Use-Cases

System

Use-case 1.1: Initialize the Market

1. **Actor:** Main system
2. **Precondition:** There is no initialized market
3. **Parameters:** Payment service, supply service
4. **Actions:**
    - Open a new Market
    - Connecting to payment and supply services
    - Create a user which will be the admin

Action	Data	Expected Result
Initialize the Market	**payment service, supply service (valid)**	**Market creates successfully**
	payment service, supply service (invalid)	**Displaying an error massage**

Use-case 1.2.1: Add external service

1. **Actor:** User
2. **Precondition:** User is an admin, market is not connected to the external service
3. **Parameters:** external service to be added
4. **Actions:**
    - Admin adding a new external service to the market(system)
    - System validates the external service to be added
    - The system returns an error/success message

Action	Data	Expected Result
Adding new external service	**Market, external service to be added (valid)**	**External service added successfully**
	Market, external service to be added (invalid)	**Fails to add an external service, displaying an error message**

Use-case 1.2.2: Remove external service

1. **Actor:** User
2. **Precondition:** User is an admin, market is connected to the external service
3. **Parameters:** external service to be removed
4. **Actions:**
    - Admin asks the system to remove an external service
    - The system verifies that the external service to be removed is not a mandatory service
    - The system returns an error/success message

Action	Data	Expected Result
Adding new external service	**Market, external service to be removed (optional service)**	**External service removed successfully**
	Market, external service to be removed (mandatory service)	**Fails to remove the external service, displaying an error message**

Use-case 1.2.3: Replace external service

1. **Actor:** User
2. **Precondition:** User is an admin, market is connected to the external service
3. **Parameters:** External service to be replaced, external service to be added instead
4. **Actions:**
    - Admin asks the system to remove the external service to be removed (1.2.2)
    - Admin asks the system to add the external service to be added (1.2.1)
    - The system returns an error/success message

Action	Data	Expected Result
Replace external service	**Market, external service to be replaced, External service to be added instead (valid)**	**External service replaced successfully**
	Market, external service to be replaced, External service to be added instead (invalid)	**Fails to remove/add the external service, displaying an error message**

Use-case 1.3: Payment

1. **Actor:** Market
2. **Precondition:** Payment service exists
3. **Parameters:** Transaction details
4. **Actions:**
    - Market getting a request for payment from a user in the system
    - Market connecting to payment service
    - Market makes the payment via the payment service
    - The market getting an answer and pass it to the actor which request the payment

Action	Data	Expected Result
Payment	**Correct transaction details**	**Payment done successfully – returning a Confirmation message**
	Incorrect transaction details	**Payment denied – returning a Refusal message**
	Correct transaction details (There is no payment service)	**Failed to connect to payment service, system crashed**

Use-case 1.4: Supply

1. **Actor:** Market
2. **Precondition:** Supply service exists
3. **Parameters:** Package details, costumer details
4. **Actions:**
    - Market getting a request for supply from a user in the system
    - Market connecting to supply service
    - Market performs the delivery process via the supply service
    - The market getting an answer and pass it to the actor which request the payment

Action	Data	Expected Result
Supply	**Correct Package details, correct costumer details**	**Supply done successfully – returning a Confirmation message**
	Costumer does not exist	**Supply denied – returning a Refusal message**
	Correct Package details, correct costumer details (There is no supply service)	**Failed to connect to supply service**

Use-case 1.5: Real-time alerts

1. **Actor:** Market
2. **Precondition:** Market has been initialized
3. **Parameters:** Destination user details (store manager, registered user, etc)
4. **Actions:**
    - After an event happening, market is connecting to Notification Handler internal service
    - Market sends the destination user details and the notification context
    - The users will check their "Inquiry box" for an unread message
    - The destination user will answer the alert if needed at the same way

Action	Data	Expected Result
Customer purchase a product from a store	**Store manager id, notification context**	**Store manager received the massage successfully**
Store is closes	**Store manager is not existing**	**Failed to deliver the alert**
A request has been applied	**Correct destination user details, a massage describes the event (there is no notification handler)**	**Notification Handler has not been initialized**

Use-case 1.6: Delayed alerts

1. **Actor:** System
2. **Precondition:** Market has been initialized
3. **Parameters:** Destination details (store manager, registered user, etc)
4. **Actions:**
    - After an event happening, connect to Notification Handler object
    - Market sends the destination user details and the notification context
    - If the destination alert user is not sign in, market is adding the massage to his delayed massages queue
    - When user will sign in, the alert will be showed to him

Action	Data	Expected Result
Customer purchase a product from a store	**Store manager id, notification context**	**adding the massage to Store manager's delayed massages queue**
Store is closes	**Store manager is not existing**	**Failed to deliver the alert**
A request has been applied	**Correct destination details, a massage describes the event (there is no notification handler)**	**Notification Handler has not been initialized**

Users

Use-case 2.1.1: Visitor – Entrance

1. **Actor:** User
2. **Precondition:** Market has been initialized
3. **Parameters:** None
4. **Actions:**
    - User enter the market, defined as a visitor
    - Visitor gets a shopping cart, can buy from the market

Action	Data	Expected Result
Visitor – Entrance	**None**	**User enter the market successfully and become a visitor**
	None	**Market has not been initialized**

Use-case 2.1.2: Visitor - Exit

1. **Actor:** User
2. **Precondition:** User is a visitor
3. **Parameters:** None
4. **Actions:**
    - The user(visitor) exits the market
    - Visitor's shopping cart is gone

Action	Data	Expected Result
Visitor - Exit	**None**	**Visitor exit the market successfully**

Use-case 2.1.3: Visitor - Register

1. **Actor:** User
2. **Precondition:** User is a visitor
3. **Parameters**: User name, password
4. **Actions:**
    - Visitor registers to the system with his credentials
    - The system checks visitor's credentials meet the constraints
    - If credentials are correct, the visitor become assigned user
    - If credentials are incorrect, an error message sent to the visitor.
    - Register failed

Action	Data	Expected Result
Visitor - Register	**User name, password (correct credentials)**	**Visitor become an assigned user successfully**
	User name, password (incorrect credentials)	**Register failed. Visitor received an error massage**

Use-case 2.1.4: Visitor - Log-In

1. **Actor:** User
2. **Precondition:** User is a visitor
3. **Parameters**: User name, password
4. **Actions:**
    - Visitor Log-In the system with his credentials
    - The system checks that the credentials are correct (the same credentials provided by the user when registering to the system)
    - If credentials are correct, the visitor become assigned user
    - If credentials are incorrect, an error message sent to the visitor.
    - Log-In failed

Action	Data	Expected Result
Visitor - Log-In	**User name, password (correct credentials)**	**Visitor become an assigned user successfully**
	User name, password (incorrect credentials)	**Log-In failed, visitor gets an error massage**

Use-case 2.2.1: Getting information about market's store.

1. **Actor:** User
2. **Precondition:** User is a visitor, the market contains at least one store.
3. **Parameters:** Store id.
4. **Actions:**
    - The system searching the store in the market by a store id.
    - Market display to the user the store information.

Action	Data	Expected Result
Getting information about the store.	**Store id-for a store that exist.**	**Displaying The store information.**
	Store id-for a store that not exist.	**Displaying failure massage.**

Use-case 2.2.2: Searching a product with parameters and filters.

1. **Actor:** User
2. **Precondition:** User is a visitor, the market contains at least one product.
3. **Parameters:** product id, filters.
4. **Actions:** The system scanning the market for products that meets the filters and parameters.

Action	Data	Expected Result
Scanning the market for products that meets the filters and parameters.	**There is at least one product which answer all parameters and filters.**	**List of products.**
	There is no product which answer all parameters.	**Displaying an appropriate message.**

Use-case 2.2.3: Saving products into a shopping basket.

1. **Actor:** User
2. **Precondition:** User is a visitor, there is at least one store in the market with at least one product
3. **Parameters:** product id, store id
4. **Actions:**
    - The user is choosing the desired product or products to save.
    - The system checks if the user has a shopping basket for specific store
    - If the user doesn’t have a shopping basket, the system will open a new one
    - The system is saving the products to the shopping basket.

Action	Data	Expected Result
User saving a product or a group of products to the shopping cart.	**Product and store id (valid).**	**The product added to the user's shopping basket.**
	Product and store id (invalid).	**Displaying an error message.**

Use-case 2.2.4A: Displaying shopping cart content

1. **Actor:** User
2. **Precondition:** User is a visitor, user has shopping cart
3. **Parameters:** None
4. **Actions:**
    - Getting the user's shopping cart by the system
    - The system display cart content to the user

Action	Data	Expected Result
Displaying cart contents.	**User has a shopping cart object(valid)**	**Displaying cart content.**
	User does not have a shopping cart object	**Displaying an appropriate error message**

Use-case 2.2.4B: Making changes to the shopping cart

1. **Actor:** User
2. **Precondition:** User is a visitor, user has shopping cart
3. **Parameters:** product id, action to do on the product
4. **Actions:**
    - Getting the user's shopping cart by the system
    - The system letting the user to change cart content

Action	Data	Expected Result
changing cart contents.	**Action, Product id(valid)**	**Changing the cart content**
	Action, Product id(invalid)	**Displaying an appropriate error message**
	User does not have a shopping cart	**Failed to get the user's shopping cart**

Use-case 2.2.5: Buying the shopping cart content.

1. **Actor:** User
2. **Precondition:** User is a visitor, user has a shopping cart with at least one shopping basket, valid payment service, valid supply service
3. **Parameters:** payment service, supply service
4. **Actions:**
    - Getting the user's shopping cart by the system
    - The system verifies the shopping cart contains at least one non empty shopping basket
    - The system is connecting to payment service in order to pay for the products.
    - The system is connecting to supply service in order to ship the products.

Action	Data	Expected Result
Shipping with supply service	**Shipping data(valid).**	**Displaying a confirmation supply message**
	Shipping data(invalid).	**Displaying an appropriate error message**
Paying with the payment service.	**Payment data(valid).**	**Displaying a confirmation payment message**
	Payment data (invalid)	**Displaying an appropriate error message**

Use-case 2.3.1: Logout

1. **Actor:** User
2. **Precondition:** User is connected to the system.
3. **Parameters:** None
4. **Actions:**
    - The system is saving the user state and his shopping cart to the DB
    - The system disconnects the user from the system.

Action	Data	Expected Result
Logout	**User's state and shopping cart (valid)**	**User's state saved to DB. User disconnected from the system**
	User is not connected to the system	**Displaying an appropriate error message**

Use-case 2.3.2: Open a store

1. **Actor:** User
2. **Precondition:** User is connected to the system, there is no other store with the same id
3. **Parameters**: store parameters (name, type, etc)
4. **Actions:**
    - The system creates a new store in the market
    - The market defines the store opener to be the first owner

Action	Data	Expected Result
Opening a store	**Store parameters(valid)**	**Store is added to the market, the opener become store owner**
	Store parameters(invalid)	**Displaying an appropriate error message**
	User is not connected to the system	**Displaying an appropriate error message**

Use-case 2.3.3: Adding a review about products

1. **Actor:** User
2. **Precondition:** User is connected to the system, product exists , the user bought the product
3. **Parameters:** review content, product id
4. **Actions:**
    - The system creates a new review with the review content
    - The system gets the store of the product
    - The system is adding the review to the store review queue

Action	Data	Expected Result
Adding a review about products	**All of the mandatory fields for a review are full and valid**	**Adding a review to the store review queue and displaying a confirmation massage.**
	At least one of the mandatory fields is invalid.	**Displaying an appropriate error message.**
	User is not connected to the system	**Displaying an appropriate error message**

Use-case 2.3.4: Adding a rating for a store or a product.

1. **Actor:** User
2. **Precondition:** User is connected to the system, store or product you want to rate is exists.
3. **Parameters**: rating content, product or store id
4. **Actions:**
    - The system creates a new rating with the rating content
    - The system gets the desired store or product
    - The system is adding the rating to the store or product ratings queue

Action	Data	Expected Result
Adding a rating for a store or a product	**Rating data(valid)**	**Adding the rating to the rating database.**
	Rating data(invalid)	**Displaying an appropriate error message.**
	User is not connected to the system	**Displaying an appropriate error message**

Use-case 2.3.5: Sending question or requests to a store

1. **Actor:** User
2. **Precondition:** User is connected to the system, the store you want to send the question to exists, request handler object exists.
3. **Parameters:** store id, question or request.
4. **Actions:**
    - The system creates a new request/question with the data given by the user.
    - The system sends the request/question to the request handler.
    - The request handler is sending the request/question to the store.

Action	Data	Expected Result
Sending question or requests to a store.	**All of the mandatory field for a review are full and valid**	**Sending the object to the store.**
	At least one parameter for the object contractor is not valid.	**Displaying an appropriate error message.**
	User is not connected to the system	**Displaying an appropriate error message.**

Use-case 2.3.6: Sending a compliant about integrity issues.

1. **Actor:** User
2. **Precondition**: User is connected to the system, the user has made a purchase
3. **Parameters:** store id, compliant content
4. **Actions:**
    - The system is connected to request handler
    - The system is adding the complaint to the admin compliant list via the request handler

Action	Data	Expected Result
Sending a compliant about integrity issues	**Compliant data(valid), store id(valid)**	**Adding the complaint to the admin compliant list**
	Compliant data or store id is invalid	**Displaying an appropriate error message.**
	User is not connected to the system	**Displaying an appropriate error message.**

Use-case 2.3.7: Getting information about user purchase history

1. **Actor:** User
2. **Precondition:** User is connected to the system, user has a purchase history
3. **Parameters:** None
4. **Actions:** The system display the purchase history.

Action	Data	Expected Result
Displaying the purchase history	**None**	**Displaying the user purchase history.**
	User does not have a purchase history	**Operation failed, nothing is showed to the user**
	User is not connected to the system	**Displaying an appropriate error message**

Use-case 2.3.8A: Displaying user information

1. **Actor:** User
2. **Precondition:** User is connected to the system
3. **Parameters:** None
4. **Actions:** The system display user information

Action	Data	Expected Result
Getting user information	**None**	**Displaying user information**
	User is not connected to the system	**Displaying an appropriate error massage.**
	User is not connected to the system	**Displaying an appropriate error massage**

Use-case 2.3.8B: Changing user information

1. **Actor:** User
2. **Precondition:** User is connected to the system
3. **Parameters:** new user information.
4. **Action:**
   The system is changing the user information.

Action	Data	Expected Result
Changing user information	**All of the new fields are valid**	**Replacing the old information with the new one.**
	There is at least one invalid field.	**Displaying an appropriate error massage.**
	User is not connected to the system	**Displaying an appropriate error massage**

Use-case 2.3.9: Update user subscription security.

1. **Actor:** User
2. **Precondition:** User is connected to the system, user has subscription security
3. **Parameters:** new security measure
4. **Actions:**
    - The system checks if the new security measure is valid
    - The system checks if user defined subscription security
    - The system updates user's subscription security

Action	Data	Expected Result
Update subscription security	**New security measure (valid)**	**Update the user subscription security.**
	New security measure (invalid)	**Displaying an appropriate error massage.**
	User is not connected to the system	**Displaying an appropriate error massage**

Use-case 2.4.1.1 : Adding an item to inventory

1. **Actor:** user
2. **Precondition:** The user is a store owner,, the user is the owner of this specific store, the user is logged in, the item is not already in the store's inventory
3. **Parameters :** Store id, item details
4. **Actions :**
    - The user adds new item with all the details
    - The system verify that the user is the store owner
    - The system verify that the user is logged in
    - The system verify correctness of the item details
    - The system generate unique id for the item
    - The user gets a response according to success or failure

Action	Data	Expected Result
adding an item to inventory	**Store_ id = user.own_stores[i] Valid item details**	**Success**
	Store_ id != user.own_stores[i] Valid item details	**Failure – the user is not the store owner**

Use-case 2.4.1.2: Removing an item from inventory

1. **Actor:** user
2. **Precondition:** The user is a store owner, the user is the owner of this specific store, the user is logged in, the item is in the store's inventory
3. **Parameters :** Store id, item id
4. **Actions :**
    - The user enters the item id
    - The system verify that the user is the store owner
    - The system verify that the user is logged in
    - The system verify that the item exist in the inventory
    - The system removes the item from the inventory
    - The user get response according to success or failure

Action	Data	Expected Result
removing an item from inventory	**Item id is exist in the store's inventory & Store_ id = user.own_stores[i]**	**Success**
	Item id isn't exist in the store's inventory	**Failure – the item is not in the store's inventory**

Use-case 2.4.1.3: Updating item details

1. **Actor:** user
2. **Precondition:** The user is a store owner, the user is the owner of this specific store, the user is logged in, the item is in the store's inventory
3. **Parameters :** Store id, item id, item details
4. **Actions :**
    - User enters the item id
    - The system verify that the user is the store owner
    - The system verify that the user is logged in
    - The system verify that the item exist in the inventory
    - The system verify correctness of new details
    - The system update the item's details
    - The user get response according to success or failure

Action	Data	Expected Result
updating item details	**Item id is exist in the store's inventory & Store_ id = user.own_stores[i] & Valid details**	**Success**
	Item id is exist in the store's inventory & Store_ id = user.own_stores[i] & Quantity < 0	**Failure – invalid quantity**

Use-case 2.4.2.1: Set discount policy

1. **Actor:** user
2. **Precondition:** The user is a store manager, the user is a manager of this specific store the user is logged in
3. **Parameters :** Store id, user id, type of users, type of discounts, policy rules
4. **Actions :**
    - The user enter all the parameters for discount policy
    - The system verify that the user is a store manager and has the required permissions
    - The system verify that the user is logged in
    - The system verify correctness of the parameters
    - The system verify that there is no contradiction of consistency rules
    - The system update the store's discount policy
    - The user get response according to success or failure

Action	Data	Expected Result
set discount policy	**Store_ id = user.manager_stores[i] & Valid details**	**Success**
	Store_ id not in user.manager_stores& Valid details	**Failure – the user is not the store's manager**

Use-case 2.4.2.2: Set purchases policy

1. **Actor:** user
2. **Precondition:** the user is a store manager, the user is a manager of this specific store, the user is logged in
3. **Parameters:** Store id, user id, type of users, type of purchases, policy rules
4. **Actions :**
    - The user enters all the parameters for purchases policy
    - The system verify that the user is a store manager and has the required permissions
    - The system verify that the user is logged in
    - The system verify correctness of the parameters
    - The system verify that there is no contradiction of consistency rules
    - The system update the store's purchases policy
    - The user get response according to success or failure

Action	Data	Expected Result
set purchases policy	**Store_ id = user.manager_stores[i] & Valid details**	**Success**
	Store_ id not in user.manager_stores & Valid details	**Failure – the user is not the store's manager**

Use-case 2.4.3.1: Set discount policy constrains

1. **Actor:** user
2. **Precondition:** The user is a store founder, the user is a founder of this specific store, the user is logged in
3. **Parameters:** Store id, user id, discount policy rules constrains
4. **Actions :**
    - The user enters discount policy rules constrains
    - The system verify that the user is a store founder
    - The system verify that the user is logged in
    - The system verify correctness of the constrains
    - The system update the store's discount policy constrains
    - The user gets a response according to success or failure

Action	Data	Expected Result
set discount policy constrains	**Store_ id = user. founder _stores[i] & Valid details**	**Success**
	Store_ id not in user. founder_stores & Valid details	**Failure – the user is not the store's founder**

Use-case 2.4.3.2: Set purchases policy constrains

1. **Actor:** user
2. **Precondition:** The user is a store founder, the user is a founder of this specific store, the user is logged in
3. **Parameters:** Store id, user id, purchases policy rules constrains
4. **Actions:**
    - The user enter purchases policy rules constrains
    - The system verify that the user is the store founder
    - The system verify that the user is logged in
    - The system verify correctness of the constrains
    - The system update the store's purchases policy constrains
    - The user get response according to success or failure

Action	Data	Expected Result
set purchases policy constrains	**Store_ id = user. founder _stores[i] & Valid details**	**Success**
	Store_ id not in user. founder_stores & Valid details	**Failure – the user is not the store's founder**

Use-case 2.4.4: appoint store owner

1. **Actor:** user
2. **Precondition:** The user is a store owner, the user is the owner of this specific store, the user is logged in, the user (who will be appointed) is not an owner of this store
3. **Parameters:** Store id, user id, appointed_user id
4. **Actions:**
    - User enters the appointed_user id
    - The system verifies that the user is the store owner
    - The system verifies that the user is logged in
    - The system verifies that appointed_user is not an owner of this store
    - The system gives appointed_user owner permissions for this store
    - The user gets a response according to success or failure

Action	Data	Expected Result
appoint store owner	**Store_ id != Appointed_user.own_stores[i] & Store_ id = user.own_stores[i]**	**Success**
	Store_ id = appointed_user.own_stores[i] & Store_ id = user.own_stores[i]	**Failure – Appointed_user is already owner of this store**

Use-case 2.4.5: remove appointment of store owner

1. **Actor:** user
2. **Precondition:** The user is a store owner, the user made the appointment of the store owner, the user is the owner of this specific store, the store owner is the owner of this specific store, the user is logged in
3. **Parameters:** Store id, user id, store owner id
4. **Actions :**
    - The user enters the store owner's id
    - The system verifies that the user is logged in
    - The system verifies that the user is the one who made the appointment of the store owner
    - The system verifies that the user is a store owner
    - The system verifies that the store owner is a store owner
    - The system removes the store owner's permissions for this store
    - The system removes all the store owners and managers who were appointed by the store owner
    - The user gets a response according to success or failure

Action	Data	Expected Result
remove appointment of store owner	**Store_ id = user.own_stores[i] = store_owner.own_stores[i] & Store_owner.nominator = user.id**	**Success**
	Store_ id = user.own_stores[i] = store_owner.own_stores[i] & Store_owner.nominator != user.id	**Failure – user is not the nominator of the store owner**

Use-case 2.4.6 : appoint store manager

1. **Actor:** user
2. **Precondition:** The user is a store owner, the user is the owner of this specific store, the user is logged in, the appointed_user is not owner or manager or founder of this store
3. **Parameters:** Store id, user id, appointed_user id
4. **Actions:**
    - User enter the appointed_user id
    - The system verifies that the user is the store owner
    - The system verifies that the user is logged in
    - The system verifies that the appointed_user is not an owner\manager\founder of this store
    - The system gives the appointed_user manager permissions for this store – for getting information
    - The system sets the appointer of the new store manager to the user
    - The user gets a response according to success or failure

Action	Data	Expected Result
appoint store manager	**Store_ id != appointed_user. manager _stores[i] & Store_ id = user. manager _stores[i]**	**Success**
	Store_ id = appointed_user. owner_stores[i] & Store_ id = user. manager _stores[i]	**Failure – the appointed_user is already manager of this store**

Use-case 2.4.7 : set permissions of store manager

1. **Actor :** user
2. **Precondition:** The user is a store owner, the user is the owner of this specific store, the user is logged in, the manager is a manager of this specific store, the manager was appointed by this user
3. **Parameters:** Store id, user id, manager id, permissions
4. **Actions :**
    - User enters the manager id
    - The system verifies that the user is a store owner
    - The system verifies that the user is logged in
    - The system verifies that the manager is a manager of this store
    - The system verifies that the appointer of the manager is the user
    - The system sets the manager permissions for this store
    - The user gets a response according to success or failure

Action	Data	Expected Result
set permissions of store manager	**Store_ id = manager. manager _stores[i] & Store_ id = user. owner_stores[i] & valid permissions & manager. nominator = user_id**	**Success**
	Store_ id = manager. manager _stores[i] & Store_ id != user. owner_stores[i] & valid permissions & manader. nominator = user_id	**Failure – the user is not an owner**

Use-case 2.4.8 : remove appointment of store manager

1. **Actor :** User
2. **Precondition:** The user is a store owner, user is the one who made the appointment of the specific store manager, the user is an owner of this specific store, the user is logged in
3. **Parameters:** Store id, user id, store manager id
4. **Actions :**
    - User enters the store manager id
    - The system verifies that the user is logged in
    - The system verifies that the user is the one who made the appointment of this store manager
    - The system verifies that the user is a store owner
    - The system verifies that the store manager is a store manager
    - The system removes store manager permissions for this store
    - The user gets a response according to success or failure

Action	Data	Expected Result
remove appointment of store manager	**Store_ id = manager. manager _stores[i] & Store_ id = user. owner_stores[i] & valid permissions & manager. nominator = user_id**	**Success**
	Store_ id = manager. manager _stores[i] & Store_ id != user. owner_stores[i] & valid permissions & manager. nominator = user_id	**Failure – the user is not an owner**

Use-case 2.4.9 : close store

1. **Actor:** User
2. **Precondition:** The user is a store founder, the user is a founder of this specific store, the user is logged in, the store is open
3. **Parameters:** Store id, user id
4. **Actions :**
    - User enters the store id
    - The system verifies that the user is logged in
    - The system verifies that the user is the store founder
    - The system verifies that the store is open
    - The system notifies the store owners & managers about the event
    - The system hides from users information about the store and its products
    - The user gets a response according to success or failure

Action	Data	Expected Result
close store	**Store_ id = user. founder _stores[i] & store.status = open**	**Success**
	Store_ id = user. founder _stores[i] & store.status = close	**Failure – the store is already closed**

Use-case 2.4.10 : open closed store

1. **Actor :** User
2. **Precondition :** The user is a store founder, the user is a founder of this specific store, the user is logged in, the store was closed by the user
3. **Parameters :** Store id, user id
4. **Actions :**
    - The user enters the store id
    - The system verifies that the user is logged in
    - The system verifies that user is the store founder
    - The system verifies that the store is closed
    - The system notifies the store owners & managers about the event
    - The user gets a response according to success or failure

Action	Data	Expected Result
open closed store	**Store_ id = user. founder _stores[i] & store.status = close**	**Success**
	Store_ id = user. founder _stores[i] & store.status = open	**Failure – the store is not closed**

Use-case 2.4.11 : get store staff & permissions information

1. **Actor :** User
2. **Precondition :** The user is a store owner, the user is an owner of this specific store, the user is logged in
3. **Parameters :** Store id, user id
4. **Actions :**
    - User enters the store id
    - The system verifies that the user is logged in
    - The system verifies that user is the store owner
    - The user gets a response with information about permissions and managers of this store according to success or failure

Action	Data	Expected Result
get store staff & permissions information	**Store_ id = user.owner _stores[i]**	**Success**
	Store_ id != user. owner _stores[i]	**Failure – the user is not a store owner**

Use-case 2.4.12 : answer users questions

1. **Actor :** User
2. **Precondition :** The user is a store manager, the user is a manager of this specific store, the user is logged in, the user has the required manager permissions
3. **Parameters :** Store id, user id
4. **Actions :**
    - The user enters the store id
    - The system verifies that the user is logged in
    - The system verifies that the user has the required manager permissions
    - The manager gets a response with users questions about this store according to success or failure
    - The user answers the question
    - The system notifies the user about his question response
    - The system changes the status of this question

Action	Data	Expected Result
answer users questions	**Store_ id = user. manager _stores[i] & user.permissions.contains(answering_questions)**	**Success**
	Store_ id != user.manager_stores[i] & !user.permissions.contains(answering_questions)	**Failure – the user does not have the required permissions**

Use-case 2.4.13 : get store purchases history

1. **Actor :** User
2. **Precondition :** The user is a store manager, the user is a manager of this specific store, the user is logged in, the user has the required manager permissions
3. **Parameters :** Store id, user id
4. **Actions :**
    - The user enters the store id
    - The system verifies that the user is logged in
    - The system verifies that the user has the required manager permissions
    - The user gets a response with purchases history of this store according to success or failure

Action	Data	Expected Result
get store purchases history	**Store_ id = user. manager _stores[i] & user.permissions.contains(access_puchases)**	**Success**
	Store_ id != user.manager_stores[i] & !user.permissions.contains(access_puchases)	**Failure – the user does not have the required manager permissions**

Use-case 2.6.1 : close a store permanently

1. **Actor :** User
2. **Precondition :** The user is an admin, the user is logged in, the store exists in the system
3. **Parameters :** Store id, user id
4. **Actions :**
    - The user enters the store id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The system verifies that the store exists
    - The system closes the store
    - The system notifies the store's managers and owners
    - The system cancel all appointments of managers and owners of the store
    - The user gets a response according to success or failure

Action	Data	Expected Result
close a store permanently	**Store_ id exists**	**Success**
	Store_id does not exists	**Failure – the store does not exist**

Use-case 2.6.2 : remove user

1. **Actor :** User
2. **Precondition :** The user is an admin, the admin is logged in, the remove_user exists in the system, the remove_user is not an admin
3. **Parameters :** Remove_user id, user id
4. **Actions :**
    - User enters the remove_user id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The system verifies that the remove_user id exists
    - The system deletes the remove_user from the system
    - The system deletes all the remove_user appointments
    - The system removes all the stores who were created by this remove_user (use-case 2.6.1)
    - The user gets a response according to success or failure

Action	Data	Expected Result
remove user	**Remove_user_ id exists**	**Success**
	Remove_user_ id does not exist	**Failure – the user does not exist**

Use-case 2.6.3 : answer users complain

1. **Actor :** user
2. **Precondition :** The user is an admin, the admin is logged in
3. **Parameters :** User id
4. **Actions :**
    - User enters the user id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The user gets the complains of the users
    - The user answers the complain
    - The system notifies the complaint_user about user's answer
    - The system change complain status
    - The user gets a response according to success or failure

Action	Data	Expected Result
answer users complains	**User_ id = Admins[i].id**	**Success**
	!Admins.contains(User_id)	**Failure – the user is not an admin**

Use-case 2.6.4.1 : get store purchases history

1. **Actor :** User
2. **Precondition :** The user is an admin, the admin is logged in
3. **Parameters :** User id, store id
4. **Actions :**
    - User enters the user id
    - User enters the store id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The user gets a response with purchases history of this specific store according to success or failure

Action	Data	Expected Result
get store purchases history	**User_ id = Admins[i].id & Store_id = Stores[i]**	**Success**
	Admins.contains(User_id) & !Stores.contains(store_id)	**Failure – the store does not exist**

Use-case 2.6.4.2 : get user purchases history

1. **Actor :** User
2. **Precondition :** The user is an admin, the admin is logged in
3. **Parameters :** User id, purchase_user id
4. **Actions :**
    - User enters the user id
    - User enter the purchase_user id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The user gets a response with purchases history of this specific purchase_user according to success or failure

Action	Data	Expected Result
get user purchases history	**User_ id = Admins[i].id & Purcahse_user_id = Users[i].id**	**Success**
	Admins.contains(User_id) & !Users.contains(purchase_user_id)	**Failure – the user does not exist**

Use-case 2.6.5 : get system statistics

1. **Actor :** User
2. **Precondition :** The user is an admin, the admin is logged in
3. **Parameters :** User id

4. **Actions :**
    - User enter the user id
    - The system verifies that the user is logged in
    - The system verifies that the user is an admin
    - The user gets a response with system statistics according to success or failure

Action	Data	Expected Result
get system statistics	**User_ id = Admins[i].id**	**Success**
	!Admins.contains(User_id)	**Failure – the user is not an admin**