# Use Cases – System 1

### 1.1.1 Use case – Init the trading system

- Actor: System manager
- Precondition: System doesn't open before
- Postcondition: system running with all its components
- Parameter: None
- Actions:
  - Open the system.
  - System manager login to the system
  - Create connections to payment service.
  - Create connections to delivery service.
- Acceptance tests:
  - Positive test: Init the system in the first time and see that it works.
  - Negative test: None.

### 1.1.2 Use case – manage connections with external services.

- Actor: System manager
- Precondition:
  - System manager logged in to the system.
- Postcondition: the changes made should be saved in the system
- Parameter: external service details
- Actions:
  - Connect to service. If service is unavailable return error message and stop.
  - Choose what you want to do.
  - Save changes.
- Acceptance tests:
  - Positive test: connect to external system with right details and check the connection work.
  - Negative test: connect to external system with wrong details and check the connection didn’t work.

### 1.1.3 Use case – Payment

- Actor: guest/member
- Precondition: guest/member login to the system
- Postcondition: the payment should be made
- Alternative: guest/member receives error message.
- Parameter: shop's bank details, guest/member's payment details and amount to pay
- Actions:
  - Guest/member confirms the details and the amount to pay
  - If amount to pay is equal or less than 0, error message will send, and the process won't continue.
  - Connect to payment service with all the parameters and making the payment.
  - Send to guest/member confirmation that the payment was made successfully.

    - Send alert to the shop manager with the confirmation that the payment was made successfully.
- Acceptance tests:
    - Positive test: Connect to payment service and try to do payment with legal details.
    - Negative test: Connect to payment service and try to do payment with 0 in the amount to pay and check that we get error message.

### 1.1.4 Use case – Delivery

- Actor: guest/member
- Precondition: guest/member have order that he need to create delivery to.
- Postcondition: the delivery should be made
- Alternative: guest/member receives error message.
- Parameter: shop's address ,guest/member's address, and package details
- Actions:
    - guest/member confirms the address.
    - Connect to delivery service with all the parameters.
    - If one of parameter is invalid the system return error massage and stop the delivery process.
    - guest/member chooses day to deliver and confirm.
    - Send to guest/member confirmation about the delivery with its details.
    - Send alert to the shop manage with the delivery details.
- Acceptance tests:
    - Positive test: Connect to delivery service and try to create delivery with legal details.
    - Negative test: Connect to delivery service and try to create delivery with invalid parameters and check that we get error message.

### 1.1.5/1.1.6 Use case – Alerts

- Actor: user
- Precondition: user exist in the system
- Postcondition: the relevant alert should be received by the relevant user
- Parameter: user's id and alert details
- Actions:
    - Check if user login to the system.
    - If user is login to the system:
        - System sends user real time alert.
    - Else, System sends to user Delayed alert, that save the alert in the system and will appear to the user when he logged in.
- Acceptance tests:
    - Positive test:
        - Send alert to user that already logged in to the system and see that he sees the alert.
        - Send alert to user that not logged in to the system and see that he sees the alert when he logged in.
    - Negative test: send alert to user that doesn't exist in the system.

# Use Cases – Guest 1

## 2.1.1 Use case- Enter

- Actor: Guest
- Precondition: None
- Postcondition: the guest can access to system
- Parameter: None.
- Action:
    - User enter the system and become a guest user.
- Acceptance tests:
    - Positive test – User enter the system and succeed to enter the system
    - Negative test – None.

## 2.1.2 Use case – Exit

- Actor: Guest
- Precondition: None
- Postcondition: guest no longer has access to the system and his cart is deleted from the system
- Parameter: User's identity
- Action:
    - User exits from the system, and get response that he exit the system.
    - If the user is guest his cart is deleted and next time he will start with empty cart.
- Acceptance tests:
    - Positive test – User exit the system and his cart removed from the system.
    - Negative test – None.

## 2.1.3Use case – Register

- Actor: Guest
- Precondition: None
- Postcondition: a user with the provided details should be registered in the system
- Alternative: guest receives error message.
- Parameter: User's details (username, password, email, and address).
- Action:
    - User enter all his details to the system.
    - If all the details are valid, and username doesn't already exist in the system, user registers to the system and can login to do member's actions.

    - Else, user gets error message that the details are not valid.
- Acceptance tests:
    - Positive test – User enter valid user's details  and register to the system. After that you try to do login to check that the user registered.
    - Negative test:
        - User try to register with username that already exist, and we supposed to get error.
        - User try to register with invalid details, and we suppose to get error.

### 2.1.4 Use case – login

- Actor: Guest
- Precondition: guest is not logged in
- Postcondition: the user is logged in to the system
- Alternative: guest is not logged in and receives error message
- Parameter: User identifier and user password
- Action:
    - User enter user's identifier and password
    - If the details are correct the user login to the system and user's state become member
    - Else, user gets error message that username or password are incorrect.
- Acceptance tests:
    - Positive test – User enter correct user's identifier (user that exist in the system) and password and login to the system.
    - Negative test – Enter correct username and wrong password, and we suppose to get error message.

## Use Cases – buying actions of guest 2

### 2.2.1 use case – get shop information

- Actor: User
- Precondition: user entered to the system
- Postcondition: the user should receive the information
- Parameter: shop or product details
- Action:
    - The user selects if he wants to get info about shop or product.
    - After that the user will enter the name of the product or shop, and if the product or shop exist in the system the user get the information about the product or the shop.
- Acceptance tests:
    - Positive test :
        - the user chooses to get information about a shop and enter a name of a shop that exist in the system and get the info about the shop.
        - the user chooses to get information about a product and enter a name of a product that exist in the system and get the info about the product.

        - Negative test:
            - the user chooses to get information about a shop and enter a name of a shop that not exist in the system and get error message that the shop not exist.
            - the user chooses to get information about a product and enter a name of a product that not exist in the system and get error message that the product not exist.

### 2.2.2 Use case – search products

- Actor: User
- Precondition: user entered to the system
- Postcondition: the user should receive a list with the relevant products
- Parameter: filter terms
- Action:
    - The user define all the filter terms on products that he wants to get.
    - The system returns all the products that match to this terms.
- Acceptance tests:
    - Positive test :
        - the user ask to get all the products with price that is under 50 NIS and the system returns all the suitable products.
    - Negative test:
        - the user ask to get a product with price that is under -10 NIS and the system return error message that the price cannot be negative.

### 2.2.3 Use case – save items to user's shopping basket

- Actor: User
- Precondition: user entered to the system
- Postcondition: the provided product is added to the relevant shop's basket of the user
- Parameter: list of products to save of specific shop.
- Action:
    - The user add products to the shopping basket in a specific shop.
    - The price of the added product will be calculated with the relevant discount.
    - The shopping basket of the user in this specific shop will be updated.
- Acceptance tests:
    - Positive test :
        - The user chooses list of products to add to his shopping, the system saves the products in the user shopping baskets.
    - Negative test:

        - the user chooses empty list of products to add to his shopping basket, the system returns error message.

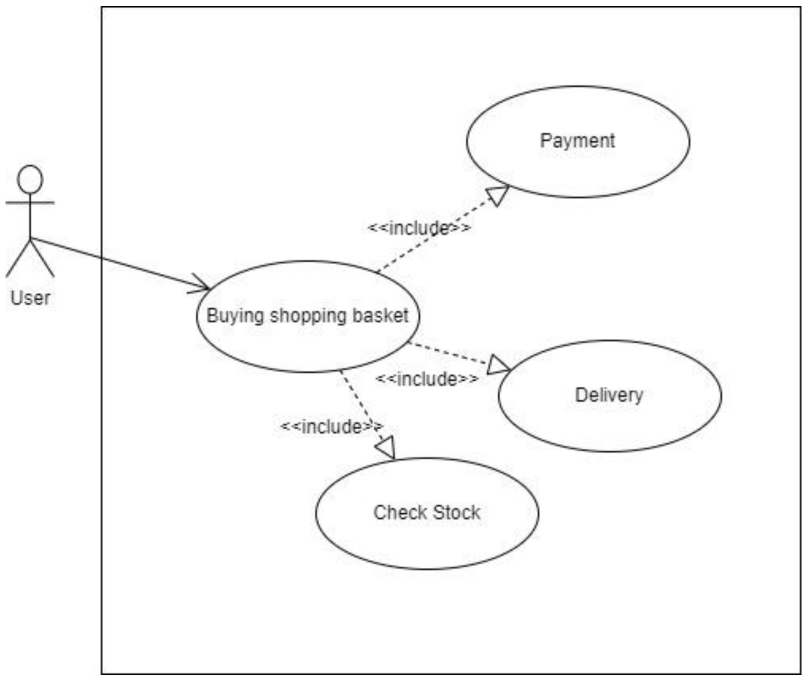
### 2.2.4 Use case – Checking shopping cart contents

- Actor: User
- Precondition: user entered to the system
- Postcondition: the user receives the contents of his shopping cart
- Parameter: user's cart
- Action:
    - The user requests to view his shopping cart and the system provides it him.
    - The user has the option to change quantities of products in shopping basket, and remove products
- Acceptance tests:
    - Positive test :
        - User select to view his cart and the system returns him his real basket.
    - Negative test:
        - User select to view his shopping cart and try to change the quantity of a product to negative number. The system returns error message that quantity of a product cannot be negative.

### 2.2.5 Use case – Buying shopping basket

- Actor: User
- Precondition: shopping cart is not empty
- Postcondition: a payment and delivery requests should be made
- Alternative: user receives error message and the purchase isn't made and the product quantities doesn't decrease.
- Parameter: valid user identifier and cart detail
- Action:
    - The user select to buy his shopping cart. For each item in the shopping cart system checks availability (UC 2.2.5.1).
    - If there is product that isn't available in stock, the system alerts the user for the missing product and continue the process without these products.
    - The system calculates the final price at each shop according to the discount policy.
    - The user approves the price and the products in the cart.
    - For each shop the system open request of payment to the payment service (UC 1.1.3), if the request denied system returns failure.
    - For each shop the system open request of new delivery to the delivery service. (UC 1.1.4), if the request denied system returns failure.
    - If all the steps below succeeded, the system returns success.
    - System sends alerts to all shop's owners.
- Acceptance tests:
    - Positive test :

        - User fill his shopping carts with some items and buy them, if all worked the process finish with success.
    - Negative test:
        - the user tries to buy empty list of products, and the system returns error message.
        - the user tries to buy when the external paying system is not working.
        - the user tries to buy an item that is out of stock.
        - the user tries to buy an item with unmatching policies to the shop.



### 2.2.5.1 Use case – Checking stock availability

- Actor: User
- Precondition: user entered to the system
- Postcondition: None
- Parameters: product, shop, quantity
- Action:
    - The user gives a shop and a product.
    - If such shop doesn't exist, the system returns failure.
    - If such product doesn't exist in the provided shop, the system returns failure.
    - Else, system checks the product quantity in the shop's stock and if the stock quantity is equal or higher than the provided quantity, system returns success, and if its lower then system returns failure.
- Acceptance tests:
    - Positive test :
        - Assume that the system has a shop p and product pr in shop p with quantity 3 in stock.
          User provides shop p, product pr and quantity 2.
          system should return success.

    - Negative test:
        - Assume that the system doesn't have a shop p.
          User provides shop p, product pr and quantity 2.
          system should return failure.
        - Assume that the system has a shop p and doesn't have product pr in shop p.
          User provides shop p, product pr and quantity 2.
          system should return failure.
        - Assume that the system has a shop p and product pr in shop p with quantity 3 in stock.
          User provides shop p, product pr and quantity 4.
          system should return failure.

## Use Cases – Member 3

### 2.3.1 Use case - logout

- Actor: Member
- Precondition: member logged in
- Postcondition: the member no longer logged in to the system
- Alternative: member receives error message.
- Parameter: User identifier
- Action:
    - If the user was logged in than the system removes the user from the logged in user list.
    - If the user is not logged in than the system returns error message that the user is not logged in.
- Acceptance tests:
    - Positive test – given as parameter identifier of a user that is logged in to the system, this user will no longer be logged in.
    - Negative test - given as parameter identifier of a user that is not logged in to the system, error message will appear "user is not logged in".

### 2.3.2 Use case: open shop

- Actor: Member
- Preconditions: member logged in to the system
- Postcondition: a shop with the provided details should be created
- Alternative: member receives error message.
- Parameter: Shop details and user identifier
- Action:
    - if the shop name provided in the details is valid (a shop with that name doesn't already exist) the system will create a new shop with the provided details and the user will be assigned as the shop founder.
    - the system defined default shop policy.
    - else return failure.
- Acceptance tests

  - Positive test: given identifier of a logged in user and shop details new shop will open and the user will be the shop founder of the new shop.
  - Negative case: given shop details with a name that already exists, error message will appear "shop with that name already exists".

## Use Cases – Shop Owner 4

### 2.4.1.1 Use case: Add product

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** a product with the provided details should be added to the provided shop.
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, product-name, product-details
- **Actions:**
  1. System asks for a shop name.
  2. Shop Owner specifies shop name.
  3. if the Shop owner is indeed a shop owner of the provided shop:
     1. System asks for product name and details
     2. Shop Owner specifies a name and details
     3. if product with such name exists in the Shop Owner's shop, System returns failure
     4. if product with such name doesn't exist:
        - i. System creates product *p* with the specified name and details.
        - ii. System adds *p* to the shop.
        - iii. System returns success.
  4. else, System returns failure.
- **Tests:**
  1. assume that we have user u and shop p without product named "X" and u is a Shop owner of p.
     - a. u provides the shop p, product name "X" and product details "..." .
     - b. expected output: success.
  2. assume that we have user u and shop p with product named "X" and u is a Shop owner of p.
     - a. u provides the shop p, product name "X" and product details "..." .
     - b. expected output: failure.
  3. assume that we have user u and shop p without product named "X" and u is **not** a Shop owner of p. u provides the shop p, product name "X" and product details "...". expected output: failure.
  4. assume that we have user u and we don't have shop p. u provides the shop p, product name "X" and product details "...". expected output: failure

### 2.4.1.2 Use case: Remove product

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided product should be removed from the provided shop
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, product-name
- **Actions:**
  1. System asks for a shop name
  2. Shop Owner specifies shop name
  3. if the Shop owner is indeed a shop owner of the provided shop:

        1. System asks for product name
        2. Shop Owner specifies a product name
        3. if a product *p* with such name exists in the Shop Owner's shop:
            i. System removes product *p* from the shop.
            ii. System returns success
        4. if product *p* with such name doesn't exist in the Shop Owner's shop, System return failure
    4. else, System returns failure
- **Tests:**
    1. assume that we have user u and shop p with product named "X" and u is a Shop owner of p. u provides the shop p, product name "X" .
        - expected output: success
    2. assume that we have user u and shop p without product named "X" and u is a Shop owner of p.
        - u provides the shop p, product name "X" and product details "..." .
        - expected output: failure
    3. assume that we have user u and shop p without product named "X" and u is **not** a Shop owner of p.
        - u provides the shop p, product name "X" and product details "..." .
        - expected output: failure
    4. assume that we have user u and we don't have shop p. u provides the shop p, product name "X" and product details "..." .
        - expected output: failure

### 2.4.1.3 Use case: Change product details

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided products details should be changed to the provided details
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, product-name, new-product-details
- **Actions:**
    1. System asks for a shop name
    2. Shop Owner specifies shop name
    3. if the Shop owner is indeed a shop owner of the provided shop:
        a. System asks for product name and the new details
        b. Shop Owner specifies a name and new details
        c. if product with such name doesn't exists in the Shop Owner's shop, System returns failure
        d. if product *p* with such name exist:
            1. System changes the details of *p* to the provided new details
            2. System returns success
    4. else, System returns failure
- **Tests:**
    1. assume that we have user u and shop p with product named "X" and u is a Shop owner of p.
        - u provides the shop p, product name "X", product details "...".
        - expected output: success
    2. assume that we have user u and shop p without product named "X" and u is a Shop owner of p.
        - u provides the shop p, product name "X" and product details "..." .

   - expected output: failure
3. assume that we have user u and shop p without product named “X” and u is **not** a Shop owner of p.
   - u provides the shop p, product name “X” and product details “...” .
   - expected output: failure
4. assume that we have user u and we don't have shop p. u provides the shop p, product name “X” and product details “...” .
   - expected output: failure
5. assume that we have user u and shop p with product named “X”, product named “Y” and u is a Shop owner of p.
   - u provides the shop p, product name “X”, product details “...” (the new name is “Y”).
   - expected output: failure.

### 2.4.2 Use case: *Change shop buying and discount policy*

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the shop buying and discount policy should be changed to the provided policy
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, new-policy
- **Actions:**
  1. System asks for a shop name
  2. Shop Owner specifies shop name
  3. if the Shop owner is indeed a shop owner of the provided shop:
     1. System asks for a new policy details
     2. Shop Owner specifies the new policy details
     3. System updates the shop buying and discount policy to the new policy
  4. else, System returns failure
- **Tests:**
  1. assume that we have users u1 and shop p, u1 is a Shop owner of p.
     - u1 provides the shop p, policy details.
     - expected output: success
  2. assume that we have users u1 and don’t have shop p.
     - u1 provides the shop p, policy details.
     - expected output: failure
  3. assume that we have users u1 and shop p, u1 is **not** a Shop owner of p.
     - u1 provides the shop p, policy details.
     - expected output: failure

### 2.4.4 Use case: *Add owner*

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided member should have owner permissions to the provided shop
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, member
- **Actions:**
  5. System asks for a shop name
  6. Shop Owner specifies shop name
  7. if the Shop owner is indeed a shop owner of the provided shop:

        1. System asks for a Member
        2. Shop Owner specifies the Member
        3. if the Member provided is not already an owner of the shop:
            1. System appoints him to be an additional Owner of the shop by the Shop owner that appoints him.
            2. System gives the Member provided, permissions regarding shop policy and management.
            3. System returns success
        4. if the Member provided is already an owner of the shop, System returns failure
    8. else, System returns failure
- **Tests:**
    4. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is not a Shop owner of shop p.
        - u1 provides the shop p, user u2.
        - expected output: success
    5. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop owner of shop p.
        - u1 provides the shop p, user u2.
        - expected output: failure
    6. assume that we have users u1,u2 and shop p, u1 is **not** a Shop owner of p.
        - u1 provides the shop p, user u2.
        - expected output: failure

### 2.4.5 Use case: Remove owner

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided member should have manager permissions to the provided shop
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, owner-shop-to-remove
- **Actions:**
    1. Check if Shop Owner appointee owner-shop-to-remove:
    2. If yes:
        a. Remove  owner-shop-to-remove from owners of shop
        b. Send Alert to owner-shop-to-remove.
        c. Remove all owners that owner-shop-to-remove appointee
    3. Else: System returns failure
- **Tests:**
    - Positive: remove owner of shop by his appointee
    - Negative: : remove owner of shop not by his appointee

### 2.4.6 Use case: Add manager

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided member should have manager permissions to the provided shop
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, subscribed-user
- **Actions:**
    1. System asks for a shop name

2. Shop Owner specifies shop name
3. if the Shop owner is indeed a shop owner of the provided shop:
    1. System asks for a Member
    2. Shop Owner specifies the Member
    3. if the Member provided is not already an owner or manager of the shop:
        1. System appoints him to be an Manager of the shop by the Shop owner that doing this action.
        2. System gives the Member provided, permissions regarding information gathering
        3. System returns success
    4. if the Member provided is already an owner or manager of the shop, System returns failure
4. else, System returns failure

- **Tests:**

1. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is not a Shop manager or Shop owner of shop p.
    - u1 provides the shop p, user u2.
    - expected output: success
2. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop manager of shop p.
    - u1 provides the shop p, user u2.
    - expected output: failure
3. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop owner of shop p.
    - u1 provides the shop p, user u2.
    - expected output: failure
4. assume that we have users u1,u2 and shop p, u1 is **not** a Shop owner of p.
    - u1 provides the shop p, user u2.
    - expected output: failure

### 2.4.7 Use case: Change manager permissions

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the provided subscribed user manager permissions in the provided shop should be changed to the provided permissions
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, subscribed-user, new-permissions
- **Actions:**

1. System asks for a shop name
2. Shop Owner specifies shop name
3. if the Shop owner is indeed a shop owner of the provided shop:
    1. System asks for a Member and new permissions to replace the old once.
    2. Shop Owner specifies the Member and the new permissions
    3. if the Member provided is a Manager of the shop and he was appointed to be Manager by the Shop owner then:
        1. System changes its permissions in the shop to the new permissions provided.
        2. System returns success
    4. else, System returns failure
4. else, System returns failure

- **Tests:**

1. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop manager of shop p and was appointed to that position by u1.
    - u1 provides the shop p, user u2 and some new permissions.
    - expected output: success
2. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop manager of shop p and was **not** appointed to that position by u1.
    - u1 provides the shop p, user u2 and some new permissions.
    - expected output: failure
3. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is **not** a Shop manager of shop p.
    - u1 provides the shop p, user u2 and some new permissions.
    - expected output: failure
4. assume that we have users u1,u2 and shop p, u1 is **not** a Shop owner of p.
    - u1 provides the shop p, user u2.
    - expected output: failure

### 2.4.9 Use case: Close shop

- **Actor:** Shop founder
- **Precondition:** Shop founder is logged in
- **Postcondition:** the provided shop should be closed
- **Alternative:** shop founder receives error message.
- **Parameters:** shop-name
- **Actions:**
  1. System asks for a shop name
  2. Shop founder specifies shop name
  3. if the Shop founder is indeed the founder of the provided shop:
      1. System changes the shop status to “Out of business”.
      2. System makes the shop information private to everyone but the shop owners and System managers
      3. System sends a message to all the owners and managers of the shop
      4. System returns success
  4. else, System returns failure
- **Tests:**
  1. assume that we have user u and shop p, u is a Shop founder of shop p.
      - u provides the shop p.
      - expected result: success
  2. assume that we have user u and shop p, u is **not** a Shop founder of shop p.
      - u provides the shop p.
      - expected result: failure
  3. assume that we have user u and we don't have shop p.
      - u provides the shop p.
      - expected result: failure

### 2.4.11 Use case: Get employee information

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the shop owner should receive the relevant information
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name, subscribed-user

- **Actions:**
  1. System asks for a shop name
  2. Shop Owner specifies shop name
  3. if the Shop owner is indeed a shop owner of the provided shop:
     1. System asks for a Member
     2. Shop Owner specifies the Member
     3. if the Member provided is an employee of the shop:
        1. if the subscribed-user provided is a manager of the shop, System returns its information and also information on the permissions the manager has in the shop
        2. else, the System returns the provided subscribed-user's information
     4. if the Member provided is not an employee of the shop, System returns failure
  4. else, System returns failure
- **Tests:**
  1. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a Shop manager of shop p.
     1. u1 provides the shop p, user u2 .
     2. expected output: information on u2 and its managing permissions.
  2. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is a worker of shop p but not a Shop manager.
     1. u1 provides the shop p, user u2.
     2. expected output: information on u2.
  3. assume that we have users u1,u2 and shop p, u1 is a Shop owner of p, u2 is **not** an employee of shop p.
     1. u1 provides the shop p, user u2 and some new permissions.
     2. expected output: failure
  4. assume that we have users u1,u2 and shop p, u1 is **not** a Shop owner of p.
     1. u1 provides the shop p, user u2.
     2. expected output: failure

### 2.4.13 Use case: Get purchase history

- **Actor:** Shop Owner
- **Precondition:** Shop owner is logged in
- **Postcondition:** the shop owner should receive the relevant information
- **Alternative:** shop owner receives error message.
- **Parameters:** shop-name
- **Actions:**
  1. System asks for a shop
  2. Shop Owner specifies the shop
  3. if the Shop owner  is not an owner of the provided shop, System returns failure
  4. else, System returns all the purchase history of the shop
- **Tests:**
  1. assume that we have user u and shop p, u is a Shop owner of p.
     - u provides the shop p.
     - expected output: the purchase history of shop p.
  2. assume that we have user u and we don't have shop p.
     - u provides the shop p.
     - expected output: failure.
  3. assume that we have user u and shop p, u is **not** a Shop owner of p.
     - expected output: failure

## Use Cases – Manager of shop 5

### 2.5 Use case: manage shop

- Actor: Shop manager
- Precondition: Shop manager is logged in
- Postcondition: the provided action should be performed
- Alternative: shop manager receives error message.
- Parameter: user identifier and the action the user wants to preform
- Action:
  - user requests to perform some action
  - system checks if this user has the authority to perform the action. If the user has the authority the system preforms this action.
  - If the user does not have the authority than the system returns error message that this user cannot perform this action.
- Acceptance tests
  - Positive case: given identifier of a logged in shop manager and action that this manager can perform the system will perform the given action without a problem.
  - Negative case: given identifier of a logged in shop manager and action that this manager cannot perform than error massage will appear "you need to ask permission before preform this action"

## Use Cases – System manager 6

### 2.6.4 Use case: get purchase history

- Actor: system manager
- Precondition: user logged in and valid user identifier or valid shop identifier
- Postcondition: the system manager get the relevant information
- Alternative: system manager receives error message.
- Parameter: user identifier or shop identifier
- Action:
  - the manager provides user or shop identifier and will get the purchase history of the user or the shop.
  - if such user or shop doesn’t exist then the system returns failure.
- Acceptance tests:
  - Positive case:
    - logged to the system as system manager and ask to get purchase history of valid user.

        - logged to the system as system manager and ask to get purchase history of valid shop.
    - Negative case:
        - logged in to the system not as system manager and ask to get purchase history of valid user and see that we get error message that we don't have permission to do that.
        - logged to the system as system manager and ask to get purchase history of invalid user. expect to get failure message.
        - logged to the system as system manager and ask to get purchase history of invalid shop. expect to get failure message.

### 2.6.6 Use case: Get member Information

- Actor: system manager
- Precondition: user logged in and valid user identifier or valid member identifier
- Postcondition: the system manager gets the relevant information.
- Alternative: system manager receives error message.
- Parameter: user identifier
- Action:
    - the manager provides user identifier and will get the Information about the member.
    - if such user or shop doesn’t exist then the system returns failure.
- Acceptance tests:
    - Positive: Ask to get info about registered member
    - Negative:
        - Ask to get info about not registered member.
        - Ask to get info about registered member, not by system manager.

## הטלת אחריות לטיפול באילוצים

### אילוצי נכונות – כיצד יובטח קיום של כל אילוץ

1. תרחיש קבלה: בעת רישום משתמש, תתבצע בדיקה לזמינות השם.
2. תרחיש קבלה: עושים רישום למנהל מערכת בעת אתחול המערכת.
3. בנייה: לפי ההיררכיה בתרשים מחלקות.
4. תרחיש קבלה: לפני ביצוע פעולות בשוק תתבצע בדיקה האם הוא מבקר בשוק.
5. תרחיש קבלה: בעת יצירת חנות, מייסד החנות הוא גם בעל חנות.
6. תרחיש קבלה: בעת יצירת החנות נקבעת מדיניות ברירת מחדל.
   בנוסף הבנייה דואגת לכך שאפשר להגדיר מדיוניות שונות לכל חנות.
7. בנייה: בתרשים מחלקות מובטח לנו שלכל משתמש/אורח יש עגלת קניות אחת והיא מכילה בתוכה אוסף של סלי קניות שכל אחד שייך לחנות אחרת.
8. בנייה: לפי הבנייה רק למשתמש עצמו יש גישה לעגלת קניות שלו.
9. תרחיש קבלה: בעת הקנייה מתבצעת בדיקה לזמינות במלאי.
10. תרחיש קבלה: בתהליך הקנייה כל תנאי היושרה נבדקים.
11. בנייה: באתחול המערכת הבנייה תבטיח חיבור לשירותים החיצוניים.

## דרישות רמת שירות

- עקביות – האחריות לשמירת על עקרון העקביות מתבצעת ע"י המערכת בבדיקה שלהם בכל מקום בו יש חשד שהם יכולים להיות מופרים ובעזרת בנייה תקינה שמונעת מהפרה של העיקרון
- פרטיות – האחריות לשמירת עקרון הפרטיות מתבצעת ע"י הבנייה שלא תאפשר למשתמשים לגשת לנתונים שלא צריך להיות להם גישה, וע"י התוכנה במקרה שיש צורך לבדוק הרשאות

## פנייה למערכות חיצוניות

אחריות לפנייה למערכות חיצונית מתבצעת ע"י חלק מוגדר בתוכנה שמטפל בגישות אלו, ויהווה מעין ADAPTER בין הממשק של המערכת שלנו למערכת החיצונית.

## Glossary of Terms

1. User - can be any kind of user that uses the system.
2. guest - user that is not logged in to the system.
3. Member - a registered user.
4. Shop manager - a Subscribed user that is also a manager of a shop
5. System manager - Subscribed user that is defined by the system and responsible for the maintenance of the system.
6. Shop owner - a Subscribed user that is also an owner of a shop
7. Shop founder - a Subscribed user that is also a founder of a shop
8. user details - username, password, email, address
9. Shopping basket - list of items of the user in a specific shop
10. Shopping cart - one for each user, contains all the shopping baskets of the user.
11. Shopping types - immediate purchase, bidding, public auction, raffle.
12. Shopping policy - defines who can buy in the shop, the possible buying methods (shopping types) and the rules that apply to these methods. defined to a shop and can be applied on products or costumers or both.
13. Discount types - public discount, conditional discount, hidden discount
14. Discount policy – defines who is eligible to get a discount, the discount type (discount types) and the rules that apply to these discounts. defined to a shop and can be applied on products or costumers or both.