אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

הפקולטה למדעי ההנדסה
המחלקה להנדסת תוכנה

Faculty of Engineering Sciences
Dept. of software Engineering

Ben-Gurion University of the Negev

“Trading System”

Version 1 – Use Cases

Table of Contents

System functional requirements use-cases

Use case : Initialize Trading System

- **Actor:** System
- **Pre-condition:** user is logged in as a system administrator, there also exists a connection to external payment and supplies services.
- **Parameter:** No Parameters.
- **Actions:**
  1. System presents an option to initialize the trading system.
  2. User selects initialize trading system.
  3. System presents a "Welcome" message (optional) and "Market is now open" message (mandatory).

  Acceptance Tests:

  - Good: A logged in system administrator user initializes a system and system responds with a "Market is now open" message.
  - Bad: A logged in system administrator user tries to initialize a system without one or more external services.

Use case : Add/Remove/Replace external service connection.

- **Actor:** User
- **Pre-condition:** user is logged in as a system administrator, the trading system is initialized.
  @param is adhering the interface of our external systems adapter.
- **Parameter:** External System connection (payments or supplies)
- **Actions:**
  1. The user navigates to "**external services settings**".
  2. The system presents 3 options: "**Add external service**", "**Remove external service**", "**Replace external service**".
3. If the user selects "**Add external service**" :
   (View "Add external service connection" use-case)
   4. If the user selects "**Remove external service**" :
   (View "Remove external service" use-case)
   5. If the user selects "**Replace external service**" :
   (View "Remove external service" use-case)

Use case : Add external service connection.

- **Actor:** User
- **Pre-condition:** user is logged in as a system administrator, the trading system is initialized.
  @param is adhering the interface of our external systems adapter.
- **Parameter:** External System connection (payments or supplies)
- **Action:**
  1. System requests a connection path.
  2. The user supplies the connection "conn" **(not sure in which format)**.
  3. The system sends a request to the appropriate handler in order to establish a connection.
     1. . **if successful**, the handler disconnects from it and adds it the repository
        (in correlation to the connection type).
        3.1.1 . The system presents a "Success" message.
        2. **if failed**, the system presents an "Error" message to the admin and returns to the former page.

Acceptance Tests:

- Good: The user chooses to add an external service connection, The system established a connection and added it to the repository.
- Bad: The system was unable to establish a connection due to wrong format and the user is presented with an "Error" message.

Use case : Remove external service connection.

- **Actor:** User
- **Pre-condition:** user is logged in as a system administrator, the trading system is initialized.
- **Parameter:** No parameters
- **Action:**
    1. . The appropriate handler presents all relevant connections excluding the one being used right now if exists, otherwise it presents a message "No Available connections."
    2. . User selects the connection **conn**.
    3. . The system removes the connection from the relevant repository.
    4. . The system presents a "Success" message.

    Acceptance Tests:
    - Good: The user chooses to remove an external service connection,
      The handler successfully removes the connection and a "Success" message appears.
    - Sad: The system was unable to find unused services,
      "No available connections" message appears.

Use case : Replace external service connection.

- **Actor:** User
- **Pre-condition:** user is logged in as a system administrator, the trading system is initialized.
  @param is adhering the interface of our external systems adapter.
- **Parameter:** External System connection (payments or supplies)
- **Action:**
  1. . System requests a connection path.
  2. . The user supplies the connection "conn" **(not sure in which format)**.
  3. . The appropriate handler tries to establish a connection,
     1. . **if successful**:

a. The system stops handling requests to the appropriate service,
And aggregates them in a queue.

b. The handler disconnects from the old service.
b. The handler sets the new service "conn" for the system.
b. The system request the handler to resume handling requests in a FIFO order.
b. The system presents a "Success" message.

2. . **if failed**, the system presents an error message to the admin and returns to the former page.

Acceptance Tests:

- Good: The user chooses to replace an external service connection,
  The handler established a connection and replaced the service, the user is presented with a "Success" message.
- Bad: The system was unable to establish a connection due to wrong format and the user is presented with an "Error" message.

Use case : External System Payment.

- **Actor:** System
- **Pre-condition:** User has items in his shopping cart and has initialized a purchase request.
- **Parameter:** Purchase information (e.g. total price, number of payments).
- **Action:**
  1. System requests for credit card and social security numbers.
  2. The user provides all required information.
  3. The system sends a payment request with (total price/ number of payments),
     credit-card and social security numbers to the external payment system handler which in turn sends the request to the external system.

4. If successful:

1. . The handler provides the user with a receipt and a "Payment successful" message.
2. . The handler saves all purchase information for future needs.
3. . The handler empties the users' cart.

5. If failed:

1. . The handler presents the user with a "Payment failed" message.

Acceptance Tests:

- Good: The user has items in his shopping cart and the payment request succeeds, the user is given a receipt.
- Bad: The user has items in his shopping cart and payment request succeeds but the receipt is for different items/more items/different sum than should be.
- Bad: The user has items in his shopping cart and payment request fails due to wrong order information(wrong name, wrong credit card, items doesn’t exist, etc).

Use case : External System Supplies Order.

- **Actor:** System
- **Pre-condition:** User is logged in as a shop owner/manager with a permission to manage inventory and has all order information, Market is open.
- **Parameter:** Order information (e.g. products, number of products, costumer info).
- **Action:**
  1. The handler checks whether the user has paid for the order.
     1. **If true**:
        a. The handler forwards an order request with @params to the external supplies system handler.
        b. If successful:
        b.1. The handler presents the user with "Order Successful" message and
        order confirmation with all relevant fields.
        b.2. The system saves the order confirmation and information for future
        purposes.
     c. **If failed:** The system presents the user with "Order failed" and hopefully the
        reason.
     2. **if false:** The system presents the user with "Order has not been paid for" message.

Acceptance Tests:

- Good: The user has the order information and is logged in as a shop-owner/manager, the user has paid for the order and the request for an order from the external service succeeds and the system presents the user with "Order successful" and an order confirmation.
- Bad: The user has not paid for the order yet, the system presents the user with "Order has not been paid for" message.

Use case : Real Time Messages.

- **Actor:** System
- **Pre-condition:** System has a message queue + handler, a logged-on user sent a message.
- **Parameter:** None.
- **Action:**
  1. Systems' message handler checks for messages in the queue.
  2. If **not empty**:
     1. Handler pops the message – **m** (object holds recipient + message text),
        from the queue.

2. Handler checks if the recipient online (if exists, otherwise there won't be a message in the queue to begin with)
   - If **true**:
     a.1. The handler presents the recipient with **m**'s text.
     a.2. The handler saves **m** in the recipients' past messages.
   - If f**alse**:
     b.1. The handler saves **m** in the recipients pending messages queue.

3. If **empty**:
   1. The handler awaits message.

Acceptance Tests:

- Good: The user has sent a message and the system presents the recipient with the message.
- Sad: The user has sent a message and the system presents the message to the wrong recipient.

Use case : Awaiting Messages.

- **Actor:** System
- **Pre-condition:** Each user has "pending messages queue", an online user sent a message to an offline user, the offline user then logged-on.
  The system has a message handler.
- **Parameter:** None.
- **Action:**
    1. The system sends a message handler for the newly logged on user.
    2. The handler checks for messages in the users' "pending message queue".
    3. If **not empty**:
        1. . The system presents the user with the option to select messages to read or exit the messaging system.
            1. If user selects a message, the system presents the message.
            2. If user selects exit, the system navigates to homepage.

4. If **empty**:

4.1. The system presents the user with "No pending messages" and returns him to the home page.

Acceptance Tests:

- Good: The user has logged in and the system presents him with all messages sent while he was offline, the user can view the messages.
- Bad: The user has logged in and the system presents him with messages not addressed to him.

Use Cases (including customer inspection):

Visitor

1. Use case: Entrance

-Actor: Visitor.

-Precondition: None.

-Parameter: None.

-Actions:

1. The visitor opens the app/web page of the system.

2. The system loads and shows the visitor popular items from several stores.

3. A new shopping cart is created for the visitor and allows him to add items to it while he is shopping.

Acceptance testing:

1. Good scenario: when loading the system can successfully gather all the information needed and the app/web page opens.
2. Bad scenario: when loading the visitor's connection disconnects so the system doesn't continue to create needed objects for him and deletes all objects created for him.

2. Use case: Exit

-Actor: Visitor.

-Precondition: the visitor already has the app/web page of the system open.

-Parameter: None.

-Actions:

1. The visitor wants to close the app/web page of the system.

2. The object guest deletes the visitor's shopping cart and all his records/information gathered while he was online. data saved by stores about his purchases(if happened) remains.

Acceptance testing:

1. Good scenario: when exiting the system all the needed actions occur and the system continues normally for other users.
2. Bad scenario: when trying to exit the visitor's internet crashes so an exit signal isn't sent to the system so the exit function doesn't occur.

3. Use case: Registration

-Actor: Visitor.

-Precondition: None.

-Parameter: None.

-Actions:

1. The visitor chooses to do the registration process.

2. The registration process starts and asks the visitor for his email, password, and information about him such as name and birthday.

3. After the visitor has finished, he chooses to finish the process.

4. the user object (created for him before) checks to see if the information given don't have any problems with the system requirements (such as same email as different user, or a password not meeting the password requirements).

4.1 If the user object detects a problem (like the ones said before) in the information he sends a notification to the visitor to tell them to reenter the information where it is needed.

4.2 Otherwise the user object updates his fields and changes now classifies as a registered user.

Acceptance testing:

1. Good scenario: when trying to register the visitor enters all the correct information and the system executes the registration process successfully.
2. Bad scenario1: when trying to register the visitor enters an invalid email (either an already registered email or a non-existent one) so the system can't do the registration process and informs the visitor.
3. Bad scenario2: when trying to register the visitor enters an invalid password (not meeting the password requirements) so the system can't do the registration process and informs the visitor.
4. Bad scenario3: when trying to register the visitor enters an invalid birthday (like to choose a date that is in the future) so the system can't do the registration process and informs the visitor.

4. Use case: Login

-Actor: Visitor.

-Precondition: The visitor has already registered to the system.

-Parameter: None.

-Actions:

1. The visitor chooses to start the login process.

2. The visitor enters his email and password he registered with.

3. The guest object related to the visitor checks the email and password given to see if it is known to the system as a registered user.

    3.1 If there is a problem (such as the email isn't registered, or the password is wrong) the user object informs the visitor and asks him to fix whatever is needed.

    3.2 Otherwise the user changes its state to a registered user and loads the information needed for him (his shopping cart and other information about him such as name and birthday).

Acceptance testing:

1. Good scenario: when trying to login the visitor enters all the correct information and the system executes the login process successfully.
2. Bad scenario1: when trying to login the visitor enters an invalid email (a non-registered email) so the system can't do the login process and informs the visitor.

Bad scenario2: when trying to login the visitor enters an invalid password (not matching the one in the database) so the system can't do the login process and informs the visitor.

5. Use case: getting information

-Actor: Visitor.

-Precondition: The visitor already has the app/web page of the system open.

-Parameter: None.

-Actions:

1. The visitor chooses to start the information process to check a store/item.

2. A request is sent to the marketController to show all the stores in the market, and he returns a list of all the stores in alphabetical order.

3. the visitor can choose each store to see more information about it (written by the store owner/founder/manager if has permission) and information about the items in the store

Acceptance testing:

1. Good scenario: when the visitor asks to get information, he gets a list of all the stores in the market including the one he wants, and he is allowed to choose it and see information about it.
2. Bad scenario1: when trying to get information the visitor searches for a store that does not exist, and the market informs the visitor of his mistake.
3. Bad scenario2: when trying to get information the visitor searches for an item that does not exist in the store, and the store informs the visitor of his mistake.

6. Use case: searching items

-Actor: Visitor.

-Precondition: The visitor already has the app/web page of the system open.

-Parameter: None.

-Actions:

1. The visitor chooses to start the search process.

2. the visitor is asked by the system to enter a name or words related to the item he is searching for.

3. The visitor can write the product name or words related to him, or he can search in a looser way and use only the modification options (including specific categories to pick, price range to pick and more).

3.1 If the visitor entered a name/keywords then the system goes over all the products in the market and checks if their name matches the search or if the search is included in their description. The market then returns a list of all the items picked by filtering (including the modification options) then presents all matching products sorted by their rating.

3.2 if nothing is written then the market searches only by the modification options for the best matches and presents them to the visitor, sorted by rating.

Acceptance testing:

1. Good scenario: when trying to get information he finds the item he wants and can successfully add it to his cart.
2. Bad scenario1: when trying to find an item the visitor enters a name of an item that doesn't exist in the market, and the system informs the visitor of his mistake.
3. Bad scenario2: when trying to find an item the visitor enters too many letters to the search and the system notifies him that he is over the limit.

7. Use case: saving selected products in the shopping cart

-Actor: Visitor.

-Precondition: The visitor already has the app/web page of the system open.

-Parameter: None.

-Actions:

1. The visitor chooses an item from a store and wants to add him to his cart.

2. the visitor sends a request (with the quantity he wants to buy) to the store to allow him to add the item to his cart.

2.1 if the item does exist and is in stock then the store allows the visitor to add the item to his cart.

2.2 Otherwise the store denies the request.

3. if the store agrees to add the item to the cart, then the user object checks if the visitor already has added an item from that store to his cart.

3.1 If he did, then the user object just adds the item to the needed basket in the cart (the one related to the store) and lets the visitor continue shopping.

3.2 otherwise the user object creates a new basket in the cart for the store and let the visitor continue shopping.

Acceptance testing:

1. Good scenario: when trying to add an item to the cart the process is successful, and the visitor can continue to shop.
2. Bad scenario1: when trying to add an item to the cart the visitor chooses an item that is out of stock and the store informs the visitor and doesn't allow him to add the item to his cart.
3. Bad scenario2: when trying to add an item to the cart the visitor chooses an item that is out of stock he already added to his cart before so the user object notifies the visitor that only the quantity of the item has changed in the cart (according to the new request).

8. Use case: cart content and modification

-Actor: Visitor.

-Precondition: The visitor already has the app/web page of the system open.

-Parameter: None.

-Actions:

1. The visitor chooses to see what his cart contains

2. the user object gets the request and returns a list of the cart content sorted by stores (in alphabetical order) and after each store all its items (also in alphabetical order).

3. the visitor can choose for each item if he wants to add 1 or remove 1 from the item's quantity.

4.1 If the visitor chooses to remove then the item is removed from the cart.

4.2 If the visitor chooses to add 1 then the quantity of the item is increased by 1 and if the visitor chooses to remove 1 the quantity is decreased by 1.

Acceptance testing:

1. Good scenario: when a visitor is trying to view his cart's content the user object can successfully load all the cart's content and present it to the visitor.
2. Bad scenario1: when a visitor is trying to view his cart's content the user object checks and sees that the visitor's cart is empty, so he informs the visitor that there is nothing to present.
3. Bad scenario2: when a visitor is trying to remove 1 from an item's quantity it reaches 0 so the user object removes the item from the visitor's cart (and if now some store's basket is empty, it is deleted as well).
4. Bad scenario3: when a visitor is trying to add 1 to an item's quantity the user object sends a request to the store to check if the needed quantity is in stock, if there isn't enough in stock then the user object informs the visitor and doesn't allow to add 1 to this item's quantity.

9. Use case: buying and paying

-Actor: Visitor.

-Precondition: The visitor already has the app/web page of the system open, and the system is already connected to a payment system.

-Parameter: None.

-Actions:

1. The visitor chooses to start the buying process of his cart.

2. the user object first checks that all the item's quantities in the cart are still available.

2.1 if not then the user object notifies the visitor the purchase cannot be made because of those items and that he should remove them if he wants to continue the purchase.

2.2 otherwise the user object sends the visitor a message containing the total price of the order and a list of all the items in the order (with their prices).

3. the visitor checks and sees everything is ok with the order and chooses to continue with the process.

4. the market system asks the visitor to enter his payment preferred option (credit card number or other methods of payment).

5. according to the visitor's choice the market system points to the right payment system and transfers his information.

5.1 If the information is false the payment system will send an error message that will pass to the visitor through the market system.

5.2 Otherwise, the transaction will go through and be approved by the payment system and the market system will send a receipt to the visitor's email.

Acceptance testing:

1. Good scenario: when a visitor is trying to purchase his cart's content the system can successfully make the transaction and send a receipt to the visitor.
2. Bad scenario1: when a visitor is trying to purchase his cart's content the payment system returns an error, so the system cancels the transaction and informs the visitor.

3. Bad scenario2: when a visitor is trying to purchase his cart's content the user object detects that the purchase cannot be made because some of the items are not in stock.
4. Bad scenario3: when a visitor is trying to purchase his cart's content the user object detects that the cart is empty and informs the visitor about it.
5. Bad scenario4: when a visitor is trying to purchase his cart's content the user object detects that store that is part of the purchase doesn't exist anymore and informs the visitor.

Registered User:

Use case : Exit Market.

- **Actor:** User
- **Pre-condition:** User is logged in.
- **Parameter:** None.
- **Action:**
  1. User selects the "Exit Market".
  2. The system receives the input and triggers the logout sequence. (See logout use-case below (2, 2.a 2.b))

3. The system presents the user with "Exit successful" and exits the market.

Acceptance Tests:

- Good: The user selects "Exit Market", and the system triggers the logout sequence, the user is then presented with an "Exit successful" message.
- Bad: The user is not logged in but is still presented with an "Exit successful" message.

Use case : Logout.

- **Actor:** User
- **Pre-condition:** User is logged in.
- **Parameter:** None.
- **Action:**
  1. User selects logout.
  2. The system receives the input and handles the logout:
     1. The logout handler removes all logged in permissions from the user.
     2. The logout handler saves all data regarding shopping cart and baskets to the database.
  3. The system presents the user with "Logout successful".

Acceptance Tests:

- Good: The user selects the logout button, and the system saved all data correctly.
- Bad: The user selects the logout button, but the system is treating him as a guest and 1. not saving all relevant information or 2. the system doesn't allow him to logout(when it should have).

Use case : Opening a store.

- **Actor:** User
- **Pre-condition:** User is logged in and chose the open store option.
- **Parameter:** None.
- **Action:**
  1. The system requests for some information about the store.
  2. The user inputs the information (such as the stores' name and description).
  3. The system validates the information.
     1. If **false**:
        a.1. The system presents the user with an error message describing what's
        wrong.
     b. If **true**:
        b.1. The Store control identifies the store with a unique ID number and sets it in the store.
        b.2. The system presents the user with "Shop has been created" message.
        b.3. The system saves all store data in the systems' data base.

Acceptance Tests:

- Good: The user tried to open a store, the system identifies the store with a unique ID and presents the user with "Shop has been created", all store data is saved in the database.
- Bad: Good: The user tried to open a store, but the information is wrong,
  the system presents the user with an error
  message.

Store Owner:

Use Cases - 4.1: Inventory management.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents an option to manage the inventory in the store **s**.
  2. User selects the inventory management option.
  3. System presents options of adding and removing products, and changing their details.
  4. If the user selects a add product. → $Usecase\ 4.1.1$
  5. If the user selects a removing product→ $Usecase\ 4.1.2$
  6. If the user selects a changing product details→ $Usecase\ 4.1.3$
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and selects one of the options.
       Excepted Output: The system will present the next screen by the option which the user selects.
  - **Negative:**
    1. **Scenario:** The user isn't logged in to the trading system. He is the owner of store s (the store s is open) and selects one of the options.
       Excepted Output: The system will present a login screen.
    2. **Scenario:** The user is logged in to the trading system. He isn't the owner of store s (the store s is open) and selects one of the options.
       Excepted Output: The system will return alert that the user isn't owner of store s.

Use Cases - 4.1.1: Inventory management – adding product.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents the possibility of adding products with all the necessary fields to be filled in regarding the product.
  2. User enters the product data and adds a product.
  3. System processes the request to update the product (Check user logged in as owner of store **s**, and store **s** doesn't existed and the store **s** is open).
  4. If the addition of the product is possible.
     a. System adds the product to the store inventory.
     b. System updates the user that the product has been added to the store's inventory.
  5. If adding the product is not possible.
     a. System updates that the product cannot be added and returns the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he enter all the necessary fields of the product and add the product.
       Excepted Output: The system will return that the product is added.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he enter the necessary fields without one necessary field of the product and add the product.
       Excepted Output: The system will return that the product isn't added and the reason.
    2. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he enter all the necessary fields but the product already exists in the store of the product and add the product.
       Excepted Output: The system will return that the product isn't added and the reason.

Use Cases - 4.1.2: Inventory management – remove product.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents the list of products in store's inventory.
  2. User selects from the list a product that he wishes to remove from the inventory of the products in the store **s**.
  3. System processes the request to remove the product (Check user logged in as owner of store **s**, , store **s** is existed and the store **s** is open and if the product is exists).
  4. If the remove of the product is possible.
     a. System removes the product from the store **s** inventory.
     b. System updates the user that the product has been removed to the store's inventory.
  5. If remove of the product is not possible.
     a. System updates that the product cannot be removed and returns the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he selects one product from the list to remove and remove the product.
       Excepted Output: The system will return that the product is removed.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he try to remove product that doesn't exist in the store s.
       Excepted Output: The system will return that the product isn't removed and the reason.

Use Cases - 4.1.3: Inventory management – Changing product details.

- **Actor:** User
- **Pre-condition:**
  5. User is logged in.
  6. The user is a store owner of store **s**.
  7. The Trading system is initialized.
  8. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents the list of products in store's inventory.
  2. User selects from the list a product for which the user wants to change the details.
  3. System presents an option to update the details of the product with all the data fields regarding the product.
  4. User enters the product details and updates the product.
  5. System processes the request to update the product (Check user logged in as owner of store **s**, and store **s** is existed and the store **s** is open and the product exist and able to change).
  6. If the update of the product is possible.
     b. System updates the product to the store inventory.
     c. System updates the user that the product has been updated.
  7. If the update of the product is not possible.
     d. System updates that the product cannot be added and returns the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he update all the necessary fields of the product and update the product.
       Excepted Output: The system will return that the product is updated.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he enter invalid input field of the product and update the product.
       Excepted Output: The system will return that the product isn't updated and the reason.

Use Cases - 4.2: Changing the types and rules (policy) of the store's purchase and discount.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System shows the possibility of changing the purchase and discount policy.
  2. User chooses which policy he wants to change.
  3. If user select changing the purchase policy.
     a. System presents to user with a selection option for the various purchase policies with a description for each one.
     b. User selects the policy for the store.
     c. System defines for the store the policy that the user defined.
     d. System updates the employees in the store about the update of the policy.
  4. If user select changing the discount policy.
     a. System presents to user with a selection option for the various discount policies with a description for each one.
     b. User selects the policy for the store.
     c. System defines for the store the policy that the user defined.
     d. System updates the employees in the store about the update of the policy.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he changed policy of the stores purchase.
       Excepted Output: The system will return that the policy is changed.
    2. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he changed policy of the stores discount.
       Excepted Output: The system will return that the policy is changed.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He isn’t the owner of store s (the store s is open) and he changed policy purchase or discount of the stores.
       Excepted Output: The system will return an alert that the user isn’t the owner of store s.

Use Cases - 4.4: Appointment of store owner.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents an option to search for users registered in the system.
  2. User enters in the search bar the name of the registered user he wants to appoint to the store owner.
  3. System presents a pool of registered users that the user has searched for.
  4. User selects from the pool the user **u** he wants to appoint as the owner of the store.
  5. System will check if the user is registered and not the owner of the store already.
  6. If the appointment is possible,
     a. System that will define the user **u** as the owner of the store **s**.
     b. System will grant permissions to the user **u** as the store owner of the store **s**.
     c. System will send the user **u** a message about the appointment.
     d. System will return a message to the user that the appointment was successfully made.
  7. If the appointment is not possible,
     a. System will return a message to the user that the appointment is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he select user from the list to be new owner.
       Excepted Output: The system will return that the appointment done and send alert about the appointment.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he selects to appointing user that already owner of store s to be store owner.
       Excepted Output: The system will return that the appointment failed with the reason.
    2. **Scenario:** The user is logged in to the trading system. He isn't the owner of store s (the store s is open) and selects to appoint new owner to store s.
       Excepted Output: The system will return alert that the user isn't owner of store s.

Use Cases - 4.6: Appointment of store manager.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. System presents an option to search for users registered in the system.
  2. User enters in the search bar the name of the registered user he wants to appoint to the store manager.
  3. System presents a pool of registered users that the user has searched for.
  4. User selects from the pool the user **u** he wants to appoint as the manager of the store.
  5. System will check if the user **u** is registered and not the owner or manager of the store already.
  6. If the appointment is possible,
     a. System that will define the user **u** as the manager of the store **s**.
     b. System will grant permissions (Getting of information) to the user **u** as the store manager of the store **s**.
     c. System will send the user **u** a message about the appointment.
     d. System will return a message to the user that the appointment was successfully made.
  7. If the appointment is not possible,
     a. System will return a message to the user that the appointment is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he select user from the list to be new manager in store s.
       Excepted Output: The system will return that the appointment done and send alert about the appointment.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he selects to appointing user that already owner or manager of store s to be store manager.
       Excepted Output: The system will return that the appointment failed with the reason.
    2. **Scenario:** The user is logged in to the trading system. He isn't the owner of store s (the store s is open) and selects to appoint new owner to store s.
       Excepted Output: The system will return an alert that the user isn't owner of store s.
    3. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he select user from the list to be new manager in store s.
       Excepted Output: The system will return that the appointment done but don't send alert about the appointment.

Use Cases - 4.7: Changing store manager's permissions.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
  5. **m** is store manager of store **s**.
- **Parameter:** Store **s** and store manager **m**.
- **Actions:**
  1. User requests to manage the **m** manager's permissions.
  2. System presents a list of employees in the store **s**.
  3. System presents the permissions that the manager has and the permissions that are not set for him.
  4. User decides which permissions to add and which to cancel and saves changes.
  5. System checks the request (Check user logged in as owner of store **s**, and **m** is manager of store s, store **s** is existed and the store **s** is open).
  6. If the request is possible,
     a. System makes the changes to the permissions.
     b. System sends an alert to the store manager **m** about the changes in his permissions.
  7. If the request is not possible,
     a. System will return a message to the user that the request to changes **m** permissions' is not possible with the reason.

- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he select user who manger in the store s and change his permissions.
       Excepted Output: The system will return that the permission change and update the user about the changes.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he selects user who isn't manger in the store s and change his permissions.
       Excepted Output: The system will return that the change of the permissions of this user cancel.
    2. **Scenario:** The user is logged in to the trading system. He isn't the owner of store s (the store s is open) and change permissions of store manager in store s.
       Excepted Output: The system will return an alert that the user isn't owner of store s.
    3. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he select user who manger in the store s and change his permissions.
       Excepted Output: The system will return that the permission change and update the user about the changes, but don't alert to the manager about the changes.

Use Cases - 4.9: Closing store.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store creator of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. User (store creator) submits a request to close the store s.
  2. System checks the request (Check user logged in as store creator of store **s**, store **s** is existed and the store **s** is open).
  3. If the request is possible,
     a. System will close the store.
     b. System will define the store as inactive and does not allow access to receive information for users who are not managers or owners of store **s**.
     c. System will send a notification to store owners and managers about the store's closing.
  4. If the request is not possible,
     a. System will return a message to the user that the request to close the store is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the creator of store s (the store s is open) and close the store.
       Excepted Output: The system will return that the store close and send about the closed message to the employees in the store.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is the creator of store s (the store s is already closed) and close the store.
       Excepted Output: The system will return that the store already closed.
    2. **Scenario:** The user is logged in to the trading system. He isn't the creator of store s (the store s is open) and he try close the store.
       Excepted Output: The system will return an alert that the user isn't creator of store s.
    3. **Scenario:** The user is logged in to the trading system. He is the creator of store s (the store s is open) and close the store.
       Excepted Output: The system will return that the store close but don't send about the closed message to the employees in the store.

Use Cases - 4.11: Request for information about positions in the store.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. User asks to receive information about the positions in the store **s**.
  2. System checks the request (Check user logged in as admin, store **s** is existed, and the user have the permission to ask this request).
  3. If the request is possible,
     a. System presents to the user a list of the store's employees with their position in store **s**.
     b. System presents an option to get information about permissions for the managers of the store **s**.
     c. If the user chooses to receive information about the store manager's permissions in store **s**. $\rightarrow Usecase\ 4.11.1$
  4. If the request is not possible,
     a. System will return a message to the user that the request is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store **s** (the store s is open) and he requests information about positions in the store.
       Excepted Output: The system will return the information which the user requested.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He isn't the owner of store **s** (the store s is open) and he requested information about positions in the store.
       Excepted Output: The system will return an alert that the user isn't owner of store **s**.

Use Cases - 4.11.1: Request for information about store manager's permissions.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a store owner of store **s**.
  3. The Trading system is initialized.
  4. The user **u** manager in store **s**.
  5. Store **s** is existing and open.
- **Parameter:** Store **s** and user **u**.
- **Actions:**
  1. User requests to receive information about store manager **u** permissions in store **s**.
  2. System checks the request (Check user logged in as admin, store **s** and user **u** are existed, and the user have the permission to ask this request).
  3. If the request is possible,
     a. System gets from the DB the data about the purchases history of buyer **b**.
     b. System presents the permissions of the store manager **u** in the store **s**.
  4. If the request is not possible,
     a. System will return a message to the user that the request is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he requests information about store manager permissions.
       Excepted Output: The system will return the information which the user requested.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He isn't the owner of store s (the store s is open) and he request for information about store manager permissions.
       Excepted Output: The system will return an alert that the user isn't the owner of store s.
    2. **Scenario:** The user is logged in to the trading system. He is the owner of store s (the store s is open) and he requests information about store manager permissions, but this employee isn't manager in store s.
       Excepted Output: The system will return an alert that the user isn't manager in store s.

Use Cases - 4.13: Receiving information about purchase history in the store.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is a admin or store owner or manager of store **s**.
  3. The Trading system is initialized.
  4. Store **s** is existing and open.
- **Parameter:** Store **s**.
- **Actions:**
  1. User requests purchase history in the store **s**.
  2. System checks the request (Check user logged in as admin, store **s** is existed, and the user have the permission to ask this request).
  3. If the request is possible,
     a. System gets from the DB the data about the purchases history of store **s**.
     b. System presents a list of the purchases history to the user.
  4. If the request is not possible,
     a. System will return a message to the user that the request is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is admin, store owner or manager of store **s** (the store s is open) and he request for information about purchase history in the store.
       Excepted Output: The system will return the information which the user requested.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He isn't admin, store owner or manager of store **s** (the store s is open) and he request for information about purchase history in the store.
       Excepted Output: The system will return an alert that the user isn't system administrator or store owner or manager of store **s**.

System Administrator:

Use Cases - 6.4: Getting information about purchase history of the store or buyers.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. User is an admin.
  3. The Trading system is initialized.
- **Parameter:** None.
- **Actions:**
  1. User asks to see purchase history.
  2. System presents an option to search for the buyer or the store for which the user would like to see the purchase history.
  3. User enters the name of the buyer or store for search.
  4. System presents a list of buyers and stores related to what the user entered in the search.
  5. User selects from the list the store or buyer whose purchases history he wants to see.
  6. If the user selects a buyer → $Usecase\ 6.4.1$
  7. If the user selects a store → $Usecase\ 4.13$
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is an admin and selects one of the options.
       Excepted Output: The system will present the next screen by the option which the user selects.
  - **Negative:**
    1. **Scenario:** The user isn't logged in to the trading system. He is an admin and selects one of the options.
       Excepted Output: The system will present a login screen.
    2. **Scenario:** The user is logged in to the trading system. He isn't the admin and selects one of the options.
       Excepted Output: The system will return an alert that the user isn't admin.

Use Cases - 6.4.1: Getting information about purchase history of buyers.

- **Actor:** User
- **Pre-condition:**
  1. User is logged in.
  2. The user is an admin.
  3. The Trading system is initialized.
  4. Buyer **b** is existing.
- **Parameter:** Buyer **b**.
- **Actions:**
  1. User requests purchase history of buyer **b**.
  2. System checks the request (Check user logged in as admin, buyer b is exist and the user have the permission to ask this request).
  3. If the request is possible,
     a. System gets from the DB the data about the purchases history of buyer **b**.
     b. System presents a list of the purchases history of buyer **b** to the user.
  4. If the request is not possible,
     a. System will return a message to the user that the request is not possible with the reason.
- **Acceptance Test:**
  - **Positive:**
    1. **Scenario:** The user is logged in to the trading system. He is admin and he requests information about purchases history of buyer.
       Excepted Output: The system will return the information which the user requested.
  - **Negative:**
    1. **Scenario:** The user is logged in to the trading system. He is admin and he requests information about purchase history in the buyer who don't register to the system.
       Excepted Output: The system will return an alert that the buyer doesn't exist in the system.