informs Store owner and seller that a product was deleted successfully from the store
    8. Else
       1. System informs the **Store owner and seller** that the product doesn’t exists in the store

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a product id that exists in the store

**Expected Result:**

* The system deleted the product from the store
* The product can't be found when searching for products in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists in the store

**Expected Result:** System informs the **Store owner and seller that the products he asks to add doesn't exists in the store**

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

* Use case: **Update product stock- add items**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition: ~~Member~~**
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** Store identification, product id, amount
  + **Actions:**
    1. Store owner and seller asks to add product items to store's stock
    2. The system asks for store identification, product id and amount
    3. Store owner and seller enters store id
    4. **Store owner and seller** provides required details- store id, product id, amount
    5. System checks if such a store exists in user's store repository
    6. If exists
       1. System checks if such product exists in the store
       2. If exists
          - System updates the product's current stock to the new amount
       3. Else
          - Store asks if the **Store owner and seller** wants to add the new product to the store
          - If **Store owner and seller** choose to add the product

**Store owner and seller** start **Add new product to store** use case

System updates product's stock to the new amount

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a product id that exists in the store, and a valid amount

**Expected Result:** The system updated the product's stock in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists in the store

**Expected Result:** System informs the **Store owner and seller** that the products he asks to add doesn't exists in the store

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides a store id of an existing store he owns and a product id that exits in that store, and an invalid amount

**Expected Result:** Message indicates that requested amount is invalid

* Use case: **Update product stock- subtract items**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition: ~~Member~~**
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** Store identification, product id, amount
  + **Actions:**
    1. Store owner and seller asks to subtract product items from store's stock
    2. The system asks for store identification, product id and amount
    3. **Store owner and seller** provides required details- store id, product id, amount
    4. System checks if such a store exists in user's store repository
    5. If exists
       1. System checks if such product exists in the store
       2. If exists
          - System updates the product's current stock to the new amount
       3. Else
          - System informs the **Store owner and seller** that such product doesn't exist in the system

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a product id that exists in the store, and a valid amount to subtract

**Expected Result:** The system updated the product's stock in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists in the store

**Expected Result:** System informs the **Store owner and seller** that the products he asks to add doesn't exists in the store

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides a store id of an existing store he owns and a product id that exits in that store, and an invalid amount

**Expected Result:** Message indicates that requested amount is invalid

* Use case: **Update existing product's details**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition: ~~Member~~**
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** Store identification, product id, product details
  + **Actions:**
    1. Store owner and seller asks to update product's details
    2. The system asks for store identification, product id and new details
    3. **Store owner and seller** provides required details- store id, product id, new details
    4. System checks if such a store exists in user's store repository
    5. If exists
       1. System checks if such product exists in the store
       2. If exists
          - System updates the product's details as required

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a product id that exists in the store, and a valid new details

**Expected Result:** The system updated the product's details

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists in the store

**Expected Result:** System informs the **Store owner and seller** that the products he asks to add doesn't exists in the store

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides a store id of an existing store he owns and a product id that exits in that store, and invalid new details

**Expected Result:** Message indicates that requested details are invalid

* Use case: **Add buying strategy to store's policy**
  + **Actor: ~~Member~~Store owner and seller**
  + **Precondition: ~~Member~~**
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** Store identification, product id, product details
  + **Actions:**
    1. Store owner and seller asks to update product's details
    2. The system asks for store identification, product id and new details
    3. **Store owner and seller** provides required details- store id, product id, new details
    4. System checks if such a store exists in user's store repository
    5. If exists
       1. System checks if such product exists in the store
       2. If exists
          - System updates the product's details as required

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a product id that exists in the store, and a valid new details

**Expected Result:** The system updated the product's details

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns, and of a product that doesn't exists in the store

**Expected Result:** System informs the **Store owner and seller** that the products he asks to add doesn't exists in the store

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides a store id of an existing store he owns and a product id that exits in that store, and invalid new details

**Expected Result:** Message indicates that requested details are invalid

* Use case: **Add buying strategy to store's policy**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id and buying strategy
  + **Actions:**
    1. **User asks to add buying strategy to store**
    2. **System requests the store id and strategy**
    3. **User provides required information**
    4. **If strategy already exists**
       1. **System doesn't change it**
    5. **Else**
       1. **System adds new strategy to store's policy**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a new buying strategy to store's policy

**Expected Result:** The system updated the store's policy

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown buying strategy

**Expected Result:** System informs the **Store owner and seller** that the buying strategy is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

* Use case: **Update buying strategy to store's policy**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id and buying strategy
  + **Actions:**
    1. **User asks to update buying strategy to store**
    2. **System requests the store id and strategy**
    3. **User provides required information**
    4. **System locates store and check if strategy exists in the store's policy**
    5. **System asks whether to delete or update existing strategy**
       1. **If user chooses to update**
          - **System update the strategy in the store with the new strategy provided**
       2. **If user chooses to delete**
          - **System removes strategy from store's policy**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an existing buying strategy in store's policy

**Expected Result:** The system updated the store's policy as required

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown buying strategy

**Expected Result:** System informs the **Store owner and seller** that the buying strategy is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a strategy that doesn't exist

**Expected Result:** System informs the **Store owner and seller** that the buying strategy doesn't exist and encourages him to use **Add new strategy to store's policy** use case

* Use case: **Add allowed discounts to store's policy**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id and discount type
  + **Actions:**
    1. **User asks to add allowed discount type to store**
    2. **System requests the store id and discount type**
    3. **User provides required information**
    4. **If strategy discount type exists**
       1. **System doesn't change it**
    5. **Else**
       1. **System adds new discount type to store's policy**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an a new **discount type** to store's policy

**Expected Result:** The system updated the store's policy

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **discount type**

**Expected Result:** System informs the **Store owner and seller** that the **discount type** is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

* Use case: **Update allowed discounts in store's policy**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id and discount type
  + **Actions:**
    1. **User asks to update** discount type **to store**
    2. **System requests the store id and** discount type
    3. **User provides required information**
    4. **System locates store and check if** discount type **exists in the store's policy**
    5. **System asks whether to delete or update existing** discount type
       1. **If user chooses to update**
          - **System update the** discount type **in the store's policy with the new** discount type **provided**
       2. **If user chooses to delete**
          - **System removes** discount type **from store's policy**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an existing discount type in store's policy

**Expected Result:** The system updated the store's policy as required

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown discount type

**Expected Result:** System informs the **Store owner and seller** that the discount type is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a discount type that doesn't exist

**Expected Result:** System informs the **Store owner and seller** that the discount type doesn't exist and encourages him to use **Add new discount type to store's policy** use case

* Use case: **View store's policy**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id and discount type
  + **Actions:**
    1. **User asks to view store's policy**
    2. **System asks for store's id**
    3. **User provides store id**
    4. **System presents store policy**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns

**Expected Result:** The system presents store's policy- buying strategies and discount types

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing store's policy is only allowed to logged in existing members

* Use case: **Add buying strategy to store's product**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id and buying strategy
  + **Actions:**
    1. **User asks to add buying strategy to product**
    2. **System requests the store id, buying strategy and product id**
    3. **User provides required information**
    4. **If store exists, buying strategy valid and product exists in store**
       1. **System check if requested buying strategy exists in store's policy**
       2. **If exits in policy - System adds buying strategy to product**
    5. **Else**
       1. **System informs user**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns, valid buying strategy that exists in store's policy and an id of a product exists in the store

**Expected Result:** The system add buying strategy to the product in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **buying strategy**

**Expected Result:** System informs the **Store owner and seller** that the **buying strategy** is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a known buying strategy and a product id that doesn’t exist in store

**Expected Result:** Message indicates that managing product doesn't exist in store

7. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a known buying strategy that doesn't exist in store's policy and a product id that exists in store

**Expected Result:** Message indicates that strategy isn't allowed in store

* Use case: **Update buying strategy to store's product**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id and buying strategy
  + **Actions:**
    1. **User asks to update buying strategy to product**
    2. **System requests the store id, buying strategy and product id**
    3. **User provides required information**
    4. **If store exists, buying strategy valid and exists and product exists in store**
       1. **System asks whether to delete or update**
          - **If update, System check if requested buying strategy exists in store's policy**

**If exits in policy- system update existing buying strategy of product in store**

* + - * + **If delete- System removes buying strategy from product in store**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns, buying strategy that exist for the product in the store and an id of a product exists in the store

**Expected Result:** The system update buying strategy to the product in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **buying strategy**

**Expected Result:** System informs the **Store owner and seller** that the **buying strategy** is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a known buying strategy and a product id that doesn’t exist in store

**Expected Result:** Message indicates that managing product doesn't exist in store

7. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a known buying strategy that doesn't exist in store's policy and a product id that exists in store

**Expected Result:** Message indicates that strategy isn't allowed in store

* Use case: **View product's buying strategies**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id
  + **Actions:**
    1. **User asks to view buying strategy of product in store**
    2. **System requests the store id and product id**
    3. **User provides required information**
    4. **If store exists and product exists in store**
       - * **System provides all exiting buying strategies of the product in the store**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an id of a product exists in the store

**Expected Result:** The system provides all existing buying strategies of the product in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a product id that doesn’t exist in the store

**Expected Result:** System informs the **Store owner and seller** that the **product doesn’t exist in the store**

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

* **Use case: Add discount to product**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id and discount
  + **Actions:**
    1. **User asks to add discounts to product**
    2. **System requests the store id, discount type and discount details and product id**
    3. **User provides required information**
    4. **If store exists, discount type and details valid and exists and product exists in store**
       1. **System checks if discount is valid in store's policy**
       2. **If valid- System adds discount to product in store**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns, valid discount type and details that valid with system in general and with store's policy, and an id of a product exists in the store

**Expected Result:** The system adds the discount to the product int the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **discount type**

**Expected Result:** System informs the **Store owner and seller** that the **discount type** is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a valid discount and a product id that doesn’t exist in store

**Expected Result:** Message indicates that managing product doesn't exist in store

7. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a valid discount that is not valid with store policy, and a product id that exists in store

**Expected Result:** Message indicates that discount isn't allowed in store

* **Use case: Update product's discount**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id and discount
  + **Actions:**
    1. **User asks to update discounts to product**
    2. **System requests the store id, discount type and discount details and product id**
    3. **User provides required information**
    4. **If store exists, discount type and details valid and exists and product exists in store**
       1. **System checks if discount is valid in store's policy**
       2. **If valid- System adds discount to product in store**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns, valid discount type and details that valid with system in general and with store's policy, and an id of a product exists in the store

**Expected Result:** The system adds the discount to the product int the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **discount type**

**Expected Result:** System informs the **Store owner and seller** that the **discount type** is unknown

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a valid discount and a product id that doesn’t exist in store

**Expected Result:** Message indicates that managing product doesn't exist in store

7. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a valid discount that is not valid with store policy, and a product id that exists in store

**Expected Result:** Message indicates that discount isn't allowed in store

* **Use case: View product's discount**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, product id
  + **Actions:**
    1. **User asks to view buying strategy of product in store**
    2. **System requests the store id and product id**
    3. **User provides required information**
    4. **If store exists and product exists in store**
       - * **System provides all exiting buying strategies of the product in the store**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns and an id of a product exists in the store

**Expected Result:** The system provides all existing buying strategies of the product in the store

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a product id that doesn’t exist in the store

**Expected Result:** System informs the **Store owner and seller** that the **product doesn’t exist in the store**

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

* **Use case: Nominate user to be store manager**
  + **Actor: Store owner and seller**
  + **Precondition:** 
    1. **Store owner and seller** is logged in to the **system**
    2. **Store owner and seller** is an owner of an existing store
  + **Parameter:** store id, user id, permissions
  + **Actions:**
    1. **User asks to nominate an existing user to be manager of the store**
    2. **System requests the store id and user id**
    3. **User provides required information**
    4. **System locates store and user**
    5. **System asks for permission to add to user**
    6. **User provides the permissions**
    7. **If all valid**
       1. **System assigns requested user to be store manager**
       2. **System adds permissions to user**
       3. **System sets user permission as new manager**
       4. **System sets user to be nominated by the user that asked for this nomination**
       5. **Requested user informs that he got the new nomination and by who**

**Acceptance Tests:**

1. **Action: Store owner and seller** is logged to the system and provides an identification of an existing store that he owns, and id of a user of the system, that is not already a store manager

**Expected Result:** The system adds the user as manager of the store, updates his permissions and set his nominator to be the user that nominated him

2. **Action: Store owner and seller** is logged to the system and provides identification of a store that doesn't exist

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

3. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store that he doesn’t own

**Expected Result:** **System** informs **Store owner and seller** that the **store** doesn’t exist in his owned store repository

4. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and an unknown **user**

**Expected Result:** System informs the **Store owner and seller** that the **user is unknown to the system**

5. **Action:** The **Store owner and seller** is not logged to the system

**Expected Result:** Message indicates that managing stock is only allowed to logged in existing members

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and a known user that is already a manager of that store

**Expected Result:** Message indicates that user can't be nominated twice to be store manager of the same store

6. **Action:** The **Store owner and seller** is logged to the system and provides identification of a store he owns and doesn't have permissions to nominate new manager

**Expected Result:** Message indicates that user can't nominate new manager to store because of permissions issues.

* Use case: **Change management permission for sub-manger**
  + **Actor: User**
  + **Precondition:** The user is owner of the store the sub mangers set is manages and the set is not empty.
  + **Parameter:** Set of permissions
  + **Actions:**
    1. **User** chooses sub-manger from an non empty sub mangers set for specific store.
    2. The **Store's Owner** sets a set of permissions for the selected sub manager.
    3. The set of permissions is valid:

**3.1**.The sub manager gets the new permissions.

**4.**Else

**4.1-**The sub manager stays with the former permissions.

**Acceptance Tests:**

1. **Action:** The user is owner of the store the sub mangers set is manages and the sub mangers set is not empty the Store's owner selects sub manager and give him valid set of permissions, the sub manager does operations required each of the permissions from the set of permissions.

**Expected Result:** All the operations succeeded.

2. **Action:** The user is owner of the store the sub mangers set is manages and the sub mangers set is not empty the Store's owner selects sub manager and give him valid set of permissions, the sub manager does operation that required permission that doesn't belongs to the provided set of permissions

**Expected Result:** Message indicates the sub manger doesn’t have the permissions shown to the sub manager.

3. **Action:** The user is owner of the store the sub mangers set is manages and the sub mangers set is not empty the Store's owner selects sub manager and give him non valid set of permissions, the sub manager does operations required each of the permissions from the set of permissions.

**Expected Result:** Message indicates the set of permissions is not valid will be shown the Store's Owner.

* Use case: **User Remove management permission for sub-manger**
  + **Actor: User**
  + **Precondition:** The user owns the given store and sub manager is actual sub manager of the store.
  + **Parameter:** Store andsub manager.
  + **Actions:**
    1. The Store's Owner removes the sub manager from the managers set of the store

**2.**The right management permissions removed from the former sub manager permissions for the specific store

**Acceptance Tests:**

1. **Action:** The user owns the store and the Store's Owner removes sub manager's permission for specific store, the sub manager tries to do operation that requires management permission for the specific store

**Expected Result:** The operation aborted and the sub manager get appropriate message.

2. **Action:** The user owns the store and the Store's Owner removes management permission for sub manager which is not sub manager of the specific store

**Expected Result:** Message indicates that the sub manager is not manager of the specific store will be shown the Store's Owner.

3. **Action:** The user owns the store and the Store's Owner removes sub manager's permission for specific store, the sub manager tries to do operation that not requires management permission for the specific store

**Expected Result:** the operation succeeded and former sub manager get approval message for the operation.

* Use case: **User requests for purchase history for the store**
  + **Actor: User**
  + **Precondition:** The user owns the given store.
  + **Parameter:** Store
  + **Actions:**
    1. The User requests to print the history of purchase for specific range of dates
    2. If The User has ownership permissions for the specific store:

2.1 The user get the requested information

**2.**Else:

2.1: The user get a message for inappropriate permissions.

**Acceptance Tests:**

1. **Action:** The user owns the store and the Store's Owner management permission for the specific store, the owner changes price of a product that belongs to the purchase history

**Expected Result:** The history printed successfully with the right price of the changed product

2. **Action:** The user doesn't owns the store and the Store's Owner management permission for the specific store, the user changes price of a product that belongs to the purchase history

**Expected Result:** Message indicates that user is not owner of the specific store will be shown the Store's Owner.

1. **Action:** The user owns the store and the Store's Owner management permission for the specific store, the owner changes details of a product that belongs to the purchase history

**Expected Result:** The history printed successfully with the right price of the changed product

* Use case: **User requests for user history** 
  + **Actor: User**
  + **Precondition:** The user is manager of the ecommerce systemand the provided user belongs to the system
  + **Parameter:** user identifier
  + **Actions:**
    1. The user requests to print the history of provided user identifier for specific range of dates
    2. If The user has ecommerce management permissions:

2.1 The user get the requested information

**2.**Else:

2.1: The user get a message for inappropriate permissions.

**Acceptance Tests:**

1. **Action:** The user is manager of the ecommerce system the provided user identifier belongs to the system and user requests to print the history of provided user identifier for specific range of dates

**Expected Result:** The history printed successfully.

2. **Action:** The user is not manager of the ecommerce system the provided user identifier belongs to the system and the user requests to print the history of provided user identifier for specific range of dates

**Expected Result:** Message indicates that user is not manager of the ecommerce system be shown the user.

1. **Action:** The user is manager of the ecommerce system the provided user identifier doesn't belongs to the system and the user requests to print the history of provided user identifier for specific range of dates

**Expected Result:** Message indicates that provided user is not belongs to the ecommerce system be shown the user.

Dictionary:

|  |  |
| --- | --- |
| **Subject** | **Explain** |
| **Member information** | * ID * Email * Username * Password * Name * Date of birth * Address |
| **Valid** | No **Member** exists such that it has the same Email, Username or ID as the provided information. |
| **Login information** | * Username * Email * Password |
| **Valid** | A **Member** exists such that a provided the Username, Email and Password match with their **Member information**. |
| **Shipping information** | * Name * ID * Address |
| **System/Market System** | Represents everything that isn’t controlled by **users**. Responsible for transferring information, presenting options and so on |
| **User** | A person who is visiting the **System**. |
| **Registration/Sign-up** | A prerequisite for **Login**. A process in which a **Guest User** can provide **Valid Member Information** and request that the **System** will save saidMember Information. |
| **Login** | A process in which a **Member** can get access to additional functionalities such as selling **Products** or **founding a store.** Moreover, the **System** will save a Member’s **Shopping cart** even after **Logout**.  In order to log in a **User** must provide **Login information**. |
| **Logout** | The opposite of **Login**. A process in which a **Logged in Member** switches state to a **Guest** |
| **Founding a Store** | A process in which a **Logged in Seller** can create a new **Store** and become into a **Store Founder.** |
| **Appoint Store owner** | A process in which a **Store Owner** can appoint a **Member** (who isn’t already one of the store owners) to become one of the Store owners. |

|  |  |
| --- | --- |
| **Appoint Store Manager** | A process in which a **Store Owner** can appoint a **Member** (who isn’t already one of the Store Owners or **Store Managers**) to become one of the Store Managers. |
| **Guest** | A **user** labeled as Guest isn’t **logged in** to the system. That mean that it’s a **user** with only the base functionalities of the **system** such and buying **products**, **logging in** or **signing up** |
| **Member** | A member is a **user** who is **registered** to the **System** in the past. Upon **login**, a member will get access to more functionalities such as **founding** **a** **store** or **managing** a store. |
| **Buyer** | Any **User** that is not **logged in** as **Admin** |
| **Seller** | A **Member** who sells products |
| **Store Founder** | A **Seller** who **founded a** **store**. Has additional functionalities such as manage **inventory**, appoint additional **owners**, appoint **managers** or manage **store policies** |
| **Store Owner** | A **Seller** who owns a **store**. Has access to special functionalities such as manage **inventory**, appoint additional **owners**, appoint **managers** or manage **store policies**.  A store owner will always have a single appointer who appointed them to be an owner. |
| **Store manager** | A **Member** that was appointed by a **Store owner** to manage a **store**. Has additional functionalities based on the permissions given to him by his appointer |
| **Admin** | A **User** that manages the **System**. Has access to System wide functionalities such as viewing all **transactions** in the system |
| **Product** | An item of value that is sold in a **store**. Can be bought and sold. |
| **Basket** | A basket represents a group of **products** that came from the same **Seller or Store**. A basket contains products |
| **Shopping cart** | Each **user** has a shopping cart in which he can place his chosen **products** (whether digital or physical). A shopping cart is divided to **Baskets**. A user can view and edit his shopping cart. |
| **Transaction/Purchase information** | Represents a single sale between a **buyer** and at least one **Seller or Store**. Saves information about **products** andtheir amounts and prices |
| **Store** | A location (physical or otherwise) in which a **Seller** can sell **products**. |

|  |  |
| --- | --- |
| **Store policy** | The combination of a **Discount Policy** and a **Purchase Policy** for a specific **Store** |
| **Discount** | A reduction in the cost of a **Product** set by the **Seller** |
| **Visible Discount** | A **Discount** that is visible to all **Buyers.** Consist of a percentage and a duration, for the duration the **Product** will cost (percentage) less |
| **Conditional Discount** | A **Discount** that is visible to all **Buyers.** Consists of a percentage, duration and condition. For the duration, if the condition is met, the **Product** will cost (percentage) less. |
| **Hidden Discount** | A **Discount** that consists of a duration, percentage and a code. For the duration, during **Checkout** a **Buyer** can enter the code and the **Product** will cost (percentage) less |
| **Discount Policy** | Information the represents conditions and parameters under which a **Buyer** will get a **Discount** for a **Product**. |
| **Purchase Policy** | Represents information chosen by a **Seller** regarding which **Buyers** can buy **products** from this seller, what are the **Buying options** for each product and what are the conditions or parameters for **purchase**. |
| **Buying option** | Different options for **Purchasing** a **Product** set by a **Seller** |
| **Direct Purchase** | A **Buyer** must pay the listed price and **purchases** the **Product** immediately. |
| **Offer-Purchase** | A **Buyer** must offer a sum for the **Product.** Afterwards the **Seller** can accept the offer (in which case the Buyer must pay the sum to purchase the Product immediately similar to **Direct Purchase**), Decline or counter-offer (the buyer then must pay the counter offer sum in order to purchase the product). In this form a buyer must get the seller’s permission in order to purchase the product |
| **Auction Purchase** | Has a starting price and a duration. For the duration, a **Buyer** can offer a sum (higher than the current price) in which case his offer will become the current price. At the end of the duration the buyer with highest offer must pay the sum they offered and **Purchase** the **Product.** |

|  |  |
| --- | --- |
| **Lottery Purchase** | Has a Product-price and a duration. For the duration a **Buyer** can buy “chances” to win the product for any amount he wishes as long as the total amount from all the buyers does not exceed the product-price.  If the product-price is achieved, a buyer will be selected at random (based on the chances they bought) and they will get the **Product** otherwise if the duration ended and the target price was not achieved each buyer will get the money they payed back. |
| **Store Inventory** | A representation of the offered **products** in a specific **Store** |
| **Checkout** | A state in which a **Buyer** can choose to **Purchase** every **Product** in their **Shopping cart**. |
| **Purchase** | the process in which a **Buyer** pays for a **Product** sold by a **Seller** |
| **Enter/Visit the System** | The process of connecting to the System and becoming a **Guest User** (either by physically walking into a market, opening up a site online and so on) |
| **Product search information** | * Product name * Product ID * Product Category * Product’s keywords |
| **System database** | A representation of all the data in the **System** (could be physical like books or non-physical such as an online database) |
| **Total price** | Represents the total price a **Buyer** must pay for all the **Products** in their **Shopping Cart** |
| **Payment information** | * Credit card number * Expiration date * ID * Name * Security code |
| **User Status** | The current state of a **User** (**Guest**, **Buyer**, **Admin** and so on) |
| **Personal Purchase/Transaction history** | A sub-set of all the **Transactions** in the System. In this case those are all transactions in which a specific **User** was the **Buyer** |
| **Sub-Manager** | A **Store Manager** that has an **appointer**. The Appointee is the Sub-Manager of the Appointer |
| **Store Purchase/Transaction History** | A sub-set of all the **Transactions** in the System. In this case those are all transactions in which a specific **Store** was the **Seller** |