**Vocabulary:**

**Actors:**

**Buyer, Client** - a user that owns a shopping cart and can buy products. A buyer is a guest or member.

**Founder** - a member which opened a new store. A founder is by default a store owner, and therefore has top priority in the store owners hierarchy.

**Guest** - a user that is not logged in (may not have an account).

**Logged In Member** - user that is logged in to an existing member’s account.

**Member** - A user that passed a registration process to the system.

**Store Manager** - A member that was appointed by the store owner to be a store manager and has management permissions in the said store.

**Store Owner** - A member that has opened a store or was appointed as a store owner by another store owner and has ownership permissions in the said store.

**System Manager, Admin** - a member with system administration privileges.

**User** - a person that uses the system.

**Visitor** - A user that enters the market. In our market every visitor is a guest or logged in member.

**Accounts & Authentication:**

**Active System** - system is on, connected to the internet, can serve users.

**Authentication Details** - A pair of username and password that follows the authentication details rules.

**Authentication Details Rules** - Constraints regarding the structure, length and permitted characters in the username and password.

**Login -** is done by the unique identifiers that were used in the registration process, at the end of the process the guest is identified by the system as a member.

**Member’s account** - an account for users registered to the system.

**Message** - Some members can communicate with others in the system, by sending messages to them.

**Password** - each member account holds a password that along with the member’s username authenticates the member’s account. mainly used in the login process.

**Registration** - the process in which a guest could create himself a member account. The process is done by supplying legal and unique identifiers: username (which isn’t already taken) and a password. The process only creates the member account for the guest. in order to be a member in the system the guest must login.

**Username** - each member account holds a name that identifies the member in the system, the name is necessarily unique.

**Market properties:**

**Active Store** - a store that has at least one store owner. An active store must show its relevant products in the general products search.

**Bag** - describes the products and amounts to buy from a certain store. a bug is contained in a cart of a buyer.

**Buying methods, Purchase options** - set of predefined properties that represents the different options to purchase products of a certain store. different products can have different purchase options, as long as they’re allowed by the store’s purchase policy. (example: an auction).

**Buying/Purchase Policy** - A contract of a store that defines: who can purchase products of the store, possible purchase options and their rules.

**Cart** - A set of store bags, maximum single bag for each store. Owned by a buyer and is used to perform its buyings in the market.

**Discount** - a reduction of the price of a product, set of products or the whole cart in the purchase process of a buyer (options for discounts: hidden, shown, conditional).

**Discount Policy** - A contract of a store that defines: who is entitled for a discount, the different types of discounts , and different rules. the contract could refer to products, buyers or both.

**Filter Properties**  - set predefined properties by which a buyer can filter the results of the products search (e.g price ranges, product/store ratings, category).

**Market** - the whole stores management system. note that the market isn’t responsible for other services that the system provides such as the external services management.

**Purchase** - an attempt of the buyer to buy his cart content. a purchase is composed of the acquisition of each store bag in the buyer’s cart. A purchase is done according to the possible purchase and discount policies (of each store respectively), and by availability of the relevant products in the inventory.

**Product** - a merchandise that a certain active store can offer for purchase. A product can hold it’s own discount policy regardless of the discount policies that the store holds.

**Product Details** - set of predefined properties to choose from (e.g. size, color).

**Search Properties** - set predefined properties by which a buyer could search for products in a variety of active stores (e.g product name, category, keywords).

**Store** - An online shop in the system, that contains: products for sale, discounts and buying policies. Each shop has a founder, and possibly several managers and store owners depending on it’s state(active/inactive).

**Store Bag** - set of products together with product selection of details and selected count.

**Store Information** - name, products in inventory, store founder, store owners, store managers, etc.

**Store Policy** - specifies the permitted buying methods and discounts in a certain store.

**External services:**

**Deal** - products that a buyer wants to buy and the payment means he provides.

**Delivery** - package and customer details

**External Services** - services which are external to the system, and are being used by the system. These are the payment and supply services.

**Payment Service** - an external payment charge system that offers the services of executing a payment transaction between buyers and stores.

**Supply Service** - an external stock delivery system, that offers the services of package shipment between stores and buyers.

**Valid Deal -** a deal in which the payment means of the buyer are valid, in-date and contain enough funds to pay for the chosen products, Only valid deals can be approved by a payment system.

**Valid Delivery** - a delivery in which the package’s details match the customer’s address details. Only valid deliveries can be approved by a supply system.

**System architecture:**

**AdminManager** - a class responsible for holding and authenticating the admins of the system, and performing admin actions.

**Communication** - the architecture layer that is responsible for using the service and communicating with the clients.

**Controller** - an ExampleController class is responsible for managing the Example objects in the system, providing an interface for getting them, adding new Example objects etc.

**Domain** - the architecture layer responsible for the system’s logic, using also the Data access layer.

**Data Access** - the architecture layer that is responsible for the persistent data in the system, by communicating with the storage.

**Facade** - a class in the service layer that supplies an interface for a certain set of actions in the domain layer, that are semantically related to each other.

**Hierarchy** - a general object that represents an hierarchical structure over a set of objects. Used in the system to save and use the appointments hierarchy of store owners and managers.

**Permission** - a set of options, where each option represents a certain permit that a member could have in a certain store.

**Purchase**  - the process of a certain buyer buying products.

**Purchase History** - the saved history of purchases in the system, saved for every store and member (maybe for member only in the future).

**PurchasesManager** - a class responsible for managing purchases (see **Purchase**).

**Response<T>** - a class which is always the return value of the service layer. The service layer returns such an object, configured either to contain an error message on the action performed, or to contain the data to be returned if the action succeeds.

**Role** - a set of options, where each option represents a certain character that a member could have. In some cases, performing as a certain role could derive some specific permissions with it.

**Service** - the architecture layer responsible to create a comfortable interface to use the Domain layer.

**Storage** - represents the space in which the persistent data is saved in the architecture diagram.

**Store Management** - the set of actions to manage a store in the system, used mainly by store owners and managers.

**SystemOperater** - a class that is responsible for opening the market system, according to **Use Case I.1 - Market activation** , creating all the facades and controllers.

**System Controller** - a class responsible for managing the external services and the connection with them, by managing interfaces that provide the domain access to them.

**Store Owner/Manager appointment** - Making a member an owner or manager of a certain store.

**ServiceDTO** - an object in the service layer that holds data and has no additional functionality, that is used to return complex data from the service layer to its clients.

**Rules Of Integrity**:   
1. System can charge money from buyers for purchases they made, and only the amount agreed.   
2. The purchase process ends successfully when payment is done and supply is approved.  
3. A seller can get payment after a successful purchase process.

**System use cases:**

**Use Case I.1 - Market activation:**

A user activates the market system, and becomes the first system manager.

Actors: user (will be set to be system manager) - see vocabulary.

Precondition: system must be off.

Parameters: initial authentication details (username and password) and details about one payment service and one supply service (more can be added later).

Postcondition: system must be on, and adheres to all the rules of integrity.

Main Scenario:

1. user: runs the system.
2. system: asks the user for initial authentication details.
3. user: enters its (new) authentication details.
4. system: verifies the validity of the given details.
5. system: opens the system, with one member and system manager that both represent the user, with the given external services and keeping all the integrity rules.
6. system: responses a success to the user.
7. system: start serving users.

Alternative Scenarios:

* User’s initial authentication details not verified successfully (4), system asks for the details again.
* Payment service or supply service are not verified successfully (4), the system asks for the details again.
* System fails preparation (5), the system cancels the action and waits for System Manager action.

**Use Case I.2 - change/switch/add an external service:**

A user changes/switches/adds an external service

Actors: user.

Precondition: system must be on.

Parameters: details of the new service (payment or stock)

Postcondition: the system must continue to work the same as before the change of the service

Main Scenario:

1. user: chooses the service he wants to edit/add/switch
2. user: enters the details of the service
3. system: authenticates the details the user entered
4. system: changes/add/switches the service for the new one
5. system : send positive response to the user

Alternative Scenarios:

* User’s initial authentication details not verified successfully (4), system asks for the details again.
* service details are incorrect and requested to enter again.

**Use Case I.3 : Using the services of an external payment system**

the system contacts a payment system which the market is familiar with and tries to perform a payment and receive payment validation

Actors: System

Precondition: the system is activated

Postcondition: the system must continue to work the same as before the change of the service

Main Scenario:

1. the system contacts the external payment system with transaction details to perform payment
2. the external payment sends back a positive validation for the transaction the system tried to perform

Alternative Scenarios:

* the transaction details the system sent are invalid
* the payment service returned a negative confirmation for the payment the system tried to do because of problem with the transaction (not because of the details)

**Use Case I.4 - Using the services of an external supply service:**

The system contacts a supply system which the market is familiar with and asks for approval of a valid supply.

Actors: System

Precondition: System is activated

Parameters: details of the supply

Postcondition: the system must continue to work the same as before the request of service

Main Scenario:

1. system: contacts an external supply system it is familiar with and sending it the details of the required package and client information
2. external service: sends back a positive answer that the request was approved

Alternative Scenarios:

* The external service sends back a negative answer (2) that the request was declined, and the reason for the decline of the request.

**Use Case I.5 :**

**(a)**

Actors: A store owner.

Precondition: System must be active, and the store owner is logged in.

Main Scenario:

1. One of the following occurs:

* A product in one of the store owner’s stores is bought.
* One of the store owner’s stores is being opened.
* One of the store owner’s stores is being closed.
* One of the store appointments as a store owner is being removed.

1. The system notifies the store owner about the event that occured.

**(b)**

Actors: A member.

Precondition: System must be active, and the member is logged in.

Main Scenario:

1. The member receives a message.
2. The system notifies the store owner about the event that occured.

**Use Case I.6 - User notifications show up when logging in:**

A member logs in to the system, and the notifications that were meant for them during the time they were logged off are shown.

Actors:. Member

Precondition: System must be active, Member must be logged in.

Postcondition: System must continue to work normally as it should after any login, without considering the notifications.

Main Scenario:

1. member: logs in to the system.
2. system: shows up all notifications of the member to them.

Alternative Scenarios:

* No new notifications to show (2) so the system doesn’t show anything to the member.

**Guest user use cases:**

**Use Case II.1.1 - Guest entrance:**

A user enters the system as a Guest and gets a cart.

Actors: user.

Precondition: System must be active.

Parameters: None

Main Scenario:

1. user: send an entrance request.
2. system: accept the user (start handling its requests).
3. system: set the user's state as Guest.
4. system: create cart associated with the user.
5. system: send a successful response to the user.

Alternative Scenarios:

* The user is disconnecting during 3-5, the system cancels the action and closes the user’s connection.

**Use Case II.1.2 - Guest disconnection:**

A Guest can disconnect from the system.

Actors: Guest.

Precondition: System must be active.

Parameters: None

Main Scenario:

1. guest: send a disconnection request.
2. system: delete guest’s cart.
3. system: disconnects the guest and closes the connection.

Alternative Scenarios:

* The guest closes the connection (1), the system acts as a disconnection request has been received.

**Use Case II.1.3 - Guest registration:**

A Guest can sign up to the system.

Actors: Guest.

Precondition: System must be active.

Parameters: authentication details

Postcondition: A member account with given authentication details exists.

Main Scenario:

1. guest: send a sign up request.
2. system: verify authentication details using authentication details rules.
3. system: insert new member with the given details.
4. system: sends a successful response to the guest (state left unchanged).

Alternative Scenarios:

* Guest’s authentication details not verified successfully (2), system cancels action and sends error response.

**Use Case II.1.4 - Guest login:**

A Guest can log in to the system.

Actors: Guest.

Precondition: System must be active.

Parameters: authentication details

Postcondition: The guest state has changed to a logged in member.

Main Scenario:

1. guest: send a login request.
2. system: verify authentication details using the existing system’s members.
3. system: change guest state to logged in member.
4. system: sends a successful response to the guest.

Alternative Scenarios:

* user’s authentication details not verified successfully (2), system cancels action and sends error response.

**Use Case II.2.1 - Info request:**

A Guest can request information about the stores and products in stores.

**Use Case II.2.1.a - Get stores info:**

A Guest can request information about the stores in the system.

Actors: Guest.

Precondition: System must be active.

Parameters: None

Result: List of the stores currently exist and active in the system

Main Scenario:

1. guest: send a store info request.
2. system: filter the currently active stores in the system.
3. system: sends a successful response to the user containing a list of the stores.

**Use Case II.2.1.b - Get products info:**

A Guest can request information about the products in an active store of the system.

Actors: Guest.

Precondition: System must be active.

Parameters: store id

Result: List of the products currently exist in the store

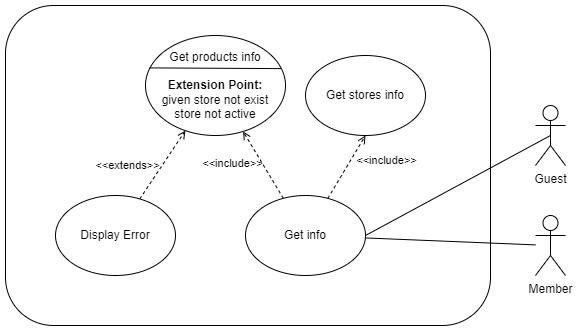
Main Scenario:

1. guest: send a product info request.
2. system: verify that the store exists and is active.
3. system: creates a list of products in the store.
4. system: sends a successful response to the user containing a list of products.

Alternative Scenarios:

* store does not exist in the system (2), cancels action and sends error response.
* store is not currently active (2), cancels action and sends error response.

**Use Case II.2.1 - diagram**



**Use Case II.2.2 - Search products:**

A Guest can search products that exist in stores, using search properties and filter properties.

Actors: Guest.

Precondition: System must be active.

Parameters: search properties (mandatory) and filter properties (optional)

Result: List of the products found.

Main Scenario:

1. guest: send a product search request.
2. system: filter the products that match the given search and filter properties.
3. system: sends a successful response to the user containing a list of products.

**Use Case II.2.3 - Adding products:**

A Guest can add products to his store bag, stored in the personal cart.

Actors: Guest.

Precondition: System must be active.

Parameters: product id, count, product details (optional)

Main Scenario:

1. guest: send an add product request.
2. system: add the product to the store bag of the guest, by the given count and store it in the personal cart.
3. system: sends a successful response to the user.

**Use Case II.2.4 - Manage cart:**

A Guest can view and edit products in his personal cart.

**Use Case II.2.4.a - View cart:**

A Guest can view products in his personal cart.

Actors: Guest.

Precondition: System must be active.

Parameters: None

Result:personal cart details.

Main Scenario:

1. guest: send a view cart request.
2. system: sends a successful response to the user with the cart details.

**Use Case II.2.4.b - Delete product from cart:**

A Guest can delete a product from his personal cart.

Actors: Guest.

Precondition: System must be active, cart must contain the product.

Parameters: product id

Result:personal cart details.

Main Scenario:

1. guest: send a delete product request.
2. system: deletes the product from the guest’s personal cart.
3. system: sends a successful response to the user with the cart details.

**Use Case II.2.4.c - Change product count in cart:**

A Guest can edit the product's count in his personal cart.

Actors: Guest.

Precondition: System must be active, cart must contain the product.

Parameters: product id, new count.

Result:personal cart details.

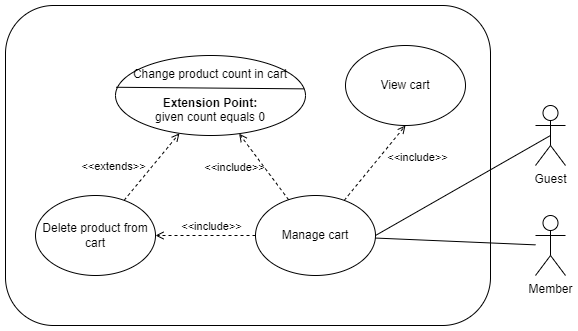
Main Scenario:

1. guest: send an edit product count request.
2. system: changes the product count to the given count in the guest's personal cart.
3. system: sends a successful response to the user with the cart details.

Alternative Scenarios:

* user sends count=0 in the request (1), system continues from step (2) in Use Case II.2.4.b.

**Use Case II.2.1 - diagram**



**Use Case II.2.5 - Purchase cart content:**

A Guest can purchase his cart content according to the buy and discount policy for guests, and according to the product's availability in store.

**Use Case II.2.5.a - Purchase cart content (available products):**

A Guest can purchase his personal cart content if all the products are available.

Actors: Guest.

Precondition: System must be active, non-empty cart.

Parameters: payment method, payment details, delivery details

Result:personal cart details.

Main Scenario:

1. guest: send a purchase request.
2. system: verify that all products are fully available in stores.
3. system: verify payment method and details with the payment company
4. system: verify delivery details with the delivery company.
5. system: make the purchase.
6. system: insert new order details to the system.
7. system: sends order details to the delivery company.
8. system: cleans the user’s cart
9. system: sends a successful response to the user with the order invoice and empty cart.

Alternative Scenarios:

* payment verification failed (3), the system should cancel the action and send an error response.
* delivery verification failed (4), the system should cancel the action and send an error response.

**Use Case II.2.5.b - Purchase cart content (products not available):**

A Guest can’t purchase his personal cart content if not all the products are available.

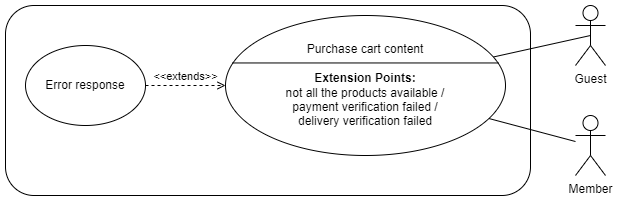
Actors: Guest.

Precondition: System must be active, non-empty cart.

Parameters: payment method, payment details, delivery details

Main Scenario:

1. guest: send a purchase request.
2. system: verify that all products are fully available in stores.
3. system: sends an error response telling which products are not available

**Use Case II.2.5 - diagram**

**Member user use cases:**

Members are able to do use cases **II.2.1-II.2.5** exactly the same way as guests.   
New precondition is that the user must be logged in.

**Use Case II.3.0(1.2) - Member disconnection:**

A Member can disconnect from the system.

Actors: Logged in member.

Precondition: System must be active, user must be logged in.

Parameters: None

Main Scenario:

1. member: send a disconnection request.
2. system: saves the member’s cart.
3. system: disconnects the member and closes the connection.

Alternative Scenarios:

* The member closes the connection (1), the system acts as a disconnection request has been received.

**Use Case II.3.1 - Member logout:**

A Member can logout from the system.

Actors: Logged in member.

Precondition: System must be active, user must be logged in.

Parameters: None

Main Scenario:

1. member: send a logout request.
2. system: saves the member’s cart.
3. system: changes user state to guest.
4. system: sends a successful response with a new empty cart.

**Use Case II.3.2 - Create new store:**

A Member can create a new store and become its founder.

Actors: Logged in member.

Precondition: System must be active, user must be logged in.

Parameters: store information

Main Scenario:

1. member: sends request for creating new store.
2. system: creates a new store with given details.
3. system: sets the member to be the founder of the store.
4. system: sends a successful response with store id.

**Store owner use cases:**

**Use Case II.4.1 - Inventory management:**

An owner can manage the store inventory (i.e. adding and remove products, changing the products’ details)

**Use Case II.4.1.a - Adding new product to the store:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in.

Parameters: new product details, amount in inventory.

Main Scenario:

1. owner: sends request for adding new product to the store.
2. system: creates new product collection using the given details.
3. system: sends a successful response.

**Use Case II.4.1.b - Removing products from the store:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in, product id must be of an active product in store.

Parameters: product id.

Main Scenario:

1. owner: sends request for removing existing product from the store.
2. system: remove product from the store.
3. system: sends a successful response.

**Use Case II.4.1.c - Changing product’s quantity in inventory:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in, product id must be of an active product in the store.

Parameters: product id, new quantity.

Main Scenario:

1. owner: sends request for updating product’s quantity in the store.
2. system: update product’s quantity in store inventory.
3. system: sends a successful response.

**Use Case II.4.1.d - Changing product’s details in the store:**

Actors: Store owner.

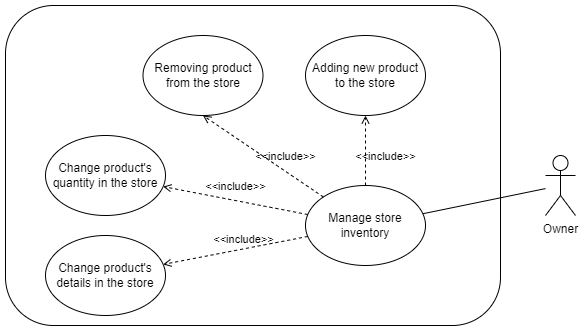
Precondition: System must be active, store must be active, store owner must be logged in, product id must be of an active product in the store.

Parameters: product id, new product’s details.

Main Scenario:

1. owner: sends request for updating product’s details in the store.
2. system: update product’s details in store.
3. system: sends a successful response.

**Use Case II.4.1 - diagram**

****

**Use Case II.4.2 - Change type of purchases and discount policy:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in.

Parameters: store id, purchases and discount policy.

Postcondition: New policy should apply

Main Scenario:

1. owner: sends request for changing policy in the store.
2. system: sets new policy for the store.
3. system: sends a successful response.

**Use Case II.4.4 - Make store co-owner:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in.

Parameters: member id.

Postcondition: Store owners hierarchy is a tree (no cycles).

Main Scenario:

1. owner: sends request for making new owner in the store.
2. system: verify member exists and not already owner.
3. system: sets new owner under the actor’s (owner) in the hierarchy.
4. system: sends a successful response.

Alternative Scenarios:

* The member does not exist (2), the system cancels the action.
* The member is already an owner of the store (2), the system cancels the action.

**Use Case II.4.6 - Make store manager:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in.

Parameters: member id.

Postcondition: The new manager permissions are getting info (4.12, 4.13)

Main Scenario:

1. owner: sends request for making new manager in the store.
2. system: verify member exists and not already manager.
3. system: sets the new manager with the owner as appointor.
4. system: sends a successful response.

Alternative Scenarios:

* The member does not exist (2), the system cancels the action.
* The member is already a manager of the store (2), the system cancels the action.

**Use Case II.4.7 - Change store manager permissions:**

Actors: Store owner.

Precondition: System must be active, store must be active, store owner must be logged in, permissions before action restrict the member’s actions..

Parameters: member id of manager, new permissions.

Postcondition: The manager’s new permissions updated and restrictions have been applied.

Main Scenario:

1. owner: sends request for making new permissions for manager of the store.
2. system: verify member exists and is manager.
3. system: sets the new manager permissions.
4. system: sends a successful response.

Alternative Scenarios:

* The member is not a manager of the store (2), the system cancels the action.

**Use Case II.4.9 - Make store inactive:**

Actors: Founder.

Precondition: System must be active, store must be active, founder must be logged in, the store’s managers and owners exist in the system.

Parameters: member id of founder, store id of the store to be closed.

Postcondition: The store owners and managers have been notified properly and the closed store products won’t show it’s products in the product search.

Main Scenario:

1. founder: sends request for closing one of his stores.
2. system: verify that the member exists and is a founder of the store.
3. system: verify that the store is indeed currently opened.
4. system: mark the store as closed.
5. system: sends a notification message to the store owners and managers.
6. system: sends a successful response.

Alternative Scenarios:

* The member is not the founder of the store (2), the system cancels the action.
* The store is already closed (3), the system cancels the action.

**Use Case II.4.11 - Get store roles information:**

Store owner can get information about the roles in store, and managers permissions.

**Use Case II.4.11.a - Get store roles list:**

Actors: Store owner.

Precondition: System must be active, store owner must be logged in.

Parameters: member id of the store owner, store id of the relevant store.

Main Scenario:

1. store owner: sends a request for information about the store’s roles.
2. system: verify that the member exists and is a store owner of the store.
3. system: sends a successful response containing the store’s members id’s with their roles.

Alternative Scenarios:

* The member is not a store owner of the store (2), the system cancels the action.

**Use Case II.4.11.b - Get store’s managers permissions:**

Actors: Store owner .

Precondition: System must be active, store owner must be logged in.

Parameters: member id of the store owner, store id of the relevant store.

Main Scenario:

1. store owner: sends a request for information about the store manager's permissions.
2. system: verify that the member exists and is a store owner of the store.
3. system: sends a successful response containing the store manager’s id along with their permissions.

Alternative Scenarios:

* The member is not a store owner of the store (2), the system cancels the action.

**Use Case II.4.13 - Get purchase history in store by store owner:**

Actors: Store owner .

Precondition: System must be active, store owner must be logged in.

Parameters: member id of store owner, store id of the relevant store.

Main Scenario:

1. store owner: sends a request for a purchase history of a certain store. .
2. system: verify that the member exists and is a store owner of the store.
3. system: sends a successful response containing the history of the store purchases history.

Alternative Scenarios:

* The member is not a store owner of the store (2), the system cancels the action.

**Use Case 6.4 - Get the purchase history of a store/buyer by a system manager (Admin):**

Actors: System manager

Precondition: System must be active, system manager must be logged in.

Parameters: Admin id, store/buyer id.

Main Scenario:

1. The admin requests the purchase history of the store/buyer and inputs the required credentials(admin id) and the information regarding the store/buyer(store/buyer id).
2. The system verifies that the admin of the given id exists in the system, and that the store/buyer id corresponds to an existing store/buyer in the system.
3. The system outputs the purchase history of the entered store/buyer.

Alternative Scenarios:

* The admin id entered by the system manager is for an admin not currently existing in the system so the action can’t be performed so an error message is returned via a notification.
* The store/buyer id does not exist in the system so the action can’t be performed so an error message is returned via a notification.