Ariel: note – any action a shop owner can do, shop founder can perform as well

* Use Case: Real-Time Notification

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

**Note:** In some cases, the system will initiate the process.

1. 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).
2. The System checks if the selected users are logged into the system.
3. The system will immediately present the notification and it’s context to the user (receiving).
4. Expected result: The notification will pop up for the receivers of the notification to see and review.

* 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 Shop owner selects a manager which he’d like to change his permissions.
6. The system checks if the Owner and Manager are both associated with the same shop.
7. The system will present the actor to permissions options he has to choose from.
8. The shop owner selects the new permissions for the shop manager.
   1. Real-Time Notification will start. The Shop Manager is notified his permissions were changed.
   2. In case Real-Time Notification failed:

Delayed Notification will start.

1. Expected result: The selected manger’s permission will be changed effective immediately.

* Use Case: Request information on shop’s officials.

1. Actor: Shop Owner
2. Precondition: The shop owner and the officials must be associated with the same shop.
3. Parameters: Shop’s officials.
4. Actions:
5. The shop owner selects which official’s information he’d like to see.
6. The system checks if the Owner and the official are associated with the same shop.
7. Expected result: The system will display the relevant information of the selected shop’s officials.

* 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 verifies the Founder is associated with the same shop.
7. 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).**
8. Expected result: The shop, it’s information, products and any other information will be unavailable to all the users except for the shop’s officials and the system managers.

* Use Case: 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. The system validates the following parameters in case they were given:
   1. Checks if date is valid.
   2. Checks if product exists.
7. 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.

|  |  |  |
| --- | --- | --- |
| Use-Case | Parameter | Expected Output |
| Real-Time Notification | A User sends a notification to a logged in user. The second user receives the notification and its displayed. | Success |
| A User sends a notification to a shop. All the shop’s relevant logged in officials receive the notification and it’s displayed to them. | Success |
| A User sends a notification to a user who is not logged in. The second user does receive the notification. | Success |
| A User sends a notification with no text to a logged in user. The second user does not receive the notification and the system displayed an error message. | Success |
| A User sends a notification to a logged in user. The second user does not receive the notification or its displayed. | Fail |
| A User sends a notification to a logged in user. The notification was sent to an unknown third user. | Fail |
| The user was able to send a notification with no text. | Fail |
| The user sends a notification to a shop. It is not received by any of the shop’s officials. | Fail |
| Change Shop Manager’s Permissions | The changes the shop owner made for the shop manager’s permissions are changed accordingly. | Success |
|  | The shop owner removed all permissions from the shop manager. He is still a shop manager with no permissions. | Success |
| The changes the shop owner made were applied to all the shop managers of the store. | Fail |
| Owner made changes and the manager was not notified. | Fail |
| Request Information on Shop’s Officials | The system displays the correct information of the selected officials. | Success |
|  | The system displays nothing if no official was selected. | Success |
| The system displayed sensitive information or information which was not supposed to be displayed (such as login ID or User’s password). | Fail |
| The system does not display the selected official’s information. | Fail |
| Close Shop | Shop founder requests to close his shop. The system does not display the shop, it’s products and its information to other users except for the shop’s officials and the system managers. | Success |
|  | A user searches for a closed shop. The system does not display the shop. | Success |
|  | A user search for a product which is sold by a closed shop. The system does not display the closed shop sells that product. | Success |
|  | The system has successfully closed the shop, but the system managers and the shop’s officials are not able to access the closed shop. | Fail |
|  | A user searches for a closed shop. The system displays the shop. | Fail |
|  | The products of the closed shop are displayed upon search. | Fail |
| Upon closing the shop, the shop remains open. | Fail |
| No notifications were sent to the store’s officials upon closing the shop. | Fail |
| Request Information of shop’s sales history | The system displays the shop’s sales history accordingly to the given filter. | Success |
|  | The system displays sales of products which are not in the shop’s inventory anymore when requested to. | Success |
|  | The System deletes or ignores sales of products which are removed from the shop’s inventory and will not display it upon request. | Fail |
| The system does not display the requested filtered sales. | Fail |
| The system displays all sales in all shops with similar products. | Fail |

// Write a function which implements case-insensitive

// lexicographical comparison of two strings

// (also known as alphabetical compare)

// a == b ⇒ return 0 ("HELLO"=="hello")

// a > b ⇒ return >0 ("banana">"apple")

// a < b ⇒ return <0 ("lemon"<"mango")

public int compareTask(String a, String b)

{

int minSize = Math.Min(a.length(), b.length());

for(int i = 0; i<minSize; i++)

{

if(a.toLowerCase()[i] < b.toLowerCase()[i])

return -1;

if(a.toLowerCase()[i] > b.toLowerCase()[i])

return 1;

}

return a.lenght() - b.length();

}

// Implement class Deferred

Deferred d = new Deferred();

d.then(res->{

   System.out.println("1 "+res);

   Deferred d1 = new Deferred();

   new Timer().schedule(new TimerTask() {

       @Override

       public void run() { d1.resolve("a"); }

   }, 1500);

   return d1;

});

d.then(res->{System.out.println("2 "+res) ; return "b";});

d.then(res->{System.out.println("3 "+res); return "c";});

d.resolve("hello");

// output of usage example

1 hello

// 1.5s later.. the rest should be printed

2 a

3 b

// =============

public Class Deferred()

{

private List<Function<String,Object>> lsOfFun;

private Object lastRet;

public Deferred()

{

lsOfFun = new ArrayList<Function<String,Object>>();

lastRet = null;

}

public void then(Function<String, Object> f)

{

lsOfFun.add(f);

}

public void resolve(String s)

{

Object ret = s;

for(Function<String,Object> f : lsOfFun)

{

if(ret instanceof String)

{

ret = f.apply(ret);

lastRet = ret;

}

else if(ret instanceof Deferred)

{

ret = f.apply(ret.resolve());

lastRet = ret;

}

}

}

public Object resolve()

{

return lastRet;

}

}