Table of Contents

[**System Scenarios** 3](#_Toc100829595)

[Use Case: Real-Time Notification 4](#_Toc100829596)

[Use-Case: Purchase Delivery 4](#_Toc100829597)

[Use-Case: Initializing Marketplace System 4](#_Toc100829598)

[Use-Case: Changing connection with external services 4](#_Toc100829599)

[Use-Case: Setup of connection to external services 4](#_Toc100829598)

[Use-Case: Setup of connection and disconnection from external services 4](#_Toc100829598)

[Use-Case: interchanging external services connection 4](#_Toc100829600)

[Use-Case: adding external services connection 4](#_Toc100829601)

[Use-Case: Payment 4](#_Toc100829602)

[**General Guest-Visitor Scenarios** 4](#_Toc100829603)

[Use-Case: Enter the market. 4](#_Toc100829604)

[Use-Case: Leave the market 4](#_Toc100829605)

[Use-Case: Login 4](#_Toc100829606)

[Use-Case: Register 4](#_Toc100829607)

[**Guest-Visitor Purchase Scenarios** 4](#_Toc100829608)

[Use-Case: getting information 4](#_Toc100829609)

[Use-Case: search Items 4](#_Toc100829610)

[Use-Case: Saving products in a shopping cart 4](#_Toc100829611)

[Use-Case: Editing the shopping cart 4](#_Toc100829612)

[Use-Case: check-out (buying) 4](#_Toc100829613)

[Use-Case: calculate the total amount of the product in the shopping basket 4](#_Toc100829614)

[Use-Case: activate shop discount policy. 4](#_Toc100829615)

[Use-Case: activate product discount 4](#_Toc100829616)

[Use-Case: check if the product is available. 4](#_Toc100829617)

[**Member-Visitor Scenarios** 4](#_Toc100829618)

[Use-Case: Logout 4](#_Toc100829619)

[Use-Case: Open Shop 4](#_Toc100829620)

[**Shop-owner Scenarios** 4](#_Toc100829621)

[Use-Case: Add Item to Inventory 4](#_Toc100829622)

[Use-Case: Remove Item from Inventory 4](#_Toc100829623)

[Use-Case: Change Item’s Detail 4](#_Toc100829624)

[Use-Case: Change Buying Shop Policy 4](#_Toc100829625)

[Use-Case: Change Discount Shop Policy 4](#_Toc100829626)

[Use-Case: Change Item’s buying Shop Policy 4](#_Toc100829627)

[Use-Case: Change Item’s Discount Shop Policy 4](#_Toc100829628)

[Use-Case: Appoint New Shop Owner 4](#_Toc100829629)

[Use-Case: Appoint New Shop Manager 4](#_Toc100829630)

[Use-Case: Change Manager Privileges 4](#_Toc100829631)

[Use Case: Change shop manager’s permissions. 4](#_Toc100829632)

[Use-Case: Close shop 4](#_Toc100829633)

[Use-Case: Request information on officials of the shop 4](#_Toc100829634)

[Use-Case: Request information of shop’s sales history. 4](#_Toc100829635)

[**Trade-System Manager Scenarios** 4](#_Toc100829636)

[Use-Case: Shop purchase's history report 4](#_Toc100829637)

[Use-Case: User purchase's history report 4](#_Toc100829638)

# **System Scenarios**

## Use Case: Real-Time Notification

1. **Actor:** User, Close shop,
2. **Precondition:** The receivers of the notification must be online
3. **Parameters:** text of message.
4. **Actions:**
5. The user selects the receivers of the notification which can be either users or shops.
6. 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).
7. The system will immediately present the notification to the user (receiving).

Expected result: The notification will pop up for the receivers of the notification to see and review.

* 1. Good Scenarios:
     1. A User sends a notification to a logged in user. The second user receives the notification and its displayed.
     2. 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.
     3. A User sends a notification to a user who is not logged in. The second user does receive the notification.
     4. 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.
  2. Bad Scenarios:
     1. A User sends a notification to a logged in user. The second user does not receive the notification or its displayed.
     2. A User sends a notification to a logged in user. The notification was sent to an unknown third user.
     3. The user was able to send a notification with no text.
     4. The user sends a notification to a shop. It is not received by any of the shop’s officials.

## Use-Case: Purchase Delivery

* + - 1. **Actor:** Shop, Market
      2. **Parameters:** items to delivery, client’s address.
      3. **Pre-Condition:** payment was successful.
      4. **Main-Scenario:**

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.
4. The supplier answers that the delivery will be send to the client.
5. The **Market** update the **Shop** that the delivery was created.
6. The **Shop** will send a notification to the **User.**

**Alternative-Scenario:**

1. If there is no available supplier, the **Market** will return to the **Shop** that the delivery cannot be deliver.
   1. 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.
   2. 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.

Usecase: Initializing Marketplace System

Actor: SystemManager

Preconditions: System Integrity rules are fully implemented

Parameters: ExternalConnector

Actions:

1. System requests for a registration of the first user meant to be the SystemManager.
2. Requests a username and password.
3. System Creates a SystemManager User.
4. The user is recorded to the database.
5. Invoke Usecase: setup of connection to external services with the PaymentService type as the parameter.
6. Invoke Usecase: 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.

Usecase: Changing connection with external services

Actor: System Manager

Preconditions: None

Postconditions: a minimum of one of each service type is connected.

Parameters: None

Actions:

1. System presents options of: “interchanging external services connection” and “adding external services connection”.
   1. If SystemManager chooses “interchanging external services connection”
   2. System presents type of services to be added – “supplier services”\”payment services”.
   3. Invoke Usecase: setup of connection and disconnection from external services with the selected service type as the parameter.
   4. If SystemManager chooses “adding external services connection”
   5. System presents type of services to be added – “supplier services”\”payment services”.
   6. SystemManager selects the service type.
   7. Invoke Usecase: setup of connection to external services with the selected service type as the parameter.

Usecase: setup of connection to external services

Actor: SystemManager

Preconditions: System Integrity rules are fully implemented

Parameters: service type

Actions:

1. System asks user to choose services to connect to and presents a pool of available services of the corresponding type.
2. SystemManager 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.

Usecase: setup of connection and disconnection from external services

Actor: SystemManager

Preconditions: System Integrity rules are fully implemented

Parameters: service type

Actions:

1. System asks user to choose services to connect to and presents a pool of available services of the corresponding type.
2. SystemManager 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.

Usecase: adding external services connection

Actor: SystemManager, Changing connection with external services usecase.

Preconditions: User is logged in and is SystemManager, Minimum of one connection of each service

Postconditions: Minimum of one connection of each service

Parameters: collections of services to connect to

Actions:

1. System iterates through services to connect to.

for each service that isn’t already connected:

* 1. Attempt connection with the service.
  2. If successful, record the service in database.
  3. If failed, rollback the entire process.

Usecase: interchanging external services connection

Actor: SystemManager, Changing connection with external services usecase.

Preconditions: Minimum of one connection of each service.

Postconditions: Minimum of one connection of each service.

Parameters: collections of services to connect to, collections of services to disconnect from.

Actions:

1. System checks that the services to connect to and services to disconnect from do not overlap.
2. System checks that the sum of **distinct** “services to connect to + service currently connected” minus the “Services to disconnect from” is at minimum 1.
3. If both conditions are true, System iterates through services to connect to.

for each service that isn’t already connected:

* 1. Attempt connection with the service.
  2. If successful, record the service in database.
  3. If failed, rollback the entire process.

1. If both conditions are true, System iterates through services to disconnect from.

for each service that is connected:

* 1. Attempt to disconnect from the service.
  2. If successful, delete the service in database.
  3. If failed, rollback the entire process.

Usecase: Payment

Actor: User, Checkout usecase.

Preconditions: User has at least 1 item in cart and is attempting to place an order,

Parameters: payment amount

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 |
| 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 |
| SystemManager wasn’t created successfully | Fail |
| 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 |
| 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 |
| Payment | Payment is successful. A distinct transaction number is returned. | Success |
|  | Payment is unsuccessful but transaction number still returns | Fail |
|  | All services to connect to are currently not connected.  System manages to connect to all services from “services to connect to”. | Success |
| adding external services connection | some services to connect to are currently connected. | Fail |
|  | System can’t connect to some services from “services to connect to”. | Fail |
| interchanging external services connection | All services to connect to are currently not connected, and all services to disconnect from are currently connected.  number of services to connect to + number of services connected > number of services to disconnect to.  System manages to connect to all services from “services to connect to” and manages to disconnect from all services from “services to disconnect from”. | Success |
|  | some services to connect to are currently connected, or some services to disconnect from aren’t currently connected. | Fail |
| System isn’t connected to some services from “services to connect to”. | Fail |
| System isn’t disconnected from all services from “services to disconnect from”. | Fail |
| number of services to connect to + number of services connected <= number of services to disconnect to. | Fail |

# **General Guest-Visitor Scenarios**

## Use-Case: Enter the market.

1. **Actor:** Guest-User
2. **Precondition:** None.
3. **Parameters:** None.
4. **Actions:**
5. The user requests the system to use its services.
6. 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:**
5. The user requests to leave the market.
6. Terminates the object’s session.

## Use-Case: Login

1. **Actor:** guest-user
2. **Precondition:**

The user is registered to the system.

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

## Use-Case: Register

1. **Actor:** guest-user
2. **Precondition:**

None-empty password.

1. **Parameters:** username, password.
2. **Actions:**
3. System: validates given username doesn’t already exist in the system.
4. If the username doesn't already exist, the register succeeds, and the guest-user registered to be **member-user**.
5. If the username exists, the system presents appropriate error message to the user and the user is requested to choose different username. Means, the user must repeat the use-case.

# **Guest-Visitor Purchase Scenarios**

## Use-Case: getting 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: getting 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 Items

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

2. **Precondition:** none.

3.**Parameters:** Identifying information for the wanted product.

The user has the wanted shop \ product.

4.**Actions:**

1. The user gives the relevant information for the desired product.

2. The Market returns the relevant products found.

|  |  |  |
| --- | --- | --- |
| 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: Saving 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 ShopManager, 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. * A payment is made that does not match the user's shopping cart. | Fail |

## 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.

1. **Parameters:** None.
2. **Actions:**
3. 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: 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.

1. **Parameters:** shop’s name
2. **Actions:**
3. System:
4. If the shop’s name doesn't exist in the system, creates new shop with shop’s name. The system presents an appropriate success message to the user and appoints the user as **shop-founder**. Also, the user is the first **shop-owner.**
5. If the shop’s name exists, the system presents appropriate error message to the user and the user is requested to choose different shop’s name.

# **Shop-owner Scenarios**

## Use-Case: Add Item to Inventory

**Actor:** Shop Owner, Shop, Inventory

**Pre-Condition:** Shop owner logged in to his account.

**Parameters:** Item, Quantity.

**Main-Scenario:**

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.
3. The **Item** is added to the **Inventory** with the given Quantity.

**Alternative-Scenario:**

1. If the **Item** is already existing in the **inventory**, the Inventory will change the quantity of the **Item**.
2. If the quantity is not a positive integer, then the **Inventory** will return an error massage and don’t add the **Item**.

## Use-Case: Remove Item from Inventory

**Actor:** Shop Owner, Shop, Inventory

**Pre-Condition:** Shop owner logged in to his account.

**Parameters:** Item.

**Main-Scenario:**

1. The **Shop Owner** request the **Inventory** of his **Shop**.
2. He sends a request to the **Inventory** to remove the **Item**.
3. The **Item** is removed from the **Inventory**.

**Alternative-Scenario:**

1. If the **Item** does not exist in the **inventory**, the Inventory will return a message alert as such.

## Use-Case: Change Item’s Detail

**Actor:** Shop Owner, Shop, Inventory, Item

**Pre-Condition:** Shop owner logged in to his account.

**Parameters:** Item.

**Main-Scenario:**

1. The **Shop Owner** request the **Inventory** of his **Shop**.
2. He sends a request to the **Inventory** to change the detail of **Item**.
3. The **Owner** chooses the detail to change.
4. The **Item** modify the given details.

**Alternative-Scenario:**

1. If the **Item** does not exist in the **inventory**, the Inventory will return a message alert as such.
2. If the detail that the **Owner** has chosen is not legal the system will alert him and don’t change them.

## Use-Case: Change Buying Shop Policy

**Actor:** Shop Owner, Shop, Shop Policy

**Pre-Condition:** Shop Owner logged in to his account.

**Parameters:** new buying policy values.

**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 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 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.

1. If the item doesn’t exist in the shop Inventory the Owner will be alerted.

## Use-Case: Appoint New Shop Owner

**Actor:** Shop Owner, Shop

**Pre-Condition:** Shop Owner logged in to his account.

**Parameters:** User ID to appoint.

**Main-Scenario:**

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.
4. The User ID will be added to the Owners list of the shop.
5. The user will be notifying about that.

**Alternative-Scenario:**

1. If the User ID doesn’t belong to a registered User, the system will return that to the Shop Owner.
2. If the User ID belong to an owner of another shop, the system will return that to the Shop Owner and the User ID won’t be added to the Owners list.
3. If the User ID already belong an Owner of this **Shop**, the system won’t add it again and alert the shop owner of such.

## Use-Case: Appoint New Shop Manager

**Actor:** Shop Owner, Shop

**Pre-Condition:** Shop Owner logged in to his account.

**Parameters:** User ID to appoint.

**Main-Scenario:**

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.
4. The User ID will be added to the Owners list of the shop.

**Alternative-Scenario:**

1. If the User ID doesn’t belong to a registered User, the system will return that to the Shop Owner.
2. 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.
3. 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.

## Use-Case: Change Manager Privileges

**Actor:** Shop Owner, Shop

**Pre-Condition:** Shop Owner logged in to his account.

**Parameters:** User ID of a manager.

**Main-Scenario:**

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 **Shop** will check that the User ID belong to a manager.
4. The **System** will show all the permission of the manager with the User ID.
5. The **Shop Owner** select the permission he wants to remove or add to the manger.
6. The permission of the manager will change, and he will be notifying about that.

**Alternative-Scenario:**

1. If the User ID doesn’t belong to a manger of the shop, the system will return that to the Shop Owner.
2. If the permission the shop owner choose is illegal the system will alert the user and don’t change the permission.

|  |  |  |
| --- | --- | --- |
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |
| Change Manager Privileges | Shop with Owners list [1,2] and manager list [3]  Owner 1 request to change manager 3 from getting notification | Success |
| Shop with Owners list [1,2] and manager list []  Owner 1 request to change manager 3 from getting notification | 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:**
5. The actor selects a manager which he’d like to change his permissions.
6. The system will present the actor to permissions options he has to choose from.
7. The shop owner selects the new permissions for the shop manager.
8. Expected result: The selected manger’s permission will be changed effective immediately.
   1. Good Scenarios:
      1. The changes the shop owner made for the shop manager’s permissions are changed accordingly and recognized by the system.
      2. The shop owner removed all permissions from the shop manager. He is still a shop manager with no permissions.
   2. Bad Scenarios:
      1. The changes the shop owner made were applied to all the shop managers of the store.
      2. The system does not recognize the new changes of the permissions.

## 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:**
5. The shop founder selects the option to close the store.
6. The system updates the shop’s official that the shop is closed.
   1. Real-Time notification action will start.
   2. Delayed notification action will start **(not in version 0).**
7. 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.
   1. Good Scenarios:
      1. 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.
      2. A user searches for a closed shop. The system does not display the shop.
      3. A user search for a product which is sold by a closed shop. The system does not display the closed shop sells that product.
   2. Bad Scenarios:
      1. The system has successfully closed the shop, but the system managers and the shop’s officials are not able to access the closed shop.
      2. A user searches for a closed shop. The system displays the shop.
      3. The products of the closed shop are displayed upon search.
      4. Upon closing the shop, the shop remains open.
      5. No notifications were sent to the store’s officials upon closing the shop.

## Use-Case: Request information on officials of the shop

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:**
5. 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.

* 1. Good Scenarios:
     1. The system displays the correct information of the selected officials.
     2. The system displays nothing if no official was selected.
  2. Bad Scenarios:
     1. The system displayed sensitive information or information which was not supposed to be displayed (such as login ID or User’s password).
     2. The system does not display the selected official’s information.

## 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:**
5. The shop owner requests the system to display the shop’s sales history, filtered by given parameters or no parameters at all.

Expected result: The system will display the sale’s history.

* 1. Good Scenarios:
     1. The system displays the shop’s sales history accordingly to the given filter.
     2. The system displays sales of products which are not in the shop’s inventory anymore when requested to.
  2. Bad Scenarios:
     1. The System deletes or ignores sales of products which are removed from the shop’s inventory and will not display it upon request.
     2. The system does not display the requested filtered sales.
     3. The system displays all sales in all shops with similar products.

# **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.

1. **Parameters:** shop’s name
2. **Actions:**
3. System: presents all the history of the shop purchases.
4. User: approves

## 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.

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

|  |  |  |
| --- | --- | --- |
| Use-Case | Parameter | Expected Output |
| Register | Username that doesn’t exist in the system. | Success |
|  | Username that already exist in the system. | Fail |
| 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 |
| 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 |
| 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 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 |
| 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 |

Capabilities and Responsibilities of Classes

* **Market** – represents the whole system.

This class interacts with 'Users' package - creates the system manager, users are visitors in the market.

Also, the class creates an external connection manager.

* **ExternalConnector (singleton)** – has a responsibility for creating and managing external connections, with authorized supply systems and payment systems.
* **User** – main entity in our system. Represent a visitor in the market, and a shopper in the market's shops. We have several kinds of user states, that defines the capabilities and responsibilities available for him.
* **Guest** – state of user. Has all the capabilities and responsibilities of guest-visitor as mentioned in the general requirement document.
* **Member** – state of user. Has all the capabilities and responsibilities of member-visitor as mentioned in the general requirement document.
* **System-Manager-** subclass of member. Special kind of member-user, that has maximum permissions available in the system.
* **Shop** – main entity in our system. Has responsibilities for managing purchase and discount policy and enforcing them in user's purchases.

In addition, has responsibility in completing purchases process: receive approval for payment and supply of the items, save of the purchases, updating the inventory about the items bought in the purchase.

* **ShopPolicy** – manages all the policy related to the shop – discounts for products, purchase policy for products and products prices.
* **Inventory** – manages all the inventory of the system. With every purchase completed, has the responsibility for updating the product's available units in the inventory. Also, the class must not allow completing order of product with X units (for example) when there are less then X units of the product in the inventory.
* **Product** – represent a general product, each shop can choose whether to sell it or not.

**Dictionary:**

* + - 1. **Market** – Refers to the system.
      2. **Guest User** - A user which is not a member in the system. See ‘מבקר אורח’ in general requirements file.
      3. **Member User** – A user which is a member in the system, with a unique identifier and password (username\ID). See ‘מבקר-מנוי’ in general requirements file.
      4. **Shop manager** – A member user which has unique permissions for a specific shop in the system. See ‘מבקר-מנוי שהינו מנהל חנות’ in the general requirements file.
      5. **Shop Owner** – A member user which owns a shop in the system. See ‘מבקר מנוי שהינו בעל חנות’ in the general requirements file.
      6. **Shop founder** – A shop owner which has created a shop in the system. See ‘מבקר-מנוי שהינו מייסד חנות’ in the general requirements file.
      7. **System Manager** – A member user which has unique permissions with the system. See ‘מבקר-מנוי שהינו מנהל מערכת’ in the general requirements file.
      8. **Officials\Shop’s official** – Shop owner, shop founder or shop manager which is associated with the given shop.
      9. **Shop** – Created by a member user in the system. Has a unique identifier. Products can be added to the shop.
      10. **Shopping Cart\User’s Cart** – The user’s list of products which he’d like to purchase.
      11. **Invoice\receipt\User’s Purchase** – Shows a user’s receipt of a successful transaction with a shop in the system. The following information must be included: Shop’s name, information of the products which were purchased, total price.
      12. **Shop’s Purchases** – All User’s purchases which has the shop’s unique identifier.
      13. **Supplier** – External service of the system which is responsible to deliver the product to the User which bought it.
      14. **Inventory** – Contains the amount of each product the shop has.
      15. **Notification** – A message which can be sent to one or more users. The message can be sent either immediately or delayed until the user is logged in.

**הערות טקסטואליות לתרשים המחלקות**

1. בקשר user-market, הקרדינליות בצד user (n..1) מציינת כי יש לפחות משתמש אחד שהוא מנהל מערכת במערכת הנ"ל.
2. מייסד חנות הוא בעל החנות הראשון שלה. מכאן, משתמע שכל מייסד חנות הוא גם בעל חנות.
3. לכל משתמש-מנוי בתפקיד בעל חנות/מנהל חנות יש משתמש שמינה אותו לתפקיד, מלבד מייסד החנות.
4. אין מעגליות במינויים לתפקידים ניהוליים בחנות.
5. User יוכל לתפקד כrole של shop manager, shop owner, shop founder רק אם הstate שלו הוא member – הכוונה שהוא משתמש מנוי למערכת המסחר.