Guest-Visitor Scenarios

* Use-Case: Enter the market.

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

Subscribed-Visitor Scenarios

* Use-case: Logout

1. **Actor:** subscribed 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: subscribed 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.

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 |

**Use Cases**

**Use-case** Supply

**Actor:** Shop, Market

**Parameters:** items to delivery, client’s address.

**Pre-Condition:** payment was successful.

**Post-Condition:** Supplier will send the delivery.

**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.

**Use-case** Add Item to Inventory

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

**Parameters:** Item, Quantity.

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

**Post-Condition:** change the inventory.

**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

**Parameters:** Item.

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

**Post-Condition:** change the inventory.

**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

**Parameters:** Item.

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

**Post-Condition:** Item is Modify.

**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

**Parameters:** new buying policy values.

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

**Post-Condition:** Shop Policy is modified.

**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

**Parameters:** new discount policy value.

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

**Post-Condition:** Shop Policy is modified.

**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

**Parameters:** items id, new buying policy value.

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

**Post-Condition:** Shop Policy is modified.

**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

**Parameters:** items id, new discount policy value.

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

**Post-Condition:** Shop Policy is modified.

**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

**Parameters:** User ID to appoint.

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

**Post-Condition:** the shop has new shop owner.

**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

**Parameters:** User ID to appoint.

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

**Post-Condition:** the shop has new shop 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 **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

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

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

**Post-Condition:** the permission of the manger is changes.

**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 |

**Usecase: Initializing Marketplace System**

Actor: System Manager

Preconditions: System Integrity rules are fully implemented

Parameters: External Connector

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. System presents method of payment service pool.
6. User must select a minimum of one option from the pool.
7. System attempts to establish connections with the selected payments services.
8. Records chosen services in the database.
9. If fails, go back to step 5 (up to 3 times, else jump to step 16).
10. System presents supplies service pool.
11. User must select a minimum of one option from the pool.
12. System attempts to establish connections with the selected supplies services.
13. Records chosen services in the database.
14. If connection succeeds, System marketplace is now open.
15. If fails, go back to step 11 (3 times).
16. Request the user to Initialize in a different time, rollback recorded data.

Successful Scenario:

System successfully creates a system manager, connects to at least one payment service, at least one supplies service.

Marketplace is up and running.

Erroneous Scenario:

System created a user that is not a System manager.

System could not connect to any payment service.

System could not connect to any supplies service.

System could not record the new System Manager\Payment services\ Supplies Services in the database.

**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 System Manager chooses “interchanging external services connection”
   2. System asks user to choose services to connect to and presents a pool of available services.
   3. System Manager chooses 1 service he wants to connect to.
   4. System asks if user is done selecting services to connect to.
   5. If not done, jump to 2.2
   6. System asks user to choose services to disconnect from and presents a pool of currently connected services.
   7. System Manager chooses 1 service he wants to disconnect from.
   8. System asks if user is done selecting services to disconnect from.
   9. If not done, jump to 2.6.

2.10. Invoke “interchanging external services connection” using parameters obtained from and 2.3 and 2.8.

* 1. If successful, inform the user.

2.12 If fails inform the user and inform that no changes were made.

* 1. If System Manager chooses “adding external services connection”
  2. System asks user to choose services to connect to and presents a pool of available services.
  3. System Manager chooses 1 service he wants to connect to.
  4. System asks if user is done selecting services to connect to.
  5. If not done, jump to 3.2
  6. Invoke “adding external services connection” using parameters obtained from and 3.4.
  7. If successful, inform user.
  8. If fails inform user and inform that no changes were made.

Successful Scenario:

System presents user with options:

User selects “interchanging external services connection” and selects one or more services to connect to and 1 or more services to disconnect to. Invoking interchanging external services connection usecase is successful.

User selects “adding external services connection” and selects one or more services to connect to. Invoking adding external services connection usecase is successful.

Erroneous Scenario:

System presents user with options:

User selects “interchanging external services connection”:

selects one or more services to connect to and 1 or more services to disconnect to. Invoking interchanging external services connection usecase is unsuccessful.

Pool of services to connect to has services already currently connected.

Pool of services to disconnect from has services that aren’t currently connected.

User selects “adding external services connection”:

selects one or more services to connect to. Invoking adding external services connection usecase is unsuccessful.

Pool of services to connect to has services already currently connected.

**Usecase: interchanging external services connection**

Actor: System Manager, 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 for each of the services.
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.

Successful Scenarios:

All services to connect to are currently not connected, and all services to disconnect from are currently connected.

For each service type: 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”.

Erroneous Scenarios:

some services to connect to are currently connected, or some services to disconnect from aren’t currently connected.

One of service types: number of services to connect to + number of services connected <= number of services to disconnect to.

System can’t connect to some services from “services to connect to”.

System can’t disconnect from all services from “services to disconnect from”.

**Usecase: adding external services connection**

Actor: System Manager, Changing connection with external services usecase.

Preconditions: User is logged in and is System Manager, 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.

Successful Scenario:

All services to connect to are currently not connected.

System manages to connect to all services from “services to connect to”.

Erroneous Scenario:

some services to connect to are currently connected.

System can’t connect to some services from “services to connect to”.

**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.

Successful Scenarios:

System presents all available payment methods; user selects one of them and is debited his payment due. Payment is successful. A distinct transaction number is returned.

Erroneous Scenario:

Some available payment methods are not presented.

User can select more than one payment method.

Payment is unsuccessful but transaction number still returns.

System is unable to connect to the selected payment service.

A different payment method is applied than the one chosen.

The amount the user was debited is different than the amount due.

**Use-case 3.1: getting information.**

1. Actor: User

2. Precondition: There is at least one registered shop \ product.

3. Parameters: the user finds the wanted shop \ product.

4. Actions:

* User: select product \ shop information.
* System: display the info of the selected object (product \ shop).
* Good scenario: user will see the shop info after selects it the click on the info button.
* Bad scenario: info will not present to the user \ will present to the user but

It’s incorrect or incomplete information.

**Use-case 3.2: search Items.**

1. Actor: User

2. Precondition: none.

3. Parameters: identifying information for the wanted product.

4. Actions:

1. User: insert the identifying information.

2. System: Search for Items that connected to the giving info.

3. The search results will appear on the screen.

* Good scenario: the system displays the correct result of the identifier given by the user.
* Bad scenario: the system does not find product even there is a product that fits with the information. The system does not display the correct product.

**Use-case 3.3: Saving products in a shopping cart.**

1.Actor: User

2. Precondition: the user finds the wanted product.

3. Parameters: the wanted product display on the screen.

4.Actions:

1. User select the product.

2. the system update the user cart if the product is in stock.

* Good scenario: the system will update the product in his cart.
* Bad scenario: the system will connection to the wrong cart (other user) \ connection with the wrong product \ will no make connection at all.

**use- case 3.4 Editing the shopping cart**

1. Actor: User

2. Precondition: the user has at least one Item on the cart.

3. Parameters: none, action of the user.

4.Actions: for each product will appears the quantity.

* + Remove Item:
    - User: The user opens his cart.
    - User: edit the number to ‘0’.
    - User: save the action.
    - System: remove the item from the user cart.
  + Edit the quantity:
    - User: edit to the wanted quantity.
    - User: save the action.
    - System: check at the shop, product from if it can provide the wanted quantity.
    - System: If it’s possible, will update the shop about the quantity. Other, the system will not allow to save the user action.
* Good scenario: the product will not appear in the user cart / will appears with the new wanted quantity. System do not edit the quantity do to shop disapproval.
* Bad scenario: the product will appear with quantity ‘0’ / wrong quantity / other Item will be updated.

**Use-case 3.5: check-out (buying)**

1. Actor: User

2. Precondition: the user has at least one Item on the cart.

3. Parameters: none, action of the user.

4.Actions:

* + The user opens he’s cart.
  + User clicks on the payment icon.
  + The system calculates the total amount of the product and display it on the screen.
  + The User insert the payment details
  + The System divides the payments by stores and sends the payment amount, for which store with the payment details to the external service (Payment option).
  + After receiving approves that payment done the system will send the invoicing to the associated shops and display the invoicing for the user.
  + The system will save that action in the data base, both the user and for the stores associated with the transaction.
* Good scenario: 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.
* Bad scenario: The system 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.
* **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).
8. 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: 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: 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.
6. 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: 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 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.
6. 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.

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.
* **Subscriber** – state of user. Has all the capabilities and responsibilities of subscriber-visitor as mentioned in the general requirement document.
* **System-Manager-** subclass of subscriber. Special kind of subscriber-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 subscribed to the system. See ‘מבקר אורח’ in general requirements file.
      3. **Subscribed User** – A user which is subscribed to the system with a unique identifier and password (username\ID). See ‘מבקר-מנוי’ in general requirements file.
      4. **Shop manager** – A subscribed 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 subscribed 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 subscribed 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 subscribed 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.