# **Index**

## **Section 1 :System use cases -------2-7**

### Use case 1 :Initialize marketplace system --------------------------8-13

### Use case 2 :System recovery from Configuration File --------------8-13

### Use case 3 :System Initialization from Initial State File -------------8-13

### Use case 4 :Failure Handling in Initialization Process --------------8-13

### Use case 5 :Api Integration – Payment System: pay ----------------8-13

### Use case 6 :API Integration – Supply System: supply---------------8-13

### Use case 7 :Fallback Mechanism for Unavailable Services-------8-13

### Use case 8 :Integrity Rules Enforcement---------------------------8-13

## **Section 2 :** **subscriber use cases ------2-7**

### Use case 1 :register as a new user --------------------------8-13

### Use case 2 :login to the system --------------8-13

### Use case 3 :logout from the system -------------8-13

### Use case 4 :view purchase history --------------8-13

### Use case 5 :submit product review ----------------8-13

### Use case 6 :submit store review --------8-13

### Use case 7 :view product reviews ---------------8-13

### Use case 8 : view store reviews ------8-13

### Use case 9 : rate store ------------------------

### Use case 10 : rate product -------------------------

### Use case 11 : buy cart items --------------

### Use case 12 : open store------------------

## **Section 3 : guest use cases------**

### Use case 1 : enter as a guest

### Use case 2 : guest exit the system

### Use case3 : search for a specific store in the system

### Use case 4 :search for a product in the system

### Use case 5 : search for product in a specific store

### Use case 6: view store details

### Use case 7 :view item information

### Use case 8 : view cart items ---------

### Use case 9 : view products in the store

### Use case 10 :edit item quantity in the cart

### Use case 11 :remove item from a cart

### Use case 12 :buy cart items as a guest

### Use case 13: add product to a cart

## **Section 4 : manager use cases**

### Use case 1 : perform actions based on given permissions

### Use case 2 : view permissions

### Use case 3: add permissions to manager

### Use case 4 : remove permissions from manager

## **Section 5: owner use cases**

### Use case

### Use case

### Use case

### Use case

## **Section 6 : admin use cases**

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

## **Section 7 : special purchase use cases**

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

## 

## **Section 8 : notification use cases**

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

### Use case

# 

# **Section 1 - System use cases:**

## Use case 1 :Initialize marketplace system

**Actor: Admin (or automated system startup)  
Trigger: Application startup or explicit admin “initialize” command  
Preconditions:**

* **A valid configuration file is present**
* **Database connection parameters are configured and reachable**
* **The system has not yet been initialized**

**Main Flow:**

1. **The system loads configuration settings.**
2. **It validates the provided initialization parameters (admin key, user name, password).**
3. **It creates the initial administrator account.**
4. **It marks the system state as “initialized.”**
5. **It returns success to the caller.**

**Alternative Flows:**

* **Invalid or missing parameters → initialization is rejected with a “Bad Request” error**
* **Admin key doesn’t match → initialization is rejected with a “Forbidden” error**
* **Persistence failure (e.g. DB down) → initialization fails with an “Internal Server Error”**
* **System already initialized → request is rejected with a “Conflict” error**

**Acceptance Tests:**

## Use case 2 : System Recovery from Configuration File-

**Actor: System (Spring Boot startup)  
Trigger: Application startup  
Preconditions:**

* **Configuration files exist and are readable**
* **Database is accessible**

**Main Flow:**

1. **The framework loads all configuration properties.**
2. **It binds each property into the system’s settings (connection info, security keys, scheduling parameters).**
3. **It establishes the database connection.**
4. **It validates (and if needed updates) the schema.**

**Alternative Flows:**

* **Missing/unreadable config → startup aborts with configuration binding errors**
* **Malformed property value → startup aborts with a binding exception**
* **Database connection failure → startup aborts with a connection‐error**
* **Schema validation/update error → startup aborts with a schema‐management error**

Acceptance test :

## Use case 3 : System Initialization from Initial State File

**Actor: System (on first startup or admin‐triggered)  
Trigger: Initialization flag on startup or explicit “load initial data” command  
Preconditions:**

* **An initial-state script file is present**
* **System is not yet marked “initialized”**

**Main Flow:**

1. **Open and read the script file.**
2. **Split its content into discrete commands.**
3. **For each command: parse its arguments and invoke the corresponding domain operation (e.g., “create user,” “open store,” “add item,” “place bid”).**
4. **After all commands complete successfully, mark the system as “initialized.”**

**Alternative Flows:**

* **Missing/unreadable file → initialization aborted, system remains uninitialized**
* **Unknown command → that line is skipped with a warning; processing continues**
* **Parse error (wrong arguments) → that line is skipped with an error log; processing continues**
* **Domain‐service failure → abort the rest of the script and leave the system uninitialized**

Acceptance test :

## Use case 4 : Failure Handling in Initialization Process

**Actor:** System  
**Trigger:** Any error during Use Case 1, 2, or 3  
**Preconditions:** Initialization or recovery is in progress

**Main Flow:**

1. The system catches exceptions or error flags.
2. It logs detailed error information.
3. It rolls back any partial changes (e.g., clears data or resets flags).
4. It halts further initialization steps.
5. It notifies the administrator (via log, email, or alert).

**Alternative Flows:**

* Transient errors (e.g. temporary I/O hiccups) → may retry a fixed number of times before giving up
* Persistent errors → full abort and require manual intervention

Acceptance test :

## Use case 5 : API Integration – Payment System: Pay :

**Actor:** System   
**Trigger:** User submits an order/payment request  
**Preconditions:**

* Order details are complete (items, amounts)
* Payment information is provided (card or account credentials)

**Main Flow:**

1. The system validates the payment request parameters.
2. It constructs an HTTP/API call to the external payment gateway.
3. It sends the request and awaits the response.
4. On success, it records the transaction ID and updates the order status to “paid.”
5. It returns confirmation (e.g. receipt) to the caller.

**Alternative Flows:**

* Missing/invalid payment data → return a “Bad Request” error
* Gateway returns an error (e.g. insufficient funds) → return a “Payment Failed” error
* Communication timeout or network error → trigger retry logic or return a “Service Unavailable” error

Acceptance test :

## Use case 6 - API Integration – Supply System: Supply:

**Actor:** System (e.g. PurchaseService)  
**Trigger:** A paid order is ready to be dispatched and the system must request shipment from the external supply service.

**Preconditions:**

* The supply‐service endpoint URL and credentials are configured in application.properties
* The order exists and has status “PAID”
* A valid shipping address (street, city, postal code, country) is associated with the order

**Parameters:**

* orderId (long) – identifier of the order to ship
* address (object) – includes street, city, postalCode, country
* items (list) – each with productId and quantity

**Main Flow:**

1. **Validate input:** the system checks that orderId refers to an existing paid order and that all address fields are non-null and non-empty.
2. **Build request payload:** it constructs the JSON (or form-encoded) body containing the orderId, address and items according to the supply-service API schema.
3. **Invoke external API:** the system sends an HTTP POST to the configured supply endpoint (/api/supply) with the payload.
4. **Process response:** on HTTP 200, it parses the response body to extract a supplyTransactionId.
5. **Update domain state:** it records supplyTransactionId on the order, sets the order status to “SHIPPED,” and persists these changes.
6. **Return success:** the service method returns a confirmation (e.g. the supplyTransactionId) to the caller.

**Alternative Flows:**

* **Missing or invalid parameters** → system returns a “400 Bad Request” error indicating which field is invalid.
* **Supply-service business error** (e.g. out of inventory) → API returns 4xx/5xx → system logs the error and returns a “Supply Failed” exception to the caller.
* **Network timeout or connection failure** → system may retry (according to a retry policy) or immediately return a “503 Service Unavailable” error.
* **Unexpected response format** → system throws a “Supply Response Parsing Error” and flags the order as “SUPPLY\_ERROR” for manual intervention.

## Use case 7 - Fallback Mechanism for Unavailable Services :

**Actor:** System  
**Trigger:** An external service call (e.g., payment or supply API) fails permanently (timeout or 5xx error)  
**Preconditions:**

* A primary service endpoint is configured
* A fallback (secondary) endpoint or offline mode strategy is defined

**Main Flow:**

1. The system attempts to call the primary external service.
2. It detects a permanent failure (e.g. HTTP 503 or network unreachable).
3. It logs the primary-service failure with details.
4. It looks up the configured fallback endpoint or strategy.
5. It invokes the fallback service call or enters offline mode (e.g., queue the request locally).
6. On successful fallback response, it records the fallback usage in logs or metrics.
7. It returns a success response (or queued‐for-later acknowledgment) to the caller.

**Alternative Flows:**

* **Fallback call fails or not configured** → the system logs a critical error and returns a “Service Unavailable” error to the caller.
* **Fallback mode is “queue for retry”** → the request is enqueued for later retry, and a “Request Queued” acknowledgment is returned.

## Use case 8 : Integrity Rules Enforcement :

**Actor:** System  
**Trigger:** Any operation that updates multiple related domain entities (e.g., purchase, bid placement, stock adjustment)  
**Preconditions:**

* All input parameters have basic validity (non-null, correct types)
* Domain invariants are known (e.g., stock ≥ 0, user balance ≥ 0, store exists)

**Main Flow:**

1. The system opens a transactional context.
2. It performs each sub-operation in turn (e.g., debit user, decrement stock, record order).
3. After each sub-operation, it runs the relevant integrity check (e.g., new stock level ≥ 0, user balance not negative).
4. If all checks pass, it commits the transaction.
5. It returns success to the caller.

**Alternative Flows:**

* **Integrity check failure** (e.g., insufficient stock) → the system rolls back the transaction and returns a “Business Rule Violation” error.
* **Unexpected persistence error** (e.g., DB constraint violation) → the system catches the database exception, rolls back, and returns a “Data Integrity Error” to the caller.

## Use case

## Use case

## Use case

# **Section 2 –** **subscriber use cases:**

## Use case 1 -register as a new user:

Actor: guest

Trigger: the guest submits a registration request to the system

Precondition:

* a valid guest session exists
* The chosen username is not used already

parameters:

* token
* age
* username
* password

main flow :

* the system validates the token of the guest
* check the username
* encode the password
* create and save the new user

Alternative flow:

* if the token is invalid -> the system returns "INVALID\_TOKEN" error
* if the username already taken -> the system returns "USERNAME\_USED" error

Acceptance test:

## Use case 2 - login to the system:

Actor: subscribed user

Trigger: the user attempts to log in to the system

Precondition:

* The user is already registered in the system
* A valid token

Parameters:

* token
* username
* password

main flow :

* the system verifies the token
* fetches the user by the username
* check if the password is correct
* generates a user token and loges in the user

Alternative flow:

* If the token is invalid -> the system returns " INVALID\_TOKEN" error
* if the username or the password is incorrect -> the system returns " WRONG\_PASSWORD" or " USER\_NOT\_FOUND" errors

Acceptance test:

## Use case 3 - log out from the system:

Actor: subscribed user

Trigger: the user tries to log out of the system

Precondition:

* the user already requests in the system and also logged in
* the token is valid

Parameter:

* user authentication token

main flow :

* the system validate the token
* loges the user out

Alternative flow:

* if the token is invalid -> the system ignores the request or return " INVALID\_TOKEN " error
* if the user not found -> the system return " USER\_NOT\_FOUND" error

Acceptance test:

## Use case 4 - view purchase history:

Actor: subscribed user

Trigger: the user request to view their past purchases

Precondition:

* The user is authenticated
* The user is successfully logged in

Parameter:

* token of the user

main flow :

* the system verifier the token
* find the user
* fetch all the orders of the user
* return it as a list

Alternative flow:

* if the token is invalid -> the system returns a " INVALID\_TOKEN " error
* if the user not found in the system -> the system return a " USER\_NOT\_FOUND" error
* if the user has no purchases -> an empty list is returned

Acceptance test:

## Use case 5 - submit product review:

Actor: subscribed user

Trigger: user submits a review for a specific product

Precondition:

* the user is authenticated ( have a valid token )
* the store exist in the system
* the product exists in the selected store
* the user is not suspended

Parameters:

* token of the user
* store id that contains the product
* productid
* The test of the review

Main flow :

* the system verifies the token
* check that the store exists in the system
* check if the store has the product
* check if the user is not suspended
* save the review in the database

Alternative flow:

* If the token is invalid (the user does not authenticated ) -> the system returns an authentication error
* if the user is suspended -> the system returns a "USER-SUSPENDED" error
* if the store does not exist -> the system returns a " STORE\_NOT\_FOUND" error
* if the product does not exist -> the system returns a " PRODUCT\_NOT\_FOUND" error

Acceptance test:

## Use case 6 - submit store review:

Actor: subscribed user

Trigger: user submits a review for a specific store

Preconditions:

* The user must be registered and logged in the system
* the user does not suspend
* the store exists in the system

Parameters:

* user token
* store id
* text of the review

Main flow:

* the system checks the token
* check if the store exists
* check if the user is suspended or not

Alternative flow:

* if the token is invalid -> the system returns a " INVALID\_TOKEN" error
* if the store does not exist -> the system returns a " STORE\_NOT\_FOUND" error
* if the user is suspended -> the system returns a " USER\_SUSPENDED" error

Acceptance test:

## Use case 7 - view product reviews:

Actor: user in the system

Trigger: the user want to see the product review in a specific store

Precondition:

* the product has to be existed in the system
* the store has contained the product
* The product exists in the store

Parameters:

* Storeid
* Productid

Main flow:

* The user asked for the review of the product
* The system finds the store by the given id
* Check if the store countian the product id
* Get the list of the product review
* Return the list

Alternative flow:

* If the store does not found -> the system return a " STORE\_NOT\_FOUND" error
* If the store does not countian the product id -> the system return a " PRODUCT\_NOT\_FOUND" error

Acceptance test:

## Use case 8 - view store reviews:

Actor: user in the system

Trigger: the user want to see the review of store

Precondition:

* The store id is exist in the system

Parameters:

* Store id

Main flow:

* The system check if the store exists
* Fetch the reviews in a list
* Return the list

Alternative flow:

* If the store does not exist in the system -> the system return a " STORE\_NOT\_FOUND" error

Acceptance test:

## Use case 9 – rate store :

Actor: user in the system

Trigger: the user tries to rate the store

Precondition:

* The user token valid
* The user does not suspended
* The store exist in the system

Parameters:

* Token
* Storeid
* Number between 1 to 5 (rate )

Main flow:

* The user aske to rate the store
* The system check the token of the user
* The system check if the store exist
* Load the store entity and invokes the new rate

Alternative flow:

* If the user token invalid -> the system return a " INVALID\_TOKEN" error
* If the user is suspended -> the system return a " USER\_SUSPENDED" error
* If the store does not found -> the system return a " STORE\_NOT\_FOUND" error

Acceptance test:

## Use case 10 – rate product :

Actor: user in the system

Trigger: the user tries to rate a product in a specific store

Precondition:

* Valid user token
* the user have to be unsuspended
* the store exist in the system
* the store have to countian the product

Parameters:

* token of the user
* store id
* product id
* number between 1 and 5 ( the rate of the product )

Main flow:

* the user asked for add new rank to a product
* the system check the token of the user
* check if the user is loges in
* check if the user is suspended
* load the store from the system
* search the product in the store stock
* and then add a rank

Alternative flow:

* if the token is invalid –> the system return a " INVALID\_TOKEN" error
* if the user does not log in -> the system return a " USER\_NOT\_LOGGED\_IN" erroe
* if the user suspended -> the system return a " USER\_SUSPENDED" error
* if the product not in the store stock -> the system return " DevException" error

Acceptance test:

## Use case 11 – buy cart items :

Actor : subscribed user

Trigger :the subscribe user want buy his shopping cart

Precondition :

* valid token
* the user have to be not suspended
* the user have to be registered
* the cart have not be empty

Parameters:

* token
* payment
* supply

Main flow:

* the system check the validity of the token
* check if the user suspended or not
* check the cart
* check each amount of item in the cart in the store stock
* update the details in the receipt of the user
* clean the cart

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the user is suspended -> the system return a " USER\_SUSPENDED" error
* if the suer is does not logged in -> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the cart details (object) does not found -> the system return a " CART\_NOT\_FOUND" error
* if the store does not exist in the system -> the system return a " STORE\_NOT\_FOUND" error
* if the payment service failed -> the system return a " PAYMENT\_ERROR" error
* if the supply service have an error -> the system return a " SUPPLY\_ERROR" error
* if the user nor registered -> return user not registered error

acceptance test : **Acceptance tests**

## Use case 12 – open store :

Actor: subscribed user

Trigger: a user open a new store in the system

Precondition:

* avaalid user token
* the user have to be registered to the system
* the user have to be unsuspended

Parameters :

* token
* store name
* category

Main flow :

* the user send an open store requests
* the system check the token
* extract the user id
* check if the user is registered
* check if the user suspended or not
* new store object
* and the owner of the store
* add the store purchase and stock …
* return store id

Alternative flow:

* if the token is unvalid -> the system return a " INVALID\_TOKEN" error
* if the user is not registered -> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the user is suspended -> the system return a " USER\_SUSPENDED" error
* if the user nor registered -> return user not registered error

Acceptance test :

# **Section 3 – guest use cases:**

## Use case 1 - enter as a guest:

Actor: guest

Trigger: a user enter the system without make a registration

Precondition:

* system initialize

Parameter: None

Main flow:

* guest enter
* the authentication make a JWT token

Alternative flow:

* if something went wrong -. The system move the user to "not found 404" page

Acceptance test:

## Use case 2 - guest exit the system:

Actor :guest

Trigger: the guest exit the system

Precondition: the guest have a valid token

Parameter:

* token

Main flow:

* the guest exit the system
* system invalidate the guest token

Alternative flow :

* if the guest have already invalid token -> the system return a " INVALID\_TOKEN" error

Acceptance test :

## Use case3 - search for a specific store in the system:

Actor: user in the system (guest or registered)

Trigger: the user searches for a specific store in the system

Precondition:

* the user has valid token
* the store exists in the system

Parameter:

* token
* store id

Main flow:

* the user searches for a specific store in the system
* the system gets all the stores in the system
* return the store object

Alternative flow:

* if the user has an invalid token -> the system returns a " INVALID\_TOKEN" error
* if the store does not exist in the system -> the system returns a " STORE\_NOT\_FOUND" error

Acceptance test:

## Use case4 - search for a product in the system:

Actor : user in the system

Trigger: the user search a specific product in the system (all the stores)

Precondition:

* the user has a valid token
* search model defined

Parameter:

* token
* optional filter (name / keyword/ category name/min price / max price / min rank / max rank )

Main flow:

* the system builds a product search category
* save the category
* check the token
* search in the stock and received a match list of products
* if the search is in a "full system " -> ignore the store id
* return a list of the products that fit the filter

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the system has no product -> the system returns a empty result
* if the name or the keyword not provided -> the system returns a " AI\_NOT\_WORK" error

Acceptance test :

## Use case5 - search for product in a specific store:

Actor : user in the system

Trigger: the user want to find a product in a specific store

Precondition:

* the user token is valid
* the search criteria have to specific a name / keyword
* and the store is in the system

Parameter:

* token
* store id
* optional filter (name / keyword/ category name/min price / max price / min rank / max rank )

Main flow:

* the system builds a product search category
* save the category
* check the token
* search in the stock and received a match list of products
* the system fitch the items that received according to the filter and the store id
* return a list of the products that fit

Alternative flow :

* if the user have invalid token -> the system returns a " INVALID\_TOKEN" error
* if the system does not have a search mode specified -> the system return a " AI\_NOT\_WORK" error
* if the keyword search fails -> the system return empty result
* if the store does not found -> the system return a " STORE\_NOT\_FOUND" error

Acceptance test :

## Use case6 - view store details:

Actor : user in the system

Trigger: the user want to see the store details

Precondition:

* valid token
* store exist in the system

Parameter:

* token of the user
* id of the store

Main flow:

* the system get the user request
* validate the token
* load the store entity and converts it to DTO and return it

Alternative flow :

* if the user token is invalid -> the system return a " INVALID\_TOKEN" error
* if the store does not exist in the system -> the system return a " STORE\_NOT\_FOUND" error

Acceptance test :

## Use case 7 - view item information:

Actor : user in the system

Trigger: the user send request to retrieve information about the item in a store

Precondition:

* a valid user token
* the product is exists in the system

Parameter:

* token
* product id

Main flow:

* the user send the request
* the system check the validation of the token
* the system check if the product exist in the system
* and then return the result if found

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the product does not found in the system -> the system return a " PRODUCT\_NOT\_FOUND" error

Acceptance test :

## Use case 8 - View Regular Cart Items

Actor : user in the system

Trigger: a user in the system trying to see the item that he was added to the cart

Precondition:

* a valid token
* the user have zero or at least one item in the cart

Parameter:

* token

Main flow:

* the user asked for view cart
* the system check the validation of the token
* the system retrieve the user regular cart and return the Dto result

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the cart is empty -> the system return an empty list

Acceptance test :

## Use case 9 - view products in the store:

Actor : user in the system

Trigger: the user want to see all the store product

Precondition:

* the store id is exist in the system
* the user have a valid token
* the store have a stock

Parameter:

* store id

Main flow:

* the user asked for the products in a specific store
* check if the store exist
* take the stock of the store
* take the information of each product in the store stock
* return result

Alternative flow :

* if the store does not exist in the system -> the system return a " STORE\_NOT\_FOUND" error
* if the stock of the store does not exist -> the system return a " STOCK\_NOT\_FOUND" error if we have a data inconsistency during the process -> the system return error

Acceptance test :

## Use case 10 - edit item quantity in the cart

Actor : user in the system

Trigger: the user want to update a quantity of an existing item in his cart

Precondition:

* the token is valid
* his cart contains at least one of the item id

Parameter:

* token
* item id
* number ( new quantity )

Main flow:

* check the validation of the token
* get the user cart
* search the item in the user cart
* check if the quantity is allowed ( more than 0 and less than available items in the stock )
* update the quantity

Alternative flow :

* if the user have invalid token -> the system return a " INVALID\_TOKEN" error
* if the cart item does not exist -> the system return a "ITEM\_NOT\_IN\_CART" error
* if the quantity in invalid -> the system return a "INVALID\_QUANTITY" error if it less than 0 and "QUANTITY\_EXCEEDS\_STOCK" error if it more than the stock

Acceptance test :

## Use case 11 - remove item from a cart

Actor : user in the system

Trigger: the user want to remove the item from his cart

Precondition:

* the token of the user is valid
* the cart contains the item

Parameter:

* token
* item id

Main flow:

* the system check the validation of the token
* search cart item by the item id scoped for the user
* remove it from the cart

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the cart item does not found in the cart -> the system return a " ITEM\_NOT\_IN\_CART" error

Acceptance test :

## Use case 12- buy cart items as a guest

Actor : guest

Trigger:the guest want to buy the items that in his cart

Precondition:

* the guest have a valid token
* the cart is not empty

Parameter:

* token
* payment details
* supply details

Main flow:

* the system check the validation of the token
* fetch the guest regular cart
* validate the stock availability of each item in the cart
* calculate total price and check the discount policies
* call the external payment service
* call the external supply service
* persist the order and delete the guest cart

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the cart is empty -> the system return a " CART\_IS\_EMPTY" error
* if the payment service fails -> the system return a " PAYMENT\_FAILED " error
* if the supply service fails -> the system returns a " SUPPLY\_FAILED" error

Acceptance test :

## Use case 13- add product to a cart

Actor : user in the system

Trigger: the user chooses a product in the system and want to add it in his cart

Precondition:

* the token is valid
* the product exists in the system

Parameter:

* token
* product id
* store id
* quantity

Main flow:

* the system check the validation of the token
* add or make a user cart
* check if the product exist
* check the stock availability
* if the product is not already in the cart the system make a new cart object and add it to the user cart
* update the cart of the user

Alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the product not found -> the system return a " PRODUCT\_NOT\_FOUND" error
* if the quantity less than 1 -> the system return a " INVALID\_QUANTITY" error
* if the quantity more than the store stock -> the system return a " QUANTITY\_EXCEEDS\_STOCK" error

Acceptance test :

# **Section 4 - manager use cases :**

## Use case 1 - perform actions based on given permissions:

Actor: store manager

Trigger: the manager request to do one of the permissions function

Precondition:

* the manager have a valid token
* the manager account is not suspended
* the manager assigned to the target store
* the store exist in the system and active
* the manager have a permission to the choosing function

Parameter:

* token
* store id
* action type
* action parameters

Main flow:

* the system received the action request
* the system check the validation of the manager token
* the system check if the manager suspended
* the system load the store and check his activation
* the system check the permissions of the manager
* the system gave the manager to make the action or not

Alternative flow:

* if the token invalid -> the system return a " AUTH\_TIMEOUT" error
* if the manager suspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist in the system -> the system return a " STORE\_NOT\_FOUND" error
* if the store not active -> the system returns a " DEACTIVATED\_STORE" error
* if the manager have no permission to do the action -> the system return a " NO\_PERMISSION"
* for each one of the permissions : if the manager cant or have error to make the action the system return an appropriate error type

Acceptance test:

## Use case 2 - view permissions:

Actor: manager

Trigger: the user manager in the system want to see his permission

Precondition:

* the manger token is valid
* the manager account is not suspended
* the manager assigned to the target store
* store is exist in the system and active
* the manager is registered in the system

Parameter:

* token
* store id

Main flow:

* the system check the validation of the manager token
* the system received all the worker in the store as a node list from the suconnection
* check if the store exist in the system
* check if the manager exist in the store workers and it’s a manager
* check if the store is active
* search to the manager from the worker list
* check if the manager is active
* and get his permissions

Alternative flow:

* if the token is invalid or expired -> the system return a " AUTH\_TIMEOUT" error
* if the manager suspended -> the system return a " USER\_SUSPENDED" error
* if the store does not found in the system -> the system return a " STORE\_NOT\_FOUND" error
* if the store not active -> the system returns a " DEACTIVATED\_STORE" error

Acceptance test:

## Use case 3 – add/change permissions to manager:

Actor: store owner or a manager with permission

Trigger: the user want to change the permission od specific manager

Precondition:

* the request carries have a valid token
* the request carries is owner
* the request carries is in the same store with the manager

Parameter:

* token
* store id
* manager id

Main flow:

* the system check if the token valid
* the system check if the user is owner
* load the store aggregate
* check if the managerid exist in the store as a manager user
* validate the permission
* update the managerid permission

Alternative flow:

* the token unvalid -> the system return a " INVALID\_TOKEN" error
* the caller is not owner -> the system return a " NOT\_STORE\_OWNER" error
* manageid not found in the store managers -> the system return a " MANAGER\_NOT\_FOUND" error
* if the added permission is invalid -> the system return a " INVALID\_PERMISSIONS" error

Acceptance test:

# **Section 5 – owner use cases :**

## Use case 1 - view owned stores:

Actor: store owner

Trigger: the user want to see his owned store

Preconditions:

* the store owner token is valid
* the owner have to be logged in
* the store owner unsuspended

Parameters:

* store id

Main flow:

* the system check the validation of the token
* get from the suConnectionRepo class all the stores id in the system that owned
* return a stores Dto in a list

Alternative flow:

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the owner does not logged in or registered -> the system return a "USER\_NOT\_LOGGED\_IN" error
* if the store id not exist -> the system return " STORE\_NOT\_FOUND" error

Acceptance test:

## Use case 2 - Change store purchase/sale policy:

Actor: owner / manager with this permission on a store

Trigger: the store owner request to add or remove a purchase policy

Preconditions:

* the owner token is valid
* the owner is registered and logged in to the system
* the store exist
* the store is active
* the manager have a permission

Parameters:

* token
* store id
* policy key
* parameter

Main flow:

* check the validation of the token
* check if the requested is online and unsuspended
* load the store from the system
* from the suconnection check if the requesting userhave the permission
* update the policies in the store

Alternative flow:

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the requesting useris suspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist -> the system return a " STORE\_NOT\_FOUND" error
* if the store inactive -> the system return a " STORE\_NOT\_ACTIVE" error
* if the user requested does not have permission -> the system return a " NO\_PERMISSION" error
* when the key is unknown -> the system return a "NO\_POLICY" error

Acceptance test:

## Use case 3 - add an existing product to this store’s stock:

Actor: owner

Trigger: the owner want to add a product to the store stock

Preconditions:

* store exist in the store
* the token of the requesting useris valid
* product id is already exist in the system

Parameters:

* token
* store id
* product id
* quantity
* price
* category

Main flow:

* check the validation of the token
* check if the requesting useris logged in and unsuspended
* check the permission of the requested user
* check if the product exist in the system
* persisting a new stock item and add it to the store stock

Alternative flow:

* if the token is invalid -> the system returns a " INVALID\_TOKEN" error
* if the user does not logged in -> the system returns a "USER\_NOT\_LOGGED\_IN"
* if the user is suspended -> the system return a " USER\_SUSPENDED" error
* if the store not exist in the system -> the system return a "STORE\_NOT\_FOUND" error
* if the requesting userhaven’t the permission -> the system return a " NO\_PERMISSION" error
* if the product not found in the system -> the system return a " PRODUCT\_NOT\_FOUND" error

Acceptance test:

|  |  |  |
| --- | --- | --- |
| 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 4 - Remove an item from this store’s stock:

Actor: owner / manager with a permission

Trigger: the owner of the store request to delete item from the store stock

Preconditions:

* the token of the requesting user is valid
* the requesting useris logged in and unsuspended
* the requesting userhave the permission to remove action

Parameters:

* token
* store id
* product id

Main flow:

* check the validation of the token
* check the log in and the unsuspended user status
* load the store aggregate and search to the store id
* check the permission of the requesting user
* check if the product in the store stock
* remove the product

Alternative flow:

* in the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the requesting usernot logged in -> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the requesting usersuspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist -> the system return a " STORE\_NOT\_FOUND" error
* if the requesting userdoes not have a permission -> the system return a " NO\_PERMISSION" error

Acceptance test:

|  |  |  |
| --- | --- | --- |
| 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 5 - Update the available quantity / price of an item:

Actor: store owner or manager with permission

Trigger: the user send a request to update the product quantity (or price)

Preconditions:

* a valid token
* the requesting useris logged in and unsuspended
* the store exist
* the user have permission

Parameters:

* store id
* token
* product id
* new quantity / new price

Main flow:

* check the validation of the token
* load the store and check the permission
* change quantity or price for the product

Alternative flow:

* in the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the requesting usernot logged in -> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the requesting usersuspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist -> the system return a " STORE\_NOT\_FOUND" error
* if the requesting userdoes not have a permission -> the system return a " NO\_PERMISSION" error

Acceptance test:

## Use case 6 – add discount:

Actor: owner or a manager with a permission

Trigger: the requesting user add new discount

Preconditions:

* a valid token
* the requesting user logged in and unsuspended
* the store exist in the system
* the requesting user have a permission

Parameters:

* store id
* token
* name
* percent
* type
* condition
* logic
* sub discount names

Main flow:

* check the requested token validation
* load the store and check the requesting user permission
* search discount in the store and remove it

Alternative flow:

* in the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error
* if the discount does not exist -> the system returns a "DISCOUNT\_NOT\_FOUND" error

Acceptance test:

## Use case 7 – remove discount:

Actor: owner or a manager with a permission

Trigger: the requesting user remove new discount

Preconditions:

* a valid token
* the requesting user logged in and unsuspended
* the store exist in the system
* the requesting user have a permission

Parameters:

* token
* store id
* discount name

Main flow:

* check the requested token validation
* load the store and check the requesting user permission
* for each sub discount names in the store , remove it and add to a list
* create new discount aggregate add it to the store

Alternative flow:

* in the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error
* if the discount does not exist -> the system returns a "DISCOUNT\_NOT\_FOUND" error

Acceptance test:

## Use case 8 – add ownership to the store :

Actor: owner

Trigger: owner of the specific store add new-owner to the store

Preconditions:

* valid token
* owner registered and logged in
* store exist
* the owner holds ownership of the store id
* new-owner is registered and logged in the system

Parameters:

* token
* store id
* new-owner user name

Main flow:

* check the validation of the owner token
* check the log in and registration of the owner
* check if the owner suspended
* check the registration and log in status of the new-owner
* load and validate the store
* check the permission of the owner
* create offer and send it

Alternative flow:

* if the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the new-owner does not logged in -> the system return a "USER\_NOT\_LOGGED\_IN" error
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the store inactive -> the system return a "DEACTIVATED\_STORE" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error

Acceptance test:

## Use case 9 – remove ownership from the store :

Actor: owner

Trigger: owner of the specific store remove-owner from the store

Preconditions:

* valid token
* owner registered and logged in
* store exist
* the owner holds ownership of the store id
* remove-owner is registered and logged in the system and owner of the store

Parameters:

* token
* store id
* remove-owner

Main flow:

* check the validation of the owner token
* check the log in and registration of the owner
* check if the owner suspended
* check the registration and log in status of the new-owner
* load and validate the store
* check the permission of the owner
* check if the remove-owner an owner in the store
* remove the remove-owner from the store

Alternative flow:

* if the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the remove-owner does not logged in -> the system return a "USER\_NOT\_LOGGED\_IN" error
* if the remove-owner not in the list of the store owners -> the system return a "
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the store inactive -> the system return a "DEACTIVATED\_STORE" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error

Acceptance test:

## Use case 10 – add manager to the store :

Actor: owner

Trigger: the store owner want to add a new manager to the store

Preconditions:

* valid token
* owner registered and logged in
* store exist and active
* the owner holds ownership of the store id
* new manager is registered and logged in the system

Parameters:

* token
* store id
* manager name
* permission list

Main flow:

* validate the token
* check if the owner online and unsuspend
* search the new manager in the system
* search the store and check active
* check the permissions of the requesting user
* send offer

Alternative flow:

* if the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the new-manager does not logged in -> the system return " USER\_NOT\_FOUND" error
* "USER\_NOT\_LOGGED\_IN" error
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the store inactive -> the system return a "DEACTIVATED\_STORE" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error

Acceptance test:

## Use case 11 – remove manager from the store :

Actor: owner

Trigger: the store owner want to remove manager from the store

Preconditions:

* valid token
* owner registered and logged in
* store exist and active
* the owner holds ownership of the store id
* new manager is registered and logged in the system manager in the store

Parameters:

* token
* store id
* manager id

Main flow:

* validate the token
* check if the owner online and unsuspend
* search the new manager in the store managers
* search the store and check active
* check the permissions of the requesting user
* send offer

Alternative flow:

* if the token is invalid -> the system returns a "INVALID\_TOKEN" error
* if the user doesn’t logged in -> the system returns a "USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system returns a "USER\_SUSPENDED" error
* if the new-manager does not logged in -> the system return " USER\_NOT\_FOUND" error
* if the manager does not manager in the store -> the system return a " MANAGER\_NOT\_FOUND" error
* "USER\_NOT\_LOGGED\_IN" error
* if the store not exist -> the system returns a "STORE\_NOT\_FOUND" error
* if the store inactive -> the system return a "DEACTIVATED\_STORE" error
* if the user has not a permission -> the system returns a "NO\_PERMISSION" error

Acceptance test:

## Use case 12 – reopen store :

**Actor**

**Trigger**

**Precondition**

**Parameters**

**Main flow**

**Alternative flow**

**Acceptance test**

# **Section 6 – system admin use cases :**

## Use case – close store in the system :

Actor: admin

Trigger : the admin trying to close specific store

Precondition :

* the token is valid
* the user that requesting is admin
* the store exist

parameters :

* token
* store id

main flow :

* the system check the validation and the admin
* search the store in the system '
* close the store
* delete owner/ manager from the store

alternative flow :

* if the token is invalid -> the system returns a " INVALID\_TOKEN" error
* if the user not a system admin -> the system return a " NOT\_ADMIN" error
* if the store not exist -> the system return a " STORE\_NOT\_FOUND" error

acceptance test :

## Use case – suspend user :

Actor : admin

Tripper : the admin want to suspend user in the system

Precondition :

* admin token is valid
* and the requesting user admin
* the user to suspend exist in the system
* the target user is not suspended already

parameters :

* token
* time
* user id

main flow :

* check the token validation
* check the admin status
* search the user in the system
* check if the user suspended already or not
* suspend the user

alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the admin not found -> the system return a " USER\_NOT\_FOUND" error
* if the requesting user doesn’t admin -> the system return a " NO\_PERMISSION" error
* if the target user already suspended -> the system return a " USER\_SUSPENDED" error

acceptance test :

## Use case – unsuspend user :

Actor : admin

Trigger: the admin want to unsuspend user

Precondition :

* the requesting user is registered to the system and admin
* the target user exist
* the target user suspend

Parameters:

* user id
* token

Main flow :

* check the admin validation and admin status
* load the list of the suspended user
* search to the user and unsuspend him

Alternative flow:

* if the toke invalid -> the system return a " INVALID\_TOKEN" error
* if the user is not admin -> the system return a " NO\_PERMISSION" error
* if the target user in not suspend -> the system return a " SUSPENSION\_NOT\_FOUND" error

Acceptance test :

## Use case – view suspended users

Actor : admin

Trigger:the admin want to see the suspended users in the system

Precondition :

* the token of the admin is valid
* the admin is registered to the system and admin

Parameters:

* token

Main flow :

* the system check the validation of the token
* check if the requesting user is admin
* fetch all suspended user
* and return

Alternative flow:

* if the toke invalid -> the system return " INVALID\_TOKEN" error
* if the requested user not registered -> the system return a " USER\_NOT\_FOUND" error
* if the requesting user is not admin -> the system return a " NO\_PERMISSION" error

Acceptance test :

## Use case – add new admin to the system

Actor : admin

Trigger: the admin want to promote user to admin

Precondition :

* the admin token is valid
* the admin is registered and logged in to the system
* the requesting user is admin
* the new-admin registered to the system and logged in

Parameters:

* token
* admin key
* user id

Main flow :

* check the validation of the token
* delegate the key
* set the new admin

Alternative flow:

* in the admin token is invalid -> the system return a "INVALID\_TOKEN" error
* if the admin does not registered -> the system return a "USER\_NOT\_FOUND" error
* if the key invalid -> the system return a "INVALID\_ADMIN\_KEY" error

Acceptance test :

# **Section 7 - special purchase use cases :**

## Use case 1 : add product in auction:

Actor: store owner / manager with permission

Trigger: user want to add a specific product to an auction

Precondition:

* valid token
* the requesting user is registered and not suspended
* the requested user have a permission
* the product exist in the store stock
* the quantity <= the stock quantity

Parameters:

* token
* store id
* product id
* quantity
* time
* start price

Main flow:

* check the validation of the token
* check if the user exist
* check if the user suspended
* check if the store exist
* check if the requested user have a permission
* check the quantity
* create auction
* notify the stores owners
* return auction id

Alternative flow:

* if the token invalid -> the system return " INVALID\_TOKEN" error
* if the requesting user suspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist -> the system return a " STORE\_NOT\_FOUND"
* if the user not admin or manager with a permission -> the system return " NO\_PERMISSION" error
* if the quantity more than the stock quantity -> the system return a " INVALID\_QUANTITY" error

Acceptance test :

## Use case 2 : add product in BID:

Actor: store owner / manager with permission

Trigger: user want to add a specific product to Bid

Precondition:

* valid token
* the requesting user is registered and not suspended
* store exist
* the requested user have a permission
* the product exist in the store stock
* the quantity <= the stock quantity

Parameters:

* token
* store id
* product id
* quantity

Main flow:

* check the validation of the token
* check if the user exist
* check if the user suspended
* check if the store exist
* check if the requested user have a permission
* check quantity
* create bid
* notify the stores owners
* return

Alternative flow:

* if the token invalid -> the system return " INVALID\_TOKEN" error
* if the requesting user suspended -> the system return a " USER\_SUSPENDED" error
* if the store does not exist -> the system return a " STORE\_NOT\_FOUND"
* if the user not admin or manager with a permission -> the system return " NO\_PERMISSION" error
* if the quantity more than the stock quantity -> the system return a " INVALID\_QUANTITY" error

Acceptance test :

## Use case 3 : add product in random:

Actor: store owner / manager with permission

Trigger: the requested user want to add a product to random

Precondition:

* A valid token is provided
* The requesting user is registered and not suspended
* The store existing, active store
* The user holds permission on that store
* The product exists in the store’s stock
* quantity ≤ the available stock quantity
* productPrice ≥ 0
* randomTime > 0

Parameters:

* token
* store id
* product id
* quantity
* price
* random time

Main flow:

* check the validation of the token
* check if the user exist
* check if the user suspended
* check if the store exist
* check if the requested user have a permission
* check quantity
* create random
* reserve stock
* schedule draw completion
* notify the stores owners
* return

Alternative flow:

* if the token is invalid → system returns INVALID\_TOKEN error
* if the requesting user is susp[ended → returns USER\_SUSPENDED
* if the store not exist → returns STORE\_NOT\_FOUND
* if the requested user have no permission → returns NO\_PERMISSION
* if the product doesn’t exist in the store stock → returns PRODUCT\_NOT\_FOUND
* quantity > stock → returns INVALID\_QUANTITY
* price < 0 → returns INVALID\_PRICE
* randomTime ≤ 0 → returns INVALID\_TIME

Acceptance test :

## Use case 4 : join auction :

Actor: subscribed user

Trigger: the user want to join the auction on a product

Precondition:

* valid token
* registered and logged in user
* the user unsuspended
* the store exist
* the product exist in the store
* the auction exist

Parameters:

* token
* store id
* auction id
* price

Main flow:

* check the validation of the token
* check the registration of the user
* check if the user online
* check if the user suspended
* check if the store exist
* in the store check if the auction exist
* add auction
* send notification to the owners
* update the special cart

Alternative flow:

* if the token is invalid -> the system returns a **“INVALID\_TOKEN”** error
* if the user is not logged in -> the system returns a **“USER\_NOT\_LOGGED\_IN”** error
* if the user is suspended -> the system returns a **“USER\_SUSPENDED”** error
* if the requesting user has no permission -> the system returns a **“NO\_PERMISSION”** error
* if the store is not found in the system -> the system returns a **“STORE\_NOT\_FOUND”** error
* if the random draw is not found in the system -> the system returns a **“RANDOM\_NOT\_FOUND”** error
* if the draw has already ended -> the system returns a **“DRAW\_ENDED”** error
* if the payment fails -> the system returns a **“PAYMENT\_ERROR”** error and restores the reserved quantity back to the random draw’s stock

Acceptance test :

## Use case 5 : join bid :

Actor: subscriber user

Trigger: the user add a Bid on a bid auction of product

Precondition:

* valid token
* the user registered in the system
* the user unsuspended
* the store id exist
* the bid exist in the store
* the price >0

Parameters:

* token
* bid id
* store id
* price

Main flow:

* check the validation of the token
* check the registration of the user
* check if the user online
* check if the user suspended
* check if the store exist
* in the store check if the bid exist
* add a bid
* update the special cart
* and send a notification to the store owners

Alternative flow:

* if the token invalid -> the system return a " **INVALID\_TOKEN" error**
* **if the user not logged in -> the system return a " USER\_NOT\_LOGGED\_IN" error**
* **if the user suspended -> the system return a " USER\_SUSPENDED" error**
* **if the stour not exist -> the system return " STORE\_NOT\_FOUND" error**
* **if the store doesn’t have the bid -> the system return a " BID\_NOT\_FOUND" error**
* **if the price invalid -> the system return a " INVALID\_PRICE" error**

Acceptance test :

## Use case 6 : join random:

Actor: subscriber user

Trigger: the user wand to join the random

Precondition:

* the token is valid
* the user registered and not suspended
* the store exist in the system
* the store is active
* the random exist in the store

Parameters:

* token
* store id
* random id
* amount paid

Main flow:

* check the token validation
* cack if the user registered in the system
* check if the user logged in
* check if the user suspended
* search the random
* add the product to a special cart of the user
* update the random details
* process the payment
* save the receipt

Alternative flow:

* if the token invalid -> the system return a " INVALID\_TOKEN" error
* if the user does not logged in-> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system return a " USER\_SUSPENDED" error
* if the requesting user have no permission -> the system return a " NO\_PERMISSION" error
* if the store not found in the system -> the system return a " **STORE\_NOT\_FOUND" error**
* **if the random not found in the system -> the system return a "** **RANDOM\_NOT\_FOUND" error**
* **if the Drow of the random ended -> the system return a "** **DRAW\_ENDED" error**
* **if the payment failed -> the system return a "** **PAYMENT\_ERROR" error and update the amount of the stock ( return the item to the random stock )**

Acceptance test :

## Use case 7 : accept bid:

Actor: owner or manager with permission

Trigger: the requested user want to accept an exit bid

Precondition:

* a valid token
* the requested user in registered
* the requested user unsuspended
* the store exist
* the user have a permission
* the bid exist in the store stock

Parameters:

* token
* store id
* bid id
* the user bid id

Main flow:

* check the token validation
* check if the user registered
* check if the user suspended
* check if the user have permission
* check if the store exist
* check if the bit exist
* accept the bid
* and notify the bidder

Alternative flow:

* if the token invalid -> the system return a " INVALID\_TOKEN" error
* if the user does not logged in-> the system return a " USER\_NOT\_LOGGED\_IN" error
* if the user suspended -> the system return a " USER\_SUSPENDED" error
* if the requesting user have no permission -> the system return a " NO\_PERMISSION" error
* if the store not found in the system -> the system return a " **STORE\_NOT\_FOUND" error**
* **if the bid not found in the system -> the system return a "** **BID\_NOT\_FOUND" error**
* **if the Drow of the random ended -> the system return a "** **DRAW\_ENDED" error**

Acceptance test :

## Use case 9 : view special cart :

Actor : subscribe user

Trigger : the user want to see the special cart

Precondition :

* the token have to be valid
* the user have to be registered to the system
* the user have zero or at least one item in the cart

parameters :

* token

main flow :

* the system received the asked of the user
* check if the validation of the token
* check if the user is registered
* get the user special cart and return

alternative flow :

* if the token is invalid -> the system return a " INVALID\_TOKEN" error
* if the user is not registered to the system -> the system return a " USER\_NOT\_LOGGED\_IN
* " error
* The special cart is empty -> the system return an empty result

Acceptance test :

# **Section 8 – notification use cases :**

## Use case 1 – Send Real-Time Notification :

Actor : system

Trigger : an event occurs, and the system needs to notify the user immediately

Precondition:

* The target user is online
* The msg not null
* The username not null

Parameters:

* Username
* Msg

Main flow:

* Check if the user is online
* Check if the socket is open and the user have at least one socket
* Write the msg in the socket
* Message is delivered over the socket.
* Client UI updates to display the new notification.

Alternative flow:

* If the user is offline → system falls back to “Queue Notification” (Use Case 2).
* If username is null or empty → return error USER\_NOT\_FOUND.
* If message is null or empty → return error INVALID\_MESSAGE\_PARAMETERS.
* If the WebSocket send fails → return error NOTIFICATION\_DELIVERY\_FAILED.

Acceptance test :

## Use case 2 – send delay notification for offline user :

Actor: System

Trigger : An attempt to send a notification to a user who is offline

Precondition :

* The user is offline
* User name is not null or empty
* Msg is not empty or null

parameters :

* message
* user name

main flow :

* check if the user is online ( and its not )
* create delay notefecation record with the user name and the msg
* persist it

alternative flow :

* If username is null or empty → return error USER\_NOT\_FOUND.
* if message is null or empty → return error INVALID\_MESSAGE\_PARAMETERS.
* If saving to the repository fails → return error PERSISTENCE\_ERROR.

Acceptance test :

## Use case 3 – user receive the delay msg

Actor : System

Trigger: user loged in and they have a msg or the user want to see his delayed msg

Preconditions:

* User name not null or empty
* The number of the delay msg have to be more than 0

Parameters:

* User name

Main flow:

* The system search to the notificaations of the user by its user name
* For each msg call the send real time notification
* Return a list of delivered msg

Alternative flow:

* If username is null or empty → return error USER\_NOT\_FOUND.
* If no messages are found → end use case silently.
* If deletion or retrieval from repository fails → return error PERSISTENCE\_ERROR.

Acceptance test:

## Use case 4 - Test-API Notification :

Actor :tester or the admin

Trigger:call the test by the HTTP POST

Preconditions:

* User name valid
* The user exist in the system
* Msg not null or empty

Parameters:

* User name
* Msg

Main flow:

* The system receive HTTP POST
* Check if the user in online and send a real time msg if so , otherwise send delayed msg
* Update the UI in the client side

Alternative flow:

* If username is null, empty, or not found → return "USER\_NOT\_FOUND".
* If message is null or empty → return "INVALID\_MESSAGE\_PARAMETERS".
* If delivery fails in service → return "NOTIFICATION\_DELIVERY\_FAILED".

Acceptance test: