Index

# System Section1 -------------------------------------2-14

# User Section 2 ---------------------------------------14-52

## Activities as guest visitor and register---------------------------14-19

## Guest Visitor Purchase Activities---------------------------------19-26

## Purchase Activities of a Subscribed Visitor in the Market----26-33

## Visitor-Member as a store owner:--------------------------------33-50

## Activities as a store manager--------------------------------------50-51

## Activities as a system manager-----------------------------------51-52

# System, section 1:

## Use case 1: Initialize Marketplace System.

Actor: System Administrator or the System (on startup)

Trigger: The administrator starts the system manually, or the system launches automatically upon deployment.

Preconditions:

* A valid configuration file is available (including external service URLs and DB credentials).
* A valid connection to a persistent database exists (via ORM).
* External payment and supply services are reachable.
* The system is not already running.

Parameters:

* configFilePath: Path to the system configuration file
* stateFilePath (optional): Path to a file with scripted initialization commands
* databaseConfig: DB connection settings (host, port, user, password)
* paymentServiceUrl: URL of the external payment service
* supplyServiceUrl: URL of the external supply service
* adminSetup: Optional default admin user credentials

Main Scenario:

1. The system is launched.  
2. It reads and parses the configuration file located at configFilePath.  
3. The system attempts to connect to the database using the credentials provided.  
 - If the connection fails, initialization is stopped.  
4. The system performs a "handshake" request to both the payment and supply services to verify connectivity.  
 - Both must return an "OK" response to proceed.  
5. The system verifies whether at least one System Administrator exists:  
 - Either from the existing database  
 - Or defined in the configuration file  
6. If a state file is provided (stateFilePath), the system:  
 - Reads and executes each command in the file sequentially  
 - If any command fails, the system logs the error and aborts the initialization.  
7. After successful execution:  
 - All system components are ready  
 - The system transitions to "active" mode  
 - Users may now interact with the system (e.g., login, explore, open stores)

Alternative Flows:

* Configuration file is missing or invalid: System logs error and aborts initialization
* Database connection fails: System stops and reports the failure
* Payment or supply service is unreachable: System logs error and refuses to initialize
* No system administrator found or defined: System reports error that admin is required
* State file contains an invalid or failing command: Logs error and aborts the entire initialization sequence
* One or more services return transaction failure (e.g. transactionId = -1): Initialization stops and reports which service failed

Postconditions:

* The system is initialized and ready for normal operation.
* All services are validated and reachable.
* Optional startup data (stores, users, etc.) is loaded and persisted.
* If any required component fails, the system does not start.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_InitializeSystem\_with\_config\_and\_state\_file | Valid config file + valid state file with startup commands | System connects to DB, verifies services, executes commands, and becomes active |
| Failure\_InitializeSystem\_missing\_config\_file | No config file found at given path | System logs error and aborts |
| Failure\_InitializeSystem\_unavailable\_payment\_service | Payment service URL in config is unreachable | System reports error and stops |
| Failure\_InitializeSystem\_with\_invalid\_state\_command | State file contains a malformed command | System logs error and cancels the whole initialization |
| Success\_InitializeSystem\_without\_state\_file | Valid config, but no state file is provided | System initializes with clean state, ready for use |
| Failure\_InitializeSystem\_without\_admin\_user | No admin defined in DB or config | System blocks startup and logs admin requirement |

## Use case 2: Payment Processing.

Actor: User

Trigger: User requests to process payment for a purchase.

Preconditions:

* - At least one recognized and available payment service is configured.  
  - User is registered and authenticated (logged in).  
  - User has at least one item selected for purchase.

Parameters:

* - token: session identifier  
  - userId: ID of the logged-in user  
  - shoppingCart: contents of the user’s cart  
  - paymentDetails: includes the following fields required by the external payment system:  
   • card\_number  
   • month  
   • year  
   • holder (cardholder name)  
   • cvv  
   • id (national ID or equivalent)

Main Scenario:

1. The user initiates a payment for the selected items.  
2. The system gathers the required payment and user details.  
3. The system verifies that a valid payment service is configured and available.  
4. The system constructs an HTTP POST request with `action\_type = pay` and the required fields.  
5. The request is sent to the external payment service.  
6. If the payment service returns a valid transaction ID (between 10000 and 100000):  
 • The system marks the payment as successful.  
 • The user proceeds to the next step in the purchase process (e.g., shipping).  
7. The user is notified that the payment was completed successfully.

Alternative Flows:

* - Transaction details are missing or invalid:  
   » System shows a validation error and cancels the request.  
    
  - No payment service configured:  
   » System shows a configuration error.  
   » Payment cannot be processed.  
    
  - Payment service is unavailable:  
   » System shows an error message.  
   » Payment is canceled and user is notified.  
    
  - Payment rejected by the service (transactionId = -1):  
   » System cancels the transaction and notifies the user.  
   » No changes are made to the cart or purchase history.

Postconditions:

* - On success: the system records the transaction and continues to supply processing.  
  - On failure: no items are marked as paid, and the cart remains unchanged.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_ProcessPayment | 1. User is logged in 2. Items selected 3. Valid payment service available 4. Valid payment details provided | 1. Payment is processed successfully 2. Transaction is recorded 3. User proceeds to shipping |
| Failure\_ProcessPayment\_with\_service\_unavailable | 1. Valid user and cart 2. Payment service is down | 1. System shows service error 2. Payment is not processed |
| Failure\_ProcessPayment\_with\_invalid\_data | 1. Valid user 2. Missing or invalid payment details | 1. System shows validation error 2. Payment is rejected |
| Failure\_ProcessPayment\_with\_no\_service\_configured | 1. Valid user 2. No payment service configured in system | 1. Configuration error shown 2. No transaction made |
| Failure\_ProcessPayment\_with\_transaction\_rejection | 1. Valid request sent 2. Payment service returns transactionId = -1 | 1. System cancels transaction 2. User notified of failure 3. Cart remains unchanged |

## Use case 3 :supply processing

Actor: User

Trigger: User completes a purchase, and a paid order is confirmed and ready for supply.

Preconditions:

* - At least one recognized and available supply service is configured.  
  - User is registered and authenticated.  
  - A confirmed payment transaction exists.

Parameters:

* - token: session identifier  
  - userId: ID of the user who made the purchase  
  - shipmentDetails: includes the following fields required by the external supply system:  
   • name  
   • address  
   • city  
   • country  
   • zip

Main Scenario:

1. The system detects that a purchase has been completed and is ready for delivery.  
2. The system gathers the necessary shipment details from the user profile and purchase.  
3. The system verifies that a supply service is configured and reachable.  
4. The system constructs an HTTP POST request to the external supply service with `action\_type = supply` and all required fields.  
5. The request is sent to the supply service endpoint.  
6. If the service returns a valid transaction ID (between 10000 and 100000):  
 • The system marks the delivery as successful.  
 • Updates the purchase record with the shipping ID.  
 • Notifies the user that their order has been shipped.  
7. The order status is updated to 'Shipped' in the system.

Alternative Flows:

* - Supply service is unavailable:  
   » The system shows an error message and does not initiate supply.  
    
  - Supply service rejects the request (transaction ID = -1):  
   » The system logs the failure and notifies the user.  
    
  - Invalid shipment details:  
   » The system cancels the supply request and logs the error.  
    
  - No supply service configured:  
   » The system shows a configuration error.  
   » Supply cannot be processed.

Postconditions:

* - On success: the system logs the shipment transaction and notifies the user.  
  - On failure: the system does not proceed with delivery, and the order remains pending shipment.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_ProcessSupply | 1. Valid supply service available 2. Valid shipment details provided | 1. Request sent and confirmation received 2. Order marked as shipped |
| Failure\_ProcessSupply\_with\_service\_unavailable | 1. Supply service is offline 2. Valid delivery info | 1. System shows error 2. Supply not processed |
| Failure\_ProcessSupply\_with\_service\_rejection | 1. Valid request sent 2. Service returns transactionId = -1 | 1. System logs failure 2. User notified 3. Order not marked as shipped |
| Failure\_ProcessSupply\_with\_invalid\_shipping\_details | 1. Shipment info is missing or invalid | 1. System cancels request 2. Logs error |
| Failure\_ProcessSupply\_with\_no\_supply\_service | 1. No supply service configured 2. Order is ready | 1. Configuration error shown 2. Supply not executed |

# Use case 4: Cancel Payment Transaction

Actor: System or User

Trigger:

* - A confirmed payment transaction must be canceled due to rollback, failure, or user request.

Preconditions:

* - A valid payment transaction ID exists.  
  - Payment service is configured and reachable.

Parameters:

* - transactionId: ID of the payment transaction to cancel  
  - action\_type: 'cancel\_pay'  
  - token: session identifier (if initiated by user)

Main Scenario:

1. The system identifies a payment that needs to be canceled.  
2. It prepares a cancellation request with `transactionId`.  
3. The system sends an HTTP POST request to the external payment service with `action\_type = cancel\_pay`.  
4. If the service confirms cancellation:  
 • The transaction is marked as canceled.  
 • Logs are updated.  
 • The user is notified (if relevant).

Alternative Flows:

* - Invalid transaction ID:  
   » System logs error and notifies admin or user.  
    
  - Payment service is unreachable:  
   » System logs failure and retries or queues for later.  
    
  - Cancellation is rejected (e.g., returns -1):  
   » Transaction not canceled, flagged for manual review.

Postconditions:

* - On success: transaction is canceled and excluded from further processing.  
  - On failure: system logs error and maintains rollback awareness.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_CancelPaymentTransaction | 1. Valid transactionId 2. Payment service reachable | 1. Cancellation confirmed 2. Logs updated |
| Failure\_CancelPayment\_InvalidTransactionId | 1. transactionId not in system | 1. Error logged 2. No action taken |
| Failure\_CancelPayment\_ServiceDown | 1. Valid transactionId 2. Payment service unreachable | 1. Cancellation fails 2. System retries or queues request |
| Failure\_CancelPayment\_RejectedByService | 1. Valid cancellation request 2. Service returns -1 | 1. Cancellation rejected 2. System flags for manual review |

# Use case 5: Cancel Supply Transaction

Actor: System or User

Trigger:

* - A confirmed shipment transaction must be canceled due to rollback, failure, or user request.

Preconditions:

* - A valid supply transaction ID exists.  
  - Supply service is configured and reachable.

Parameters:

* - transactionId: ID of the supply transaction to cancel  
  - action\_type: 'cancel\_supply'  
  - token: session identifier (if initiated by user)

Main Scenario:

1. The system identifies a supply transaction that must be canceled.  
2. It prepares a cancellation request using the `transactionId`.  
3. The system sends an HTTP POST request to the external supply service with `action\_type = cancel\_supply`.  
4. If the service confirms cancellation:  
 • The system marks the shipment as canceled.  
 • Updates the logs.  
 • Notifies the user if applicable.

Alternative Flows:

* - Invalid transaction ID:  
   » System shows an error and logs failure.  
    
  - Supply service is unavailable:  
   » System logs the failure and retries or alerts admin.  
    
  - Cancellation rejected (response = -1):  
   » System flags the rollback as partial failure.

Postconditions:

* - On success: the shipment is marked as canceled, and the order is updated accordingly.  
  - On failure: error is logged; system may retry or escalate.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_CancelSupplyTransaction | 1. Valid transactionId 2. Supply service reachable | 1. Cancellation confirmed 2. Logs updated |
| Failure\_CancelSupply\_InvalidTransactionId | 1. transactionId not found | 1. Error logged 2. No cancellation performed |
| Failure\_CancelSupply\_ServiceDown | 1. Valid transactionId 2. Supply service unreachable | 1. Retry or error logged 2. Cancellation fails |
| Failure\_CancelSupply\_RejectedByService | 1. Valid cancellation request 2. Service returns -1 | 1. Cancellation rejected 2. Flagged for manual intervention |

# Use case 6: System Recovery After Crash / Restart

Actor: System

Trigger:

* - The system restarts after a crash, power failure, or manual reboot.

Preconditions:

* - A persistent database exists with previous system state.  
  - Configuration file is accessible for startup.  
  - External services (payment/supply) are operational.

Parameters:

* - configFilePath: path to the system configuration file  
  - databaseConfig: DB connection parameters (host, port, user, pass)  
  - recoveryLog: internal log of persisted events (optional)  
  - ORM persistence layer: all domain entities stored in DB

Main Scenario:

1. The system is started following a crash or shutdown.  
2. It loads the configuration file from `configFilePath`.  
3. It connects to the database using the stored credentials.  
4. It loads all persisted entities: users, stores, items, bids, and purchase history.  
5. The system rebuilds in-memory data structures from the ORM layer.  
6. The system resumes operations without needing a state file replay.  
7. Services (e.g., schedulers, notifiers) are restarted.  
8. The system becomes fully operational.

Alternative Flows:

* - Database is unreachable:  
   » System logs critical error and refuses to boot.  
    
  - Data is partially corrupted or unreadable:  
   » System logs recovery failure and attempts best-effort restore.  
    
  - Configuration file missing:  
   » System cannot connect and reports initialization error.

Postconditions:

* - The system resumes operation with the latest consistent state from the database.  
  - All restored entities are available and functional.  
  - No manual intervention is needed unless recovery fails.

Acceptance tests:

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Success\_RecoveryAfterCrash | 1. Database contains full persisted state 2. System restarts normally | 1. System loads all data and resumes operation |
| Failure\_Recovery\_DBUnavailable | 1. DB server is offline at boot time | 1. System logs error and blocks startup |
| Failure\_Recovery\_CorruptData | 1. DB is reachable 2. Some entities are corrupted | 1. System attempts recovery 2. Logs issues for admin review |
| Success\_Recovery\_SkipStateFileReplay | 1. Database is available 2. State file is not used after crash | 1. System uses ORM data 2. Initialization completes without errors |
| Failure\_Recovery\_MissingConfigFile | 1. Configuration file is missing or unreadable | 1. System fails startup and logs configuration error |

## Use case 7: Real-Time Notifications

* 1. **Actor:** System
  2. **Trigger:** A relevant event occurs that requires notifying the user in real-time.
  3. **Precondition:**
* The market system is operational
* Users are registered and logged into the system

**4.Parameters:** senderId, receiverId(s), message

5.**Main Scenario:**

* System detects an event that requires a notification.
* System generates a notification message.
* System sends the notification (or stores it if user is offline).

**6. Alternative flow:**

* **User tries to purchase from a closed store  
  » The system generates a real-time notification that informs the user the action cannot be completed.**
* **User is offline  
  » The system stores the notification as a delayed message.**
* **Notification delivery fails  
  » The system retries. If it fails again, the message is stored for later.**
* **User is not registered (invalid ID)  
  » The system does not send the notification and logs the failure.**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_RealTimeNotification\_with\_user\_online | 1. Registered user is online 2. A relevant event occurs (e.g. message received) | 1. Notification is delivered to the user in real time |
| Failure\_RealTimeNotification\_with\_user\_offline | 1. Registered user is offline 2. A relevant event occurs | 1. Notification is stored for later delivery |
| Failure\_RealTimeNotification\_with\_delivery\_failure | 1. User is online 2. Notification fails due to temporary issue | 1. System retries sending 2. If it fails again, notification is stored and logged |
| Failure\_RealTimeNotification\_with\_invalid\_user\_id | 1. Event occurs 2. Recipient is not a registered user | 1. Notification is not sent 2. System logs the issue |

## Use case 8: Delayed Notifications

* + 1. **Actor** :System
    2. **Trigger:**A relevant event occurs that requires notifying the user who is currently offline
    3. **Precondition:**
* The recipient is a registered subscriber.
* A notification event has occurred.
* The user is currently offline in the market.
  + 1. **Parameters:** userId
    2. **Main Scenario:**
* System detects an event that requires a notification.
* System generates a notification message based on the event type.
* System saves the notification.
* When the User login the system will send the notification

1. **Alternative flow**:

* **Notification Data is Invalid**

» Notification is not stored.

» System logs the issue for review.

* **User Never Logs Back In**

» Notification remains stored in the system until expiration or manual removal.

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_StoreDelayedNotification\_with\_user\_offline | 1. Registered user is offline 2. A notification is triggered | 1. Notification is stored for future delivery |
| Success\_DeliverStoredNotification\_with\_user\_logged\_in | 1. User logs in after being offline 2. Stored notification exists | 1. Notification is delivered and displayed to the user |
| Failure\_StoreDelayedNotification\_with\_invalid\_message\_data | 1. Notification is missing user ID or message | 1. Notification is not stored 2. System logs the issue |
| Failure\_StoreDelayedNotification\_with\_user\_never\_returning | 1. Notification is saved 2. User does not return | 1. Notification remains stored until expiration or manual removal |

# Users, section 2:

## 1.Activities as guest visitor and register:

### Use case 1 - Enter as Guest

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Enter as Guest |
| Actor | User |
| Trigger | Visiting the market and explore all the items and stores in it. |
| Precondition | 1 The system is properly initialized. |
| Parameters | There is no need for params. |
| Main Scenario | 1. The system boots up.  2.The user request to explore all markets.  3.the user receive number of markets and can load more until get all markets.  4. a temporary shopping cart created for the visitor and session token . |
| Alternative flow | 1. **There are no markets in the system** » The system notifies the user that there are no markets available 2. **Internal server error occurs** » The system notifies the user to try again later |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_EnterAsGuest | 1. System is initialized 2. Markets are available | 1. Guest user enters successfully 2. Can view all markets and products |
| Failure\_EnterAsGuest\_with\_no\_markets\_available | 1. System is initialized 2. No markets are created | 1. System shows a message: no markets available |
| Failure\_EnterAsGuest\_with\_internal\_error | 1. System is initialized 2. Server error occurs during loading | 1. System shows error and suggests to try again later |

### Use case 2 - Exit as Guest

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Exit as Guest |
| Actor | User/Guest |
| Trigger | Exiting the market while not registered |
| Precondition | The market system is running and was explore on it . |
| Parameters | token |
| Main Scenario | * The system is running and available for users * The guest user entered the system and explored it * The user exits from the system * The shopping cart associated with this guest is deleted |
| Alternative flow | * **No markets exist** » The system notifies the user that there are no markets to explore * **Session token is invalid or expired** » The system cancels the operation and shows a general error |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_ExitAsGuest | 1. User entered as guest 2. Explored system 3. Initiates exit | 1. Guest exits successfully 2. Shopping cart is deleted |
| Failure\_ExitAsGuest\_with\_no\_markets | 1. System running 2. No markets exist in the system | 1. System shows a message: no markets available |
| Failure\_ExitAsGuest\_with\_invalid\_token | 1. Guest session token is missing or expired | 1. System cancels operation 2. Shows an error message to the user |

### Use case 3 –Register

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Register for a new user. |
| Actor | User/ Guest |
| Trigger | New user register to the system. |
| Precondition | The system is properly initialized (according to his usecase) |
| Parameters | token, username, password,age |
| Main Scenario | 1.the system is running and available.  2. The user guest open the system site.  3. user try to register and entered a valid info.  4. user registered successfully .  5. return to home page as a register. |
| Alternative flow | 1. **Username already exists** » The system rejects the registration and informs the user  2. **Registration details are invalid** » The system rejects the registration and shows a validation error |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_Register | 1. Guest enters valid registration info (username, password, age) | 1. User is registered successfully 2. Redirected to home page as registered user |
| Failure\_Register\_with\_existing\_username | 1. Username already exists in the system | 1. System rejects the registration 2. Shows message about duplicate username |
| Failure\_Register\_with\_invalid\_parameters | 1. Registration info includes invalid or missing fields | 1. System shows validation error 2. Registration is not completed |

### Use case 4 – Login

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Logging in to the system. |
| Actor | User guest |
| Trigger | User attempt to login. |
| Precondition | 1. The system is properly initialized  2. user already registered to the system.  3.User is logout. |
| Parameters | token, username, password |
| Main Scenario | 1.the user try to login using his username and password  2.the user logged in successfully.  3.created session token for this user . |
| Alternative flow | * **Username not found** » The system shows a general error indicating that the user does not exist * **Incorrect password** » The system shows a general error indicating login failure * **User is already logged in on another session** *(if applicable)* » The system allows login and refreshes the session (optional based on policy) |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_Login | 1. User is registered 2. Correct username and password are provided | 1. User is logged in successfully 2. Session token is created |
| Failure\_Login\_with\_nonexistent\_username | 1. User enters a username that does not exist in the system | 1. System shows error 2. Login is rejected |
| Failure\_Login\_with\_incorrect\_password | 1. User enters correct username but incorrect password | 1. System shows error 2. Login is rejected |
| Failure\_Login\_with\_invalid\_input\_format (optional) | 1. User submits form with empty or malformed fields | 1. System shows validation error 2. Login is not processed |

## 2.Guest Visitor Purchase Activities:

### Use case 1 - Explore markets

|  |  |
| --- | --- |
| Name | Explore markets |
| Actor | User |
| Trigger | User want to explore markets in the systems. |
| Precondition | 2. guest is in the system. |
| Parameters | token |
| Main Scenario | 1. The guest-visitor inputs the store id  2. The system shows the guest-visitor information about the requested store and its products.(\*\*) |
| Alternative flow | **No stores are active or available** » The system shows a message that no markets are currently available  **System fails to load markets (e.g., temporary error)** » The system shows a general error and allows the user to retry |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_ExploreMarkets\_with\_active\_stores | 1. System initialized 2. At least one store is active | 1. Store list is displayed 2. User can browse markets |
| Failure\_ExploreMarkets\_with\_no\_stores\_available | 1. System initialized 2. No active or visible stores in the system | 1. System shows message: no markets available |
| Failure\_ExploreMarkets\_with\_loading\_error | 1. System initialized 2. Temporary issue occurs when fetching market data | 1. System shows error 2. User may retry or refresh |

### Use case 2 - Search Filters

|  |  |
| --- | --- |
| Name | Search Filters. |
| Actor | User |
| Trigger | User requested to search a market with filter. |
| Precondition | 1.guest is in the system.  2.A valid supply service is available and connected and the store contain products. |
| Parameters | token, searchCriteria |
| Main Scenario | 1.User guest request to search market with filters.  3.User get a list of all markets that fit the filters .  4.User select a filters for items on the market.  5.User get a list of items that fit the filters. |
| Alternative flow | **No products match the filters** » The system shows a message indicating that no results were found  **Invalid filter combination** » The system shows an error or ignores conflicting filters  **Empty search with no filters** » The system returns all available products |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_Search\_with\_valid\_filters | 1. Products exist 2. User applies valid filters (e.g., category: Electronics, price < 1000) | 1. Filtered product list is displayed |
| Success\_Search\_with\_partial\_filters | 1. User applies some filters only (e.g., name or store) | 1. Filtered results returned accordingly |
| Failure\_Search\_with\_no\_matching\_results | 1. Valid filters applied 2. No product meets all criteria | 1. System shows message: no results found |
| Failure\_Search\_with\_conflicting\_filters | 1. User applies conflicting filters (e.g., minPrice > maxPrice) | 1. System handles gracefully (shows error or resets filters) |
| Success\_Search\_with\_empty\_filters | 1. User submits empty search and no filters | 1. All available products are displayed |

### Use case 3 – add to Cart Case

|  |  |
| --- | --- |
| Actor | User Geust |
| Trigger | User requested to add an item to the cart. |
| Precondition | 1.guest is in the system.  3.A valid supply service is available and connected.(\*\*) |
| Parameters | token, itemToAdd |
| Main Scenario | 1.The user guest requests to explore a market in the system.  2.the user requests to add an item in the market to cart.  3.the item added to cart of user guest.  4.The user open the cart and find the item that he added . |
| Alternative flow | * **Product is not found** » The system shows an error and does not add the product * **Requested quantity exceeds available stock** » The system limits the quantity to available stock or rejects the request * **Negative or zero quantity entered** » The system rejects the request and shows an error |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddProductToCart\_with\_valid\_data | 1. User is logged in or guest 2. Product exists 3. Quantity is available | 1. Product is added to cart 2. Cart is updated |
| Failure\_AddProductToCart\_with\_nonexistent\_product | 1. Product ID is invalid | 1. System shows error 2. Product is not added |
| Failure\_AddProductToCart\_with\_insufficient\_stock | 1. Product exists 2. Requested quantity exceeds stock | 1. System shows error or limits quantity 2. Item not added or partially added |
| Failure\_AddProductToCart\_with\_invalid\_quantity | 1. Product exists 2. Quantity is zero or negative | 1. System shows error 2. Product is not added |

### Use case 4 – Cart Edit

|  |  |
| --- | --- |
| Name | Explore markets |
| Actor | User Guest |
| Trigger | User requested to edit items on the cart. |
| Precondition | 1.user is in the system. |
| Parameters | token, itemToAdd, quantity |
| Main Scenario | 1.The user guest requests to explore the cart.  2.the user requests to add the quantity of one item.  3.the item quantity on the cart has been increased as user request. |
| Alternative flow | * **Product is out of stock when trying to add/update** » The system shows an error and prevents the operation * **Cart session expired or does not exist** » The system starts a new cart or shows an error * **Invalid quantity (negative or zero)** » The system rejects the change and shows an error   . |

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddProduct\_to\_cart | 1. User is logged in or guest 2. Product exists and is in stock | 1. Product added to cart 2. Cart is updated |
| Success\_RemoveProduct\_from\_cart | 1. Product exists in cart 2. User chooses to remove it | 1. Product removed from cart 2. Cart is updated |
| Success\_UpdateProductQuantity\_in\_cart | 1. Product is in cart 2. User changes quantity 3. New quantity is valid | 1. Quantity is updated 2. Updated total is shown |
| Failure\_EditCart\_with\_product\_out\_of\_stock | 1. User tries to add or update a product that is not in stock | 1. System rejects operation 2. User is notified |
| Failure\_EditCart\_with\_invalid\_quantity | 1. User tries to update item to negative or zero quantity | 1. System rejects request 2. Error is shown |
| Failure\_EditCart\_with\_missing\_cart\_session | 1. User’s session expired or cart not initialized | 1. System initializes new cart or shows an error |

### Use case 5 – Buy the cart

|  |  |
| --- | --- |
| Name | Explore markets |
| Actor | User Guest |
| Trigger | User requested to buy the items on the cart. |
| Precondition | 1.User is in the system.  2.a cart with at least one item. |
| Parameters | token |
| Main Scenario | 1.Check policies for item and the user  2. availability of each product in cart  3. stock updates delete them from the stock of the store .  4. create receipt for the cart to the user .  5.delete the cart item and make it empty.  6.the system moves the user to payment process (service) with the cost of the cart.  7.the system create connection with the supply service.  8.User bought all the items on the cart.  9. the system notifies the user of the successful purchase. |
| Alternative flow | 1. **Cart is empty** » The system shows a message and prevents the transaction 2. **Payment is rejected** » The system cancels the transaction and informs the user 3. **Product in cart is out of stock** » The system shows a message and does not complete the purchase 4. **Store purchase policy blocks the transaction** » The system rejects the cart and informs the user |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_BuyCart\_as\_registered\_user | 1. Cart has items 2. User is logged in 3. Valid payment and delivery info | 1. Payment is processed 2. Delivery is arranged 3. Receipt is generated |
| Success\_BuyCart\_as\_guest\_user | 1. Cart has items 2. Guest session 3. Valid payment and delivery info | 1. Same as above — process completes successfully |
| Failure\_BuyCart\_with\_empty\_cart | 1. Cart is empty 2. User initiates purchase | 1. System shows error 2. No purchase is made |
| Failure\_BuyCart\_with\_payment\_failure | 1. Cart is valid 2. Payment service rejects the request | 1. System cancels transaction 2. User is notified |
| Failure\_BuyCart\_with\_product\_out\_of\_stock | 1. Cart includes item that is no longer in stock | 1. System shows error 2. Purchase not completed |
| Failure\_BuyCart\_with\_policy\_restriction | 1. Cart violates store purchase policy (e.g., minimum age not met, amount restriction) | 1. System blocks transaction 2. User is notified |

## 3.Purchase Activities of a Subscribed Visitor in the Market:

### Use Case 1: Logout

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user requests to log out.
3. **Preconditions:**

* The user is logged into the system.
* The user is a registered member.

1. **Parameters:** SessionToken
2. **Main Scenario:**

* System verifies the session token.
* System marks the user as logged out.
* System stores the user's shopping cart for future visits.
* System redirects the user to the guest mode.

1. **Alternative Flows:**  
    **Session token is invalid**  
   » The system denies the logout request and shows an error

 **User is logged in on multiple devices**  
» Only the current session is logged out  
» Other sessions remain active  
» User is redirected to guest mode only on the current session

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_Logout\_with\_valid\_session | 1. User is logged in with a valid session token | 1. Session is invalidated 2. Cart is saved 3. User is redirected to guest mode |
| Failure\_Logout\_with\_invalid\_session\_token | 1. Session token is invalid or expired | 1. System denies logout 2. Error message shown |
| Success\_Logout\_with\_multiple\_active\_sessions | 1. User is logged in on multiple devices 2. Logout requested from one device | 1. Session on that device ends 2. Other sessions remain active |

### Use Case 2: Open Store

* 1. **Actor:** Subscribed Visitor
  2. **Trigger:** The user initiates a request to open a new store.
  3. **Preconditions:**
* The user is logged in.
* The user does not currently own the store requested.
  1. **Parameters:** token, storeName, category
  2. **Main Scenario:**
* System validates that the user can open a store.
* System registers the store with the given details.
* System assigns the user as the store’s founder.
* System confirms the store opening to the user.

1. **Alternative Flow :**  **\* Store name already exists  
   » The system rejects the request and informs the user that the store name is already taken  
   \* User is not logged in  
   » The system denies the request and prompts the user to log in**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_OpenStore | 1. User is logged in 2. Store name is unique 3. Valid category is provided | 1. Store is registered 2. User is assigned as founder |
| Failure\_OpenStore\_with\_existing\_store\_name | 1. User is logged in 2. Store name already exists | 1. System rejects request 2. Error shown to user |
| Failure\_OpenStore\_with\_user\_not\_logged\_in | 1. User is not authenticated 2. Store creation request is sent | 1. System denies request 2. Prompts user to log in |

### Use Case 4: Rate Product/Store

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user requests to submit a rating.
3. **Preconditions:**

* The user has purchased the product/store being rated.
* The user is logged in.

1. **Parameters:** token, storeId, productId (for product), newRank
2. **Main Scenario:**

* System verifies that the user has purchased the product/store.
* System records the rating and review.
* System updates the average rating of the product/store.
* System confirms the submission.

1. **Alternative flow**:

* **User has not purchased the product/store**  
  » The system rejects the rating request and informs the user
* **User submits multiple ratings** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  » The new rating replaces the previous one  
  » The average rank is recalculated

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_RateProduct\_with\_valid\_purchase | 1. User is logged in 2. User has purchased the product 3. Valid rating submitted | 1. Rating is stored 2. Product rank is updated |
| Success\_UpdateRating\_with\_previous\_rating\_existing | 1. User previously rated product 2. New rating is submitted | 1. Previous rating is replaced 2. Average rank is recalculated |
| Failure\_RateProduct\_with\_no\_purchase\_history | 1. User is logged in 2. User never purchased the product/store | 1. System rejects the request 2. Rating is not stored |

### Use Case 5: Send Messages to Store

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user sends a message to a store.
3. **Preconditions:**

* The user is logged in.
* The store exists in the system.

1. **Parameters:** Parameters: senderId, receiverId, message
2. **Main Scenario:**

* System verifies the store's existence.
* The user write the message in input box.
* System sends the message to the store's message inbox.
* System notifies the store owner.
* System confirms the message submission to the user.

1. **Alternative flow:**

* **Store does not exist  
  » The system informs the user that the store was not found**
* **Message content is empty  
  » The system rejects the message and shows an error**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_SendMessage\_to\_store | 1. User is logged in 2. Store exists 3. Message is written | 1. Message is stored 2. Store owner is notified |
| Failure\_SendMessage\_with\_nonexistent\_store | 1. User is logged in 2. Store ID does not exist | 1. System shows error 2. Message is not sent |
| Failure\_SendMessage\_with\_empty\_content | 1. User is logged in 2. Store exists 3. Message field is empty | 1. System rejects message 2. Error is shown 3. Store owner not notified |

### Use Case 7: View Purchase History

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user requests their purchase history.
3. **Preconditions:**

* The user is registered and logged in.
* The user has made past purchases.

1. **Parameters:** token
2. **Main Scenario:**

* System retrieves the user's past purchases.
* System presents the list of purchases.
* System ensures that removed products and stores remain in history.

1. **Alternative flow:**

* **User has no purchase history  
  » The system shows a message indicating there are no recorded purchases**
* **Purchased product or store was removed  
  » The system includes the entry and indicates the product/store is no longer available**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_ViewPurchaseHistory\_with\_existing\_purchases | 1. User is logged in 2. User has completed purchases | 1. System displays list of past purchases |
| Failure\_ViewPurchaseHistory\_with\_no\_purchase\_data | 1. User is logged in 2. User has not made any purchases | 1. System shows message: no purchase history found |
| Success\_ViewPurchaseHistory\_with\_removed\_product\_or\_store | 1. User is logged in 2. User has purchased an item that was later removed | 1. System shows the purchase with a note: item no longer available |

### Use Case 9: Submit Purchase Bid

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user submits a bid for a product.
3. **Preconditions:**

* The user is logged in.
* The product is available for bidding.

1. **Parameters:** token, bid
2. **Main Scenario:**

* System verifies that the product is available for bidding.
* System checks that the bid amount is valid (positive and greater than 0).
* System records the user's bid.
* System notifies the store owner in real-time about the new bid.
* System confirms the bid submission to the user according to the store owner comments(yes /no/ …).

1. **Alternative flow:**

* **Product is not available for bidding**  
  » The system rejects the bid and informs the user
* **Bid amount is invalid**  
  » The system rejects the request and shows a general error
* **Store is closed**  
  » The system rejects the bid and informs the user that bidding is disabled

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_SubmitBid\_with\_valid\_data | 1. User is logged in 2. Product allows bidding 3. Valid bid amount | 1. Bid is submitted 2. Store owner is notified |
| Failure\_SubmitBid\_with\_bidding\_closed | 1. Product is not open for bids 2. User is logged in | 1. System rejects bid 2. User is informed |
| Failure\_SubmitBid\_with\_invalid\_amount | 1. Product allows bidding 2. Bid amount is negative or zero | 1. System rejects bid 2. Error is shown to user |
| Failure\_SubmitBid\_with\_store\_closed | 1. Product is in a closed store 2. User is logged in | 1. System blocks bid 2. User is notified that bidding is disabled |

### Use Case 10: Buy Product in Auction

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user wins an auction and proceeds with the purchase.
3. **Preconditions:**

* The auction has ended.
* The user is the highest bidder.

1. **Parameters:** token,AuctionID, PaymentDetails
2. **Main Scenario:**

* System verifies the auction outcome.
* System processes the payment.
* The user try to buy the product using confirm price.
* System confirms the purchase to highest bidder.

1. **Alternative flow:**

* **User is not the winning bidder  
  » The system denies the purchase and informs the user they did not win**
* **Payment fails  
  » The system cancels the transaction and informs the user of the failure**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_BuyAuctionProduct\_as\_winner | 1. Auction has ended 2. User is highest bidder 3. Valid payment details | 1. Payment is processed 2. Product added to cart 3. User is notified |
| Failure\_BuyAuctionProduct\_with\_non\_winner\_user | 1. Auction has ended 2. User is not the winning bidder | 1. System denies purchase 2. User is informed they are not the winner |
| Failure\_BuyAuctionProduct\_with\_payment\_error | 1. User won the auction 2. Payment details are invalid | 1. Payment fails 2. Purchase is not completed 3. User is notified |

### Use Case 11: Buy Product in Lottery

1. **Actor:** Subscribed Visitor
2. **Trigger:** The user wins a lottery and proceeds with the purchase.
3. **Preconditions:**

* The lottery has ended.
* The user was randomly selected as the winner.

1. **Parameters:** LotteryID, PaymentDetails
2. **Main Scenario:**

* System verifies the lottery outcome.
* System processes the payment.
* System confirms the purchase.

1. **Alternative flow:**

* **User is not the selected winner  
  » The system denies the purchase and informs the user**
* **Payment fails  
  » The system cancels the transaction and informs the user**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_BuyLotteryProduct\_as\_winner | 1. Lottery has ended 2. User is the selected winner 3. Valid payment details | 1. Payment is processed 2. Product added to cart 3. User is notified |
| Failure\_BuyLotteryProduct\_with\_non\_winner\_user | 1. Lottery has ended 2. User is not the selected winner | 1. System denies purchase 2. User is informed they are not the winner |
| Failure\_BuyLotteryProduct\_with\_payment\_error | 1. User won the lottery 2. Payment details are invalid | 1. Payment fails 2. Purchase is not completed 3. User is notified |

## 4.Visitor-Member as a store owner:

### Use-case: 1.a Stock management- AddProduct.

1. **Actor:** store owner.
2. **Trigger:** add a product to a specific store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Product should be given.

1. **Parameters:** storeId, token, productId, quantity, price, category
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System adds the product successfully.

1. **Alternative flow:**

* **Store not found  
  » The system notifies the user that the store does not exist**
* **User is not the owner of the store  
  » The system rejects the request and denies permission**
* **No product data provided  
  » The system shows an error and does not proceed**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddProduct\_to\_store | 1. Store exists 2. User is logged in and is the owner 3. Product data is valid | 1. Product is added to store 2. Product appears in the inventory |
| Failure\_AddProduct\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in as store owner | 1. System shows error 2. Product is not added |
| Failure\_AddProduct\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the store owner | 1. System denies request 2. Product is not added |
| Failure\_AddProduct\_with\_missing\_product\_data | 1. Store exists 2. User is the store owner 3. Product details are missing | 1. System shows error 2. Product is not added |

### Use-case: 1.b Stock management- DeleteProduct.

1. **Actor:** store owner.
2. **Trigger:** delete a product from a specific store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Product exists.

1. **Parameters:** storeId, token, productId
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System validates that the product exists.
* System deletes the product successfully.

1. **Alternative flow:**

* **Store not found  
  » The system notifies the user that the store does not exist**
* **User is not the owner of the store  
  » The system denies the request due to lack of permissions**
* **Product does not exist in the store  
  » The system shows an error and does not perform deletion**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_RemoveProduct\_from\_store | 1. Store exists 2. User is logged in and is the store owner 3. Product exists | 1. Product is deleted from store 2. Product is no longer visible |
| Failure\_RemoveProduct\_with\_nonexistent\_store | 1. Store does not exist 2. User is logged in and attempts deletion | 1. System shows error 2. Product is not deleted |
| Failure\_RemoveProduct\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but is not the store owner | 1. System denies request 2. Product is not deleted |
| Failure\_RemoveProduct\_with\_nonexistent\_product | 1. Store exists 2. User is the owner 3. Product ID is invalid or missing | 1. System shows error 2. Product is not deleted |

### Use-case: 1.c Stock management- ChangeProduct.

1. **Actor:** store owner.
2. **Trigger:** change a product in a specific store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Product exsits.

1. **Parameters:** storeId, token, productId, newQuantity, newPrice
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System validates that the product exists by the productId.
* System replaces the old product with the new one that is given.

1. **Alternative flow:**

* **Store not found  
  » The system informs the user that the store does not exist**
* **User is not the owner  
  » The system denies permission to perform the update**
* **Product not found  
  » The system cannot update the product and informs the user**
* **No update data provided  
  » The system cannot perform the update and shows an error**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_ChangeProduct\_with\_valid\_data | 1. Store exists 2. User is store owner 3. Product exists 4. Valid price and/or quantity | 1. Product is updated successfully 2. Changes are reflected in the store |
| Failure\_ChangeProduct\_with\_nonexistent\_store | 1. Store does not exist 2. User is logged in | 1. System shows error 2. Product is not updated |
| Failure\_ChangeProduct\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but is not the owner | 1. System denies request 2. Product is not updated |
| Failure\_ChangeProduct\_with\_nonexistent\_product | 1. Store exists 2. User is owner 3. Product ID is invalid | 1. System shows error 2. Product is not updated |
| Failure\_ChangeProduct\_with\_no\_update\_data\_provided | 1. Store and product exist 2. No price or quantity provided | 1. System shows error 2. Product is not updated |

### Use-case:2.a Add purchase/sale policies in the store:

1. **Actor:** store owner.
2. **Trigger:** add new purchase or sale policies to the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* New policy should be given.

1. **Parameters:** storeId, token, policyDetails
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System adds the new policy to the store.

1. **Alternative flow:**

* **Store not found  
  » The system shows an error and does not proceed**
* **User is not the store owner  
  » The system denies the request**
* **Policy is invalid or malformed  
  » The system rejects the policy and shows a validation error**
* **Duplicate policy exists  
  » The system rejects the request and notifies the user**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddPolicy\_to\_store | 1. Store exists 2. User is store owner 3. Valid and unique policy provided | 1. Policy is added to the store |
| Failure\_AddPolicy\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. Policy not added |
| Failure\_AddPolicy\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but is not the store owner | 1. System denies request 2. Policy not added |
| Failure\_AddPolicy\_with\_invalid\_policy\_data | 1. Store exists 2. User is owner 3. Policy format is incorrect | 1. System shows validation error 2. Policy not added |
| Failure\_AddPolicy\_with\_duplicate\_policy | 1. Store exists 2. User is owner 3. Identical policy already exists | 1. System rejects request 2. No duplicate policy is stored |

### Use-case: 2.b Delete purchase/sale policies in the store:

1. **Actor:** store owner.
2. **Trigger:** delete purchase or sale policies to the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* policy should be given to delete.

1. **Parameters:** storeId, token, policyId.
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System delete the policy from the store.

1. **Alternative flow:**

* **Store not found  
  » The system shows an error and does not proceed**
* **User is not the store owner  
  » The system denies the request**
* **Policy not found in the store  
  » The system shows an error and does not delete anything**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_DeletePolicy\_from\_store | 1. Store exists 2. User is store owner 3. Valid policy exists | 1. Policy is deleted from the store |
| Failure\_DeletePolicy\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No deletion occurs |
| Failure\_DeletePolicy\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the store owner | 1. System denies request 2. Policy is not deleted |
| Failure\_DeletePolicy\_with\_policy\_not\_found | 1. Store exists 2. User is owner 3. Policy ID does not exist in the store | 1. System shows error 2. No deletion occurs |

### Use-case: 3 Add Ownership to a store.

1. **Actor:** store owner, new store owner.
2. **Trigger:** adding a new owner to the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* “New store-owner” exists and identifies as a visitor-Member.
* Store-owner is logged in.

1. **Parameters:** storeId, token, newOwnerId
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the owner is a Visitor-Member and identifies as a store-owner.
* System validates that the newOwner is a Visitor-Member and not identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System makes an offer to the newOwner to give it an ownership.
* newOwner agrees and the System makes it an owner.

1. **Alternative flows:**

* **Nominated user is already an owner**  
  » The system rejects the request and notifies the current owner
* **User is not the owner of the store**  
  » The system denies permission to assign ownership
* **Nominated user rejects the ownership offer**  
  » The system cancels the process and notifies the requesting owner

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddOwnership\_to\_user | 1. Store exists 2. User is store owner 3. Nominated user accepts the request | 1. Nominated user becomes store co-owner |
| Failure\_AddOwnership\_with\_user\_already\_owner | 1. Store exists 2. Nominated user is already an owner | 1. System rejects the request 2. No changes made |
| Failure\_AddOwnership\_with\_user\_not\_store\_owner | 1. Store exists 2. Requesting user is not a store owner | 1. System denies the request 2. Ownership not assigned |
| Failure\_AddOwnership\_with\_nominee\_rejection | 1. Store exists 2. Valid nominee 3. Nominee rejects the invitation | 1. System cancels the process 2. Owner is notified |

### Use-case: 4 Delete Ownership from a store.

1. **Actor:** firststore owner, second store owner.
2. **Trigger:** deleting an owner from the store.
3. **Precondition:**

* Store exists.
* The two store owners exist and identify as a store owner.
* First store-owner is logged in.
* Store owner got his ownership from the first store owne”.

1. **Parameters:** storeId, token, ownerToDelete
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the two owners are a Visitor-Member and identify as a store-owner.
* System validates that these two owners own the given storeId.
* System validates that the second owner got his ownership from the first owner.
* System deletes the ownership from the second owner.
* System deletes all members who received their membership from the second owner.
* System deletes all managers who received their management from the second owner.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and cannot perform the action
* **User is not a store owner**  
  » The system denies the request
* **The revoking owner is not the one who assigned ownership**  
  » The system denies the action and informs the user

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_DeleteOwnership\_from\_appointee | 1. Store exists 2. Both users are owners 3. Remover assigned the target owner | 1. Ownership removed 2. All roles assigned by that owner are also removed |
| Failure\_DeleteOwnership\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No ownership removed |
| Failure\_DeleteOwnership\_with\_user\_not\_owner | 1. Store exists 2. Requesting user is not a store owner | 1. System denies request 2. No ownership removed |
| Failure\_DeleteOwnership\_with\_owner\_not\_assigning\_target | 1. Store exists 2. Target owner was assigned by someone else | 1. System denies action 2. Ownership remains unchanged |

### Use-case:6 Add a manager to a store.

1. **Actor:** store owner, new manager.
2. **Trigger:** adding a manager to the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner..
* Store-owner is logged in.
* Manager exists and a is not a store owner or a manager either.

1. **Parameters:** storeId, token, managerId
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the owner is a Visitor-Member and identifies as a store-owner.
* System validates that the new manager is a Visitor-Member and is not a manager already or an owner either.
* System validates that the owner owns the given storeId.
* System makes an offer to the new manager to give it management.
* New Manager agrees and the System makes it the Manager.
* Owner makes specific authorizations to the new manager.

1. **Alternative flow:**

* **User is not the owner of the store  
  » The system denies the request to appoint a manager**
* **Nominated user is already a manager or owner  
  » The system does not allow assigning duplicate roles**
* **Nominated user rejects the role  
  » The system informs the owner that the appointment was declined**

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddManager\_to\_store | 1. Store exists 2. User is store owner 3. Nominated user is valid and not a manager/owner | 1. Manager request sent 2. Upon approval, manager is added and permissions assigned |
| Failure\_AddManager\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the owner | 1. System denies request 2. Manager is not added |
| Failure\_AddManager\_with\_user\_already\_in\_role | 1. Store exists 2. Nominated user is already a manager or owner | 1. System does not allow duplicate assignment |
| Failure\_AddManager\_with\_nominee\_rejection | 1. Store exists 2. Nominee rejects the manager role | 1. System cancels appointment 2. Owner is notified |

### Use-case:7.a Add authorization for a specific manager.

1. **Actor:** store owner, manager.
2. **Trigger:** change the authorization for a manger in the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Manager exists.
* Manager got his management deal from this store owner.

1. **Parameters:** token, managerId, storeId, authorizations
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the owner is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System validates that the manager got his management deal form this owner.
* Owner adds newAuthorization this manager.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and does not proceed
* **User is not the store owner**  
  » The system denies the request
* **Manager was not appointed by this owner**  
  » The system denies the change of permissions

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_AddAuthorization\_to\_manager | 1. Store exists 2. User is the store owner 3. Manager exists and was appointed by this owner | 1. New authorization is assigned to the manager successfully |
| Failure\_AddAuthorization\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No authorization is added |
| Failure\_AddAuthorization\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the owner | 1. System denies request 2. No authorization is added |
| Failure\_AddAuthorization\_with\_owner\_not\_assigning\_manager | 1. Store exists 2. Manager was not appointed by this owner | 1. System denies request 2. No authorization is added |

### Use-case:7.b Delete(change) authorization from a specific manager.

1. **Actor:** store owner, manager.
2. **Trigger:** change the authorization for a manger in the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Manager exists.
* Manager got his management deal from this store owner.

1. **Parameters:** token, managerId, storeId, authorizations
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the owner is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System validates that the manager got his management deal form this owner.
* Owner deletes authToDelete from this manager.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and does not proceed
* **User is not the owner**  
  » The system denies the request
* **Manager was not assigned by this owner**  
  » The system denies the modification of permissions

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_DeleteAuthorization\_from\_manager | 1. Store exists 2. User is store owner 3. Manager exists and was appointed by the owner | 1. Authorization is removed successfully |
| Failure\_DeleteAuthorization\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No action is performed |
| Failure\_DeleteAuthorization\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the store owner | 1. System denies request 2. Authorization is not removed |
| Failure\_DeleteAuthorization\_with\_owner\_not\_assigning\_manager | 1. Store exists 2. Manager was not appointed by the current owner | 1. System denies request 2. Authorization is not removed |

### Use-case:9 Deactivate store.

1. **Actor:** store owner.
2. **Trigger:** deactivatea store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.

1. **Parameters:** storeId, token
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System deactivates the store.
* System notifies all the managers and the members of this store.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and does not proceed
* **User is not the owner**  
  » The system denies the request and does not deactivate the store

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_DeactivateStore\_by\_owner | 1. Store exists 2. User is the owner 3. Valid token | 1. Store is deactivated 2. All associated managers and owners are notified |
| Failure\_DeactivateStore\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. Store is not deactivated |
| Failure\_DeactivateStore\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the store owner | 1. System denies the request 2. Store remains active |

### Use-case:11 Get Information about store.

1. **Actor:** store owner.
2. **Trigger:** display information from the stores owned by the owner and view the authorizations granted to the management team.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.

1. **Parameters:** token, storeId
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System displays the information requested by the owner.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and does not return information
* **User is not the owner**  
  » The system denies access and does not display any data

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_GetStoreInfo\_by\_owner | 1. Store exists 2. User is logged in as owner 3. Valid store ID is provided | 1. System displays store information including name, roles, and manager details |
| Failure\_GetStoreInfo\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No store info is displayed |
| Failure\_GetStoreInfo\_with\_user\_not\_owner | 1. Store exists 2. User is not the store owner | 1. System denies access 2. Info is not returned |

### Use-case:12 Messages in the store.

1. **Actor:** store owner, member.
2. **Trigger:** members of the store asking questions and the owner replies.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.
* Member exists.

1. **Parameters:** senderId, receiverId, message
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System validates that the user is validating as guest or register
* Owner replies to the message that he got from the member.

1. **Alternative flows:**

* **Store not found**  
  » The system shows an error and cannot retrieve history
* **User is not the owner of the store**  
  » The system denies the request and does not show the data

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_ViewStorePurchaseHistory\_by\_owner | 1. Store exists 2. User is the store owner 3. Valid purchases exist | 1. System displays full purchase history for the store |
| Failure\_ViewStorePurchaseHistory\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. No history is shown |
| Failure\_ViewStorePurchaseHistory\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but not the owner | 1. System denies access 2. User is not allowed to view history |

### Use-case: 13 Show purchases history in the store.

1. **Actor:** store owner.
2. **Trigger:** showing the history of the purchases in the store.
3. **Precondition:**

* Store exists.
* Visitor-Member exists and identifies as a store owner.
* Store-owner is logged in.

1. **Parameters:** storeID.
2. **Main Scenario:**

* System searches and validates that the store exists.
* System validates that the User is a Visitor-Member and identifies as a store-owner.
* System validates that the owner owns the given storeId.
* System shows all the purchases history to the owner.

1. **Alternative flows:**

* **Store not found:** System notifies the owner the store not found.
* **Member is not the owner:** System notifies the user it does not have the permission see this info.

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Results |
| Purchase history displayed successfully. | 1. Store exists  2. Store owner is logged in  3. Store owner owns the store  4.store owner request to show history purchase of specific store by id | System displays all purchase history related to the store that provide the id |
| Store not found | 1. Store does not exist  2. User requests purchase history | System notifies the user that the store is not found |
| User is not the store owner | 1.Store exists  2. User logged in  3. User is not the store owner | System notifies the user that they do not have permission |

## 5.Activities as a store manager:

### Use case 1 – Store manager permissions

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Store manager permissions |
| Actor | User Store |
| Trigger | Manager wants to do some management stuff. |
| Precondition | 1.User is logged in.  2.the user is a store manager.  5.the user has at least one permission . |
| Parameters | User as a store manager. |
| Main Scenario | 1.The store user add for the manager a permission to delete an item.  2.The store manager can delete an item. |
| Alternative flow | 1. **User is not logged in** » The system denies the request and shows an error 2. **Manager has no permissions** » The system denies the action and informs the user 3. **User is not a store manager** » The system denies the action and shows a general error |

**Acceptance tests**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_StoreManagerAction\_with\_valid\_permission | 1. User is logged in 2. Is store manager 3. Has required permission | 1. Action is performed successfully |
| Failure\_StoreManagerAction\_with\_no\_permissions | 1. User is store manager 2. User has no permissions assigned | 1. System denies the action 2. User is informed |
| Failure\_StoreManagerAction\_with\_user\_not\_logged\_in | 1. User is not logged in | 1. Request is denied 2. System shows error |
| Failure\_StoreManagerAction\_with\_user\_not\_a\_manager | 1. User is logged in 2. User is not a store manager | 1. System denies action 2. User is notified |

## 6.Activities as a system manager:

### Use case 1 – Close a store

|  |  |
| --- | --- |
| Use Case Description | |
| Name | Close a store |
| Actor | System manager |
| Trigger | System manager want to close a store. |
| Precondition | Store exist in the system. |
| Parameters | storeId, token |
| Main Scenario | 1.system manager enters the system.  2. manager chooses a store to close permanently.  3. store closes successfully.  4. all managers and owners for this store were notified .  5. all managers and owners for this store were no longer in this position. |
| Alternative flow | **Store not found** » The system shows an error and does not proceed **User is not the owner** » The system denies permission to close the store |

**Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| Test Name | Setup & Parameters | Expected Result |
| Success\_CloseStore\_by\_owner | 1. Store exists 2. User is the store owner 3. Valid token provided | 1. Store is deactivated 2. Owners and managers are notified |
| Failure\_CloseStore\_with\_nonexistent\_store | 1. Store ID does not exist 2. User is logged in | 1. System shows error 2. Store is not deactivated |
| Failure\_CloseStore\_with\_user\_not\_owner | 1. Store exists 2. User is logged in but is not the owner | 1. System denies request 2. Store remains active |