# **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 – Payment System: cancel pay--------8-13

## Use case 7 :API Integration – Supply System: supply---------------8-13

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

## Use case 9 :Retry Logic for External API Failures--------------------8-13

## Use case 10 :Fallback Mechanism for Unavailable Services-------8-13

## Use case 11 :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

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

## Use case

# **Section 6 : admin use cases**

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

# **Section 8 : notification use cases**

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

## Use case 1 :Initialize marketplace system

Actor : system (Spring Boot Runtime)

Trigger :the application is started by the runtime environment

Precondition :

* The application configuration is properly loaded , including : Database connection details ,JWT token settings , Admin credentials
* The database is accessible
* The profile is active
* The context is successfully initialized

Parameters :

* Admin initialization parameters
* Application settings parameters
* JWT configuration (key and expiration time )
* Database configuration

Alternative flow :

* admin already exists – the system catches the exception and skips admin creation without
* app already exist – no action is taken ; system proceeds with startup
* database or configuration error : if the critical failure occurs -> the application startup fails

Acceptance test :

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

Actor :

Trigger :

Precondition :

Parameters :

Main scenario :

Alternative flow :

Acceptance test :

# **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:

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 |

## 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:

|  |  |  |
| --- | --- | --- |
| 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 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
* The user have to be in the database

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 of ReceiptDTO

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:

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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
* take the user repo (authrepo)
* 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

acceptance test : **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 |

## 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

Acceptance test :

|  |  |  |
| --- | --- | --- |
| 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 |

# **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: None

Acceptance test:

|  |  |  |
| --- | --- | --- |
| 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 - 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 :

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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 :

|  |  |  |
| --- | --- | --- |
| 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 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 "DevException" 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 :

|  |  |  |
| --- | --- | --- |
| 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\_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 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 :

|  |  |  |
| --- | --- | --- |
| Success\_RemoveProduct\_from\_cart | 1. Product exists in cart 2. User chooses to remove it | 1. Product removed from cart 2. Cart is updated |

## 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 :

|  |  |  |
| --- | --- | --- |
| 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 |

## 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 :

|  |  |  |
| --- | --- | --- |
| 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 |

# **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 controller call the action of the action from the application layer – store service class
* 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:

|  |  |  |
| --- | --- | --- |
| 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 |

## 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:

|  |  |  |
| --- | --- | --- |
| 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 |

# **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:

|  |  |  |
| --- | --- | --- |
| 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 |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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:

|  |  |  |
| --- | --- | --- |
| 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 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:

## **Section 6 - special purchase use cases :**

## Use case 1 : view the items in the 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 the dto

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 7 – notification use cases :**

Use case 1 :Initialize marketplace system

**Section 1: System Use Cases**

* **Initialize Marketplace System**
* **System Recovery from Configuration File**
* **System Initialization from Initial State File**
* **Failure Handling in Initialization Process**
* **API Integration – Payment System: pay**
* **API Integration – Payment System: cancel\_pay**
* **API Integration – Supply System: supply**
* **API Integration – Supply System: cancel\_supply**
* **Retry Logic for External API Failures**
* **Fallback Mechanism for Unavailable Services**
* **Integrity Rules Enforcement**

**Section 4: Subscribed User Use Cases**

* **Send Message to Store**
* **Submit a Purchase Bid**
* **Buy Product via Auction**
* **Buy Product via Lottery**

**Section 5: Store Owner Use Cases**

* **Add Product to Store**
* **Remove Product from Store**
* **Edit Product in Store**
* **Define Purchase Policy**
* **Delete Purchase Policy**
* **View Store Purchase History**
* **View Manager Authorizations**
* **Appoint Store Manager**
* **Remove Store Manager**
* **Add Store Owner**
* **Remove Store Owner**
* **Deactivate Store**

**Section 7: Notifications and Policy Use Cases**

* **Send Real-Time Notifications**
* **Store Delayed Notifications**
* **Deliver Stored Notifications on Login**
* **Suspend User Activity**
* **Resume Suspended User**
* **View List of Suspended Users**
* **Define Simple Discount**
* **Define Conditional Discount**
* **Define Composite Discount (AND / OR / XOR)**
* **Define Discount Predicates**
* **Edit or Delete Discount Rule**
* **Store owner Reply to Customer Messages**

**Section 8: Payment and Supply Service Use Cases**

* **Process Payment**
* **Cancel Payment**
* **Trigger Supply Delivery**
* **Cancel Supply Delivery**
* **Handle Supply Rejection**
* **Handle Payment Failure**
* **Confirm Delivery Success**