Table of Contents

System Scenarios

Use-Case: Real-Time Notification

1. **Actor:** System\Market, User, Close shop
2. **Precondition:** The receivers of the notification must be online
3. **Parameters:** text of message.
4. **Actions:**
    1. The user or Market selects the receivers of the notification which can be either users or shops.

        Note: In some cases, the system will initiate the process.
    2. In case the user selected a shop, the notification will be sent to all the relevant users associated with the shop (Shop founder, owners, relevant managers).
    3. The System checks if the selected users are logged into the system.
    4. The system will immediately present the notification and its context to the user (receiving).

Use-Case	*Parameter*	*Expected Output*
Real-Time Notification	A User sends a notification to a logged in user. The second user receives the notification and its displayed.	Success
	A User sends a notification to a shop. All the shop's relevant logged in officials receive the notification and it's displayed to them.	Success
	A User sends a notification to a user who is not logged in. The second user does receive the notification.	Success
	A User sends a notification with no text to a logged in user. The second user does not receive the notification and the system displayed an error message.	Success
	A User sends a notification to a logged in user. The second user does not receive the notification or its displayed.	Fail
	A User sends a notification to a logged in user. The notification was sent to an unknown third user.	Fail
	The user was able to send a notification with no text.	Fail
	The user sends a notification to a shop. It is not received by any of the shop's officials.	Fail

Use-Case: Supply

1. **Actor:** Shop, Market
2. **Parameters:** items to delivery, client's address.
3. **Pre-Condition:** payment was successful.
4. **Post-Condition:** Supplier will send the delivery.
5. **Actions:**
    1. **Shop** verified that payment was received.
    2. **Shop** sends a request to the market to supply a delivery to a client address with his bought item.
    3. The **Market** search an available supplier to the delivery and transfer the request to him using the external services.
        a. If there are no available suppliers, the delivery request is failed and a notification to the user will be sent to try again later, the initial process of the order will be kept.
    4. The supplier answers that the delivery will be send to the client.
        a. If the supplier cannot deliver the delivery, he will return to the Market that the delivery was not created, and the Market will choose another supplier.
        b. If all the suppliers cannot deliver the requested items, the Market will return to the Shop that the request to create a delivery cannot be done, and a notification will be sent to the user to try again later, and the initial process of order will be kept.
    5. The **Market** update the **Shop** that the delivery was created.
    6. The **Shop** will send a notification to the **User.**

Use Case	*Parameter*	*Expected output*
Supply	Market with 0 supplier A delivery of at-least 1 item and an address	Fail
	Market with 1 available supplier A delivery of at-least 1 item and an address	Success
	Market with 2 unavailable suppliers A delivery of at-least 1 item and an address	Fail

Use-Case: Initializing Marketplace System

1. **Actor:** System Manager
2. **Preconditions:** System Integrity rules are fully implemented
3. **Parameters:** ExternalConnector
4. **Actions:**
    1. System requests for a registration of the first user meant to be the System Manager.
    2. Requests a username and password.
    3. System Creates a System Manager User.
    4. The user is recorded to the database.
    5. Invoke Use case: setup of connection to external services with the PaymentService type as the parameter.
    6. Invoke Use case: setup of connection to external services with the SupplierService type as the parameter.
    7. If connection succeeds, System marketplace is now open.
    8. Else Request the user to Initialize in a different time.

Use-Case	*Parameter*	*Expected Output*
Initializing Marketplace System	System successfully creates a system manager, and is connected to at least one payment service, and at least one supplies service	Success
	System couldn't connect to any payment service	Fail
	System couldn't connect to any supplier service	Fail
	System Manager wasn't created successfully	Fail

Use-Case: Setup of connection to external services

1. **Actor:** System Manager
2. **Preconditions:** System Integrity rules are fully implemented
3. **Parameters:** service type
4. **Actions:**
    1. System asks user to choose services to connect to and presents a pool of available services of the corresponding type.
    2. System Manager selects 1 service they want to connect to.
    3. System asks if user is done selecting services to connect to.
    4. If not done, jump to 1.
    5. Invoke "adding external services connection" using the services to connect to as the parameter.
    6. System notifies which all connections were successful or not.

Use-Case	*Parameter*	*Expected Output*

setup of connection to external services	A non-empty collection of external services to connect to is sent to “adding external services”.	Success
	An empty collection of external services to connect to is sent to “adding external services”.	Fail

Use-Case: Setup of connection and disconnection from external services

1. **Actor:** System Manager
2. **Preconditions:** System Integrity rules are fully implemented
3. **Parameters:** service type
4. **Actions:**
    1. System asks user to choose services to connect to and presents a pool of available services of the corresponding type.
    2. System Manager selects 1 service they want to connect to.
    3. System asks if user is done selecting services to connect to.
    4. If not done, go to step 1.
    5. System asks user to choose services to disconnect from and presents a pool of currently connected services.
    6. SystemManager chooses 1 service he wants to disconnect from.
    7. System asks if user is done selecting services to disconnect from.
    8. If not done, go to step 5.
    9. Invoke “interchanging external services connection” using the services to connect to and services to disconnect from as the parameters.
    10. System notifies which all connections were successful or not.

Use-Case	*Parameter*	*Expected Output*
setup of connection and disconnection from external services	A non-empty collection of external services to connect to and A non-empty collection of external services to disconnect from was sent to “interchanging external services”.	Success
	An empty collection of external services to connect to is sent to “interchanging external services”.	Fail
	An empty collection of external services to disconnect from is sent to “interchanging external services”.	Fail

Use-Case: Adding external services connection

1. **Actor:** System Manager, changing connection with external services use case.
2. **Preconditions:** User is logged in and is System Manager, Minimum of one connection of each service
3. **Parameters:** collections of services to connect to
4. **Actions:**
    1. System iterates through services to connect to.
       for each service that isn't already connected:
       1.1. Attempt connection with the service.
       1.2. If successful, record the service in database.
       1.3. If failed, rollback the entire process.

Use-Case	*Parameter*	*Expected Output*
adding external services connection	All services to connect to are currently not connected. System manages to connect to all services from "services to connect to".	Success
	some services to connect to are currently connected.	Fail
	System can't connect to some services from "services to connect to".	Fail

Use-Case: Removing external services connection

1. **Actor:** System Manager, changing connection with external services use case.
2. **Preconditions:** User is logged in and is System Manager, Minimum of one connection of each service
3. **Post-Conditions:** Minimum of one connection of each service
4. **Parameters:** collections of services to remove
5. **Actions:**
    1. System checks that the number of connections to remove from each service is at most the number of active connections in service plus 1, to remain the system invariant.
    2. System iterates through services to remove
       for each service that already connected:
        a. Disconnects the connection with the service.
        b. If successful, remove the record of the service in database.
        c. If failed because the service doesn't exist, rollback the entire process.

Use-Case	*Parameter*	*Expected Output*

removing external services connection	All connections to disconnect to are currently connected. At least one connection of each service type will remain after the removal. System manages to disconnect all the connections from the list	Success
	some connections to disconnect to are currently disconnected.	Fail
	System can't disconnect from some services in the list.	Fail

Use-Case: Payment

1. **Actor:** User, Checkout use case.
2. **Preconditions:** User has at least 1 item in cart and is attempting to place an order,
3. **Parameters:** payment amount
4. **Actions:**
    1. System presents a pool of available payment methods.
    2. User selects one from of them from the pool.
    3. System sends a request to an external service to complete payment for the amount specified, via the payment method chosen.
    4. If payment completed successfully, record the payment in the database, and return the payment Id.
    5. If unsuccessful, signal back the transaction was unsuccessful.

Use-Case	*Parameter*	*Expected Output*
Payment	Payment is successful. A distinct transaction number is returned.	Success
	Payment is unsuccessful but transaction number still returns	Fail

General Guest-Visitor Scenarios

Use-Case: Enter the market.

1. **Actor:** Guest-User
2. **Precondition:** None.
3. **Parameters:** None.

4. **Actions:**
    1. The user requests the system to use its services.
    2. The system will create a new Guest-User object.

Use-Case: Leave the market

1. **Actor:** User
2. **Precondition:** None.
3. **Parameters:** None.
4. **Actions:**
    1. The user requests to leave the market.
    2. Terminates the object's session.

Use-Case: Login

1. **Actor:** guest-user
2. **Precondition:**
   The user is registered to the system.
3. **Parameters:** username, password.
4. **Actions:**
    1. System: system verifies that there is a match between the given password, to the user's password stored in the system.
        a. If there is a match, the login succeeds, and the guest-user is now logged-in and changes status to **member-user**.
        b. If the given password is wrong the use-case failed, and the system notifies the user.

Use-Case	*Parameter*	*Expected Output*
Login	Valid username (registered in system), password matches to the password in the system.	Success
	Username that doesn't exist in the system.	Fail
	Valid username (registered in the system), password mismatch.	Fail

Use-Case: Register

1. **Actor:** guest-user
2. **Precondition:**
   None-empty password.
3. **Parameters:** username, password.
4. **Actions:**
    1. System: validates given username doesn't already exist in the system.

(i) If the username doesn't already exist, the register succeeds, and the guest-user registered to be **member-user**.

(ii) If the username exists, the notifies the user and the user is requested to choose different username. Means, the user must repeat the process from the beginning.

Use-Case	*Parameter*	*Expected Output*
Register	Username that doesn't exist in the system. Contains valid username and password.	Success
	Username that already exist in the system.	Fail
	Username or password are not valid.	Fail

Guest-Visitor Purchase Scenarios

Use-Case: Get information about available shops

1**. Actor:** guest-user \ registered-user.

2.**Precondition**:

There is at least one registered shop \ product.
Registered-user succeed Login use-case.

3**. Parameters:** None

4**. Actions:**

1. User: asks to receive information about the relevant open shops in the market.
2. The Market returns list of all the names of the available open shops and their status as open shops in the market.

Use-Case	*Parameter*	*Expected Output*
Getting information about available shops	None	If all the shops in the list are with open status - success
	None	If the list contains a shop that its status is closed - fail

Use-Case: Get information about available products in specific shop

1**. Actor:** guest-user \ registered-user.

2.**Precondition**:

Shop with shop identifier given as parameter exists in market.
Registered-user succeed Login use-case.

3**. Parameters:** shop identifier

4**. Actions:**

1. User: asks to receive information about the products in shop with shop identifier as given.
2. System: class guest/member asks from the shop the information, and the shop returns list of all the available products, with their names and description.

Use-Case	*Parameter*	*Expected Output*
Getting information.	shopID, shop exists.	Success
	shopID, shop doesn't exist	Fail

Use-Case: Search product by name

1. **Actor:** guest-user / registered-user.
2. **Precondition:** none.
3. **Parameters:** product name
4. **Actions:**
    1. User: search a product by his name.
    2. System: returns all the products with name matches the parameter. (Returns a list in case there is more than one).

Use-Case	*Parameter*	*Expected Output*
Search products.	The user has received products that match the preliminary information.	Success
	- The user receives Irrelevant products to the given information. - Do not founds right product even its in the market.	Fail

Use-Case: Search product by keyword

1. **Actor:** guest-user / registered-user.
2. **Precondition:** none.
3. **Parameters:** some keyword
4. **Actions:**
    1. User: searches products by given keyword.
    2. System: returns all the products that their name contains that keyword. (possibly an empty list)

Use-Case	*Parameter*	*Expected Output*
Search products.	The user has received products that match the preliminary information.	Success
	- The user receives Irrelevant products to the given information. - Do not founds right product even its in the market.	Fail

Use-Case: Save products in a shopping cart

1.**Actor:** guest-user / registered-user.

2. **Precondition:** none.

3.**Parameters:**

The user has a product he wants to save.

4.**Actions:**

1. The user select the product.
2. The Market check if the product available with the relevant shop through the Shop Manager, by invoking Use-Case: check if the product available.
3. The Market insert the selected product to the user shopping Basket.

Use-Case	*Parameter*	*Expected Output*
Saving products in a shopping Basket.	The user received the product into his shopping basket.	Success
	- The user received the wrong product into his shopping basket. - The Market attach the product to another user. - The Market will not be able to attach the product to the shopping cart.	Fail

Use-Case: Editing the shopping cart

1.**Actor:** guest-user / registered-user.

2. **Precondition:** none.

3.**Parameters:**

The user has a product he wants to save.

4.**Actions:**

1. the user request the shopping basket.
2. He sends a request to the shopping basket to change the quantity of product.
3. the Market operate the **Use-Case**: check if the product available of the new quantity.
4. shopping basket update the right quantity of the product according to the answer of the use-case.

4.1 if the product is not available the Market does not update the amount.

Use-Case	*Parameter*	*Expected Output*
Editing the shopping cart.	- the product will not appear in the user cart. - appears with the new wanted quantity. - Market do not edit the quantity due to shop disapproval.	Success
	- the product will appear with quantity '0'. - wrong quantity / another product will be updated.	Fail

Use-Case: Check-out (buying)

1.**Actor:** User / Market.

2. **Precondition:** none.

3.**Parameters:**

The user has a product he wants to save.

4.**Actions:**

1.Market check for each product in the shopping basket if its available by invoking **Use-Case**: check if the product available.

2. Market calculate the total amount of the products for the available products by invoking **Use-Case**: calculate the total amount of the products in the shopping basket.

3. The user transmits payment information.

4. Market divides the payments by stores (products for each shop).

5. Market activate the **Use-case** (payment service) for each shop.

6. according to the use case result the Market sends the invoicing both the user and for the stores associated with the transaction.

6.1. if the transaction succeeded the market cleans the user shopping cart.

6.2. otherwise, Market notifies the user about it and returns the state of the market before performing the user case.

Use-Case	*Parameter*	*Expected Output*
Check-out (buying)	The user manages to make a payment for the products kept in the shopping cart, each store receives an invoice for its part in the transaction.	Success
	- The market is unable to make the payment - The stores do not receive notification of the transaction.	Fail

	- A payment is made that does not match the user's shopping cart.	

Use-Case: Calculate the total amount of the product in the shopping basket

1.**Actor:** Market.

2. **Precondition:** none.

3.**Parameters:**

The user has a product he wants to save.

4.**Actions:** **-** invoke this use-case for each product in the shopping basket.

1. Market check available of the product by invoking Use-Case: check if the product available.

1.2 in case the product is available Market remove the acquire quantity of the product from the shop by invoking change product's details **use-Case.**

1.3 otherwise, Market deletes the product from the shopping basket and inform the user about it by present the update shopping basket.

2. according to the answer of the use-case, calculate product discount by invoking Use-case: activate product discount.

4. – for each shop the Market check shop discount policy by invoking Use-case: activate shop discount policy.

Use-Case	*Parameter*	*Expected Output*
Calculate the total amount of the products in the shopping basket.	The market calculates the correct price for the shopping cart under the existing constraints (checking quantities, discounts and more).	Success
	Incorrect scheme of product prices.	Fail

Use-Case: Activate shop discount policy.

1.**Actor:** Market.

2. **Precondition:** invoking at Use-Case: calculate the total amount of the products in the shopping basket.

3.**Parameters:** product to buy.

The user has a product he wants to buy.

4.**Actions:**

1. Market is approaching the relevant shop.
2. Shop checking the discount policy.
3. Market checking if the current buying meets the conditions for a discount.
4. If there is a discount, the Market returns the current price of the shop's products.

Use-Case	*Parameter*	*Expected Output*
Activate shop discount policy.	The Market updates the price of the product not in accordance with the shop's discount policy.	Success
	- The market does not update the price of the products in accordance with the existing discount - The market updates the price of the products not in accordance with the existing discount.	Fail

Use-Case: Activate product discount

1.**Actor:** Market.

2. **Precondition:** invoking at Use-Case: calculate the total amount of the products in the shopping basket.

3.**Parameters:** product to buy.

The user has a product he wants to buy.

4.**Actions:**

1. Market is approaching the relevant shop.
2. Market checking if there is a discount for the current product.
3. If there is a discount, the market returns the current price of the product.

Use-Case	*Parameter*	*Expected Output*
Activate product discount	The market updates the price of the product according to the existing discount.	Success
	- The market does not update the price of the product in accordance with the existing discount. - The market updates the price of the product not in accordance with the existing discount.	Fail

Use-Case: Check if the product is available.

1.**Actor:** Market.
2. **Precondition:** user activate Use-Case: saving products in a shopping Basket.
3.**Parameters:**
Quantity of product, product id, shop
4.**Actions:**
1. The Market approaches to the relevant store through the ShopManager.
2. relevant shop check at the inventory that the quantity at the shop is ≥ quantity the user wants.
3. Market notifies if quantity of the product is available.

Use-Case	*Parameter*	*Expected Output*
Check if the product available.	The Market manages to get correct information for the required product.	Success
	The Market receives incorrect information for the required product.	Fail

Member-Visitor Scenarios

Use-Case: Logout

1. **Actor:** member user
2. **Precondition:**
   The user is registered and logged-in to the system.
3. **Parameters:** None.
4. **Actions:**
   1. System: The user is logged-out of the system and the system changes its status back to **guest-user**. User's shopping cart is saved.

Use-Case	*Parameter*	*Expected Output*
Logout	The user requesting to logout is logged-in (and registered).	Success
	The user requesting to logout isn't registered to system – username doesn't exist.	Fail
	The user requesting to logout isn't logged-In to the system.	Fail

Use-Case: Open Shop

1. **Actor:** member user
2. **Precondition:**
   The user is registered and logged-in to the system.
   The given shop name doesn't already exist in the system.
3. **Parameters:** shop's name
4. **Actions:**
   1. System:
      (i) If the shop's name doesn't exist in the system, creates new shop with shop's name. The system notifies the user the action succeeded and appoints the user as **shop-founder**. Also, the user is the first **shop-owner.**
      (ii) If the shop's name exists, the system notifies the user the action failed and he should repeat the process.

Use-Case	*Parameter*	*Expected Output*
Open Shop	Shop's name doesn't already exist in the system, user is registered and logged in.	Success
	User isn't registered to the system (user doesn't exist)	Fail
	User is registered but isn't logged in.	Fail
	User is registered and logged in, Shop's name already exists.	Fail
	User tries to add to shop invalid product (one that doesn't exist in the storage)	Fail
	User chooses invalid purchase policy for certain product. (Policy doesn't exist)	Fail

Shop-owner Scenarios

Use-Case: Add Item to Inventory

1. **Actor:** Shop Owner, Shop, Inventory
2. **Parameters:** Item, Quantity.
3. **Pre-Condition:** Shop owner logged in to his account.
4. **Post-Condition:** change the inventory.
5. **Actions:**
    1. The **Shop Owner** request the **Inventory** of his **Shop**.
    2. He sends a request to the **Inventory** to add the Item with the given Quantity.
        a. If the Item is already existing in the inventory, the Inventory will change the quantity of the Item.
        b. If the quantity is not a positive integer, then the Inventory will alert the user and don't add the Item.
    3. The **Item** is added to the **Inventory** with the given Quantity.

Use Case	*Parameter*	*Expected output*
Add Item to Inventory	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 2 with quantity 1	Success
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 2 with quantity 0	Failure
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 2 with quantity -1	Failure

	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 1 with quantity 1	Success
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 1 with quantity 0	Failure
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to add item 1 with quantity -1	Failure

Use-Case: Remove Item from Inventory

1. **Actor:** Shop Owner, Shop, Inventory
2. **Parameters:** Item.
3. **Pre-Condition:** Shop owner logged in to his account.
4. **Post-Condition:** change the inventory.
5. **Actions:**
    1. The **Shop Owner** request the **Inventory** of his **Shop**.
    2. He sends a request to the **Inventory** to remove the **Item**.
        a. If the **Item** does not exist in the **inventory** the inventory will alert the user.
    3. The **Item** is removed from the **Inventory**.

Use Case	*Parameter*	*Expected output*
Remove Item from Inventory	Shop with Inventory {Item: 1, Quantity: 3} Owner request to remove item 1	Success
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to remove item 2	Failure

Use-Case: Change Item's Detail

1. **Actor:** Shop Owner, Shop, Inventory, Item
2. **Parameters:** Item.
3. **Pre-Condition:** Shop owner logged in to his account.
4. **Post-Condition:** Item is Modify.
5. **Actions:**
    1. The **Shop Owner** request the **Inventory** of his **Shop**.
    2. He sends a request to the **Inventory** to change the detail of **Item**.
        a. If the Item does not exist in the inventory, the process will stop, and the Inventory will notify the user.
    3. The **Owner** chooses the detail to change.
        a. If the detail that the **Owner** has chosen is not legal the system will alert him and don't change them.

4. The **Item** modify the given details.

Use Case	*Parameter*	*Expected output*
Change Item Details	Shop with Inventory {Item: {ID:1, category: dairy}, Quantity: 3} Owner request to change the category of item 1 to bread	Success
	Shop with Inventory {Item: 1, Quantity: 3} Owner request to change the category of item 2	Failure

Use-Case: Change Buying Shop Policy

1. **Actor:** Shop Owner, Shop, Shop Buying Policy
2. **Parameters:** none.
3. **Pre-Condition:** Shop Owner logged in to his account.
4. **Post-Condition:** Shop Policy is modified.

Main-Scenario:

1. The **Shop Owner** request to change the shop buying policy from the **Shop.**
2. The **Shop Owner** view the current buying policy and decide which aspect.
3. The **Shop Policy** is modifying.

Alternative-Scenario:

1. If the new policy contradicts the consistency policy given to the founder the shop policy won't be modified and an alert will be given to the user.

Use-Case: Change Discount Shop Policy

Actor: Shop Owner, Shop, Shop Policy
Pre-Condition: Shop Owner logged in to his account.
Parameters: new discount policy value.
Main-Scenario:

1. The **Shop Owner** request to change the shop policy from the **Shop.**
2. The **Shop Owner** request to change the discount policy of the shop.
3. The **Shop Owner** send the new value of the policy.
4. The **Shop Policy** is modifying.

Alternative-Scenario:

1. If the new policy contradicts the consistency policy given to the founder the shop policy won't be modified and an alert will be given to the user.

Use-Case: Change Item's buying Shop Policy

Actor: Shop Owner, Shop, Shop Policy
Pre-Condition: Shop Owner logged in to his account.
Parameters: items id, new buying policy value.
Main-Scenario:

1. The **Shop Owner** request to change the shop policy from the **Shop.**
2. The **Shop Owner** request to change the buying policy of the items in the shop.
3. The **Shop Owner** send the new value of the policy and the items id.
4. The **Shop Policy** is modifying.

Alternative-Scenario:

1. If the new policy contradicts the consistency policy given to the founder the shop policy won't be modified and an alert will be given to the user.
2. If the item doesn't exist in the shop Inventory the Owner will be alerted.

Use-Case: Change Item's Discount Shop Policy

Actor: Shop Owner, Shop, Shop Policy
Pre-Condition: Shop Owner logged in to his account.
Parameters: items id, new discount policy value.
Main-Scenario:

1. The **Shop Owner** request to change the shop policy from the **Shop.**
2. The **Shop Owner** request to change the buying policy of the items in the shop.
3. The **Shop Owner** send the new value of the policy and the items id.
4. The **Shop Policy** is modifying.

Alternative-Scenario:

1. If the new policy contradicts the consistency policy given to the founder the shop policy won't be modified and an alert will be given to the user.
2. If the item doesn't exist in the shop Inventory the Owner will be alerted.

Use-Case: Appoint New Shop Owner

1. **Actor:** Shop Owner, Shop
2. **Parameters:** User ID to appoint.
3. **Pre-Condition:** Shop Owner logged in to his account.
4. **Post-Condition:** the shop has new shop owner.
5. **Actions:**
    1. The **Shop Owner** enter to his shop setting and choose to add new shop owner.
    2. **The Shop Owner** enter the **User** ID of the user he wants to appoint.
    3. The **System** will check that the User ID belong to a registered User.

   a. If the User ID doesn't belong to a registered User, the system will return a notification to the owner.
   b. If the User ID belong to an owner of another shop, the system will return a notification to the Shop Owner and the User ID won't be added to the Owners list.
   c. If the User ID belong to an owner of another shop, the system will return a notification to the Shop Owner and the User ID won't be added to the Owners list.
4. The User ID will be added to the Owners list of the shop.
5. The user will be notifying about that.

Use Case	*Parameter*	*Expected output*
Appoint New Shop Owner	Shop with Owners list [1,2] Owner 1 request to appoint user 3 to be owner	Success
	Shop with Owners list [1,2] Owner 1 request to appoint non-user 3 to be owner	Failure
	Shop with Owners list [1,2] Owner 1 request to appoint user 2 to be owner	Failure
	Shop with Owners list [1,2] Shop2 with Owners list [3] Owner 1 request to appoint user 3 to be owner.	Failure

Use-Case: Appoint New Shop Manager

1. **Actor:** Shop Owner, Shop
2. **Parameters:** User ID to appoint.
3. **Pre-Condition:** Shop Owner logged in to his account.
4. **Post-Condition:** the shop has new shop Manager.
5. **Actions:**
    1. The **Shop Owner** enter to his shop setting and choose to add new shop owner.
    2. The **Shop Owner** enter the **User** ID of the user he wants to appoint.
        a. If the User ID doesn't belong to a registered User, the system will return that to the Shop Owner.
        b. If the User ID belong to an owner or manager of another shop, the system will return that to the Shop Owner and the User ID won't be added to the mangers list.
        c. If the User ID already belong an Owner or a Manager of this Shop, the system won't add it again and alert the shop owner of such.
    3. The **System** will check that the User ID belong to a registered User.
    4. The User ID will be added to the Owners list of the shop.

Use Case	*Parameter*	*Expected output*
Appoint New Shop Manager	Shop with Owners list [1,2] and manager list [] Owner 1 request to appoint user 3 to be manager	Success
	Shop with Owners list [1,2] and manager list [] Owner 1 request to appoint user 2 to be manager	Failure
	Shop with Owners list [1,2] and manager list [] Owner 1 request to appoint non-user 3 to be manager	Failure
	Shop with Owners list [1,2] and manager list [3] Owner 1 request to appoint user 3 to be manager	Failure
	Shop with Owners list [1,2] and manager list [] Shop2 with Owners list [3] and manager list [] Owner 1 request to appoint user 3 to be manager	Failure
	Shop with Owners list [1,2] and manager list [] Shop2 with Owners list [3] and manager list [4] Owner 1 request to appoint user 4 to be manager	Failure

Use-Case: Change shop manager's permissions.

1. **Actor:** Shop Owner.
2. **Precondition:** Both the actor and the shop manger must be associated with the same store. Permissions must be valid.
3. **Parameters:** New permissions.
4. **Actions:**
    1. The Shop owner selects a manager which he'd like to change his permissions.
    2. The system checks if the Owner and Manager are both associated with the same shop.

3. The system will present the actor to permissions options he has to choose from.
4. The shop owner selects the new permissions for the shop manager.
    a. Real-Time Notification will start. The Shop Manager is notified his permissions were changed.
    b. In case Real-Time Notification failed: Delayed Notification will start.

Use-Case	*Parameter*	*Expected Output*
Change Shop Manager's Permissions	The changes the shop owner made for the shop manager's permissions are changed accordingly.	Success
	The shop owner removed all permissions from the shop manager. He is still a shop manager with no permissions.	Success
	The changes the shop owner made were applied to all the shop managers of the store.	Fail
	Owner made changes and the manager was not notified.	Fail

Use-Case: Close Shop

1. **Actor:** Shop Founder.
2. **Precondition:** The shop founder must be associated with the given shop. Shop must be open.
3. **Parameters:** none.
4. **Actions:**
    1. The shop founder selects the option to close the store.
    2. The system verifies the Founder is associated with the same shop.
    3. The system updates the shop's official that the shop is closed.
        1.1 Real-Time notification action will start.
        1.2 Delayed notification action will start **(not in version 0).**
    2. Expected result: The shop, it's information, products and any other information will be unavailable to all the users except for the shop's officials and the system managers.

Use-Case	*Parameter*	*Expected Output*
Close Shop	Shop founder requests to close his shop. The system does not display the shop, it's products and its information to other users except for the shop's officials and the system managers.	Success
	A user searches for a closed shop. The system does not display the shop.	Success
	A user search for a product which is sold by a closed shop. The system does not display the closed shop sells that product.	Success
	The system has successfully closed the shop, but the system managers and the shop's officials are not able to access the closed shop.	Fail
	A user searches for a closed shop. The system displays the shop.	Fail
	The products of the closed shop are displayed upon search.	Fail
	Upon closing the shop, the shop remains open.	Fail
	No notifications were sent to the store's officials upon closing the shop.	Fail

Use-Case: Request information on shop's officials

1. **Actor:** Shop Owner
2. **Precondition:** The shop owner and the officials must be associated with the same shop.
3. **Parameters:** Shop's officials.
4. **Actions:**
    1. The shop owner selects which official's information he'd like to see.

Expected result: The system will display the relevant information of the selected shop's officials.

a. Good Scenarios:
    i. The system returns the correct information of the selected officials.
    ii. The system returns nothing if no official was selected.
b. Bad Scenarios:
    i. The system returns sensitive information or information which was not supposed to be displayed (such as login ID or User's password). How to check??
    ii. The system does not display the selected official's information.

Use-Case	*Parameter*	*Expected Output*
Request Information on Shop's Officials	The system displays the correct information of the selected officials.	Success
	The system displays nothing if no official was selected.	Success
	The system displayed sensitive information or information which was not supposed to be displayed (such as login ID or User's password).	Fail
	The system does not display the selected official's information.	Fail

Use-Case: Request information of shop's sales history.

1. **Actor:** Shop Owner.
2. **Precondition:** The shop owner must be associated with the shop.
3. **Parameters:** Date, product, price (Also may be none, in which case will display all, or more filter options).
4. **Actions:**
    1. The shop owner requests the system to display the shop's sales history, filtered by given parameters or no parameters at all.
    2. The system validates the following parameters in case they were given:
        a. Checks if date is valid.
        b. Checks if product exists.
    3. The system returns the requested sales history to the user.

Use-Case	*Parameter*	*Expected Output*
Request Information	The system displays the shop's sales history accordingly to the given filter.	Success

of shop's sales history		
	The system displays sales of products which are not in the shop's inventory anymore when requested to.	Success
	The System deletes or ignores sales of products which are removed from the shop's inventory and will not display it upon request.	Fail
	The system does not display the requested filtered sales.	Fail
	The system displays all sales in all shops with similar products.	Fail

Trade-System Manager Scenarios

Use-Case: Shop purchase's history report

1. **Actor:** system manager
2. **Precondition:**
   The user performing the action is logged in and has system manager permissions.
   The requested shop exists in the trade system.
3. **Parameters:** shop's name
4. **Actions:**
   1. System: presents all the history of the shop purchases.

Use-Case	*Parameter*	*Expected Output*
Shop purchase's history report	The given parameter shop's name exists in the system as valid shop, the user performing the action is logged in and has system manager permissions.	Success
	The user requesting the action isn't logged in.	Fail
	The user requesting the action logged in and doesn't have system manager permissions.	Fail
	The given parameter shop's name doesn't exist in the system.	Fail

Use-Case: User purchase's history report

1. **Actor:** system manager
2. **Precondition:**
   The user performing the action is logged in and has system manager permissions.
   The requested user (for the report) registered in the trade system.

3. **Parameters:** username
4. **Actions:**
    1. System: presents all the history of the user purchases.

Use-Case	*Parameter*	*Expected Output*
User purchase's history report	The given parameter username exists in the system as registered user, the user performing the action is logged in and has system manager permissions.	Success
	The user requesting the action isn't logged in.	Fail
	The user requesting the action logged in and doesn’t have system manager permissions.	Fail
	The given parameter username doesn't exist in the system.	Fail