Store Model Service Design Document

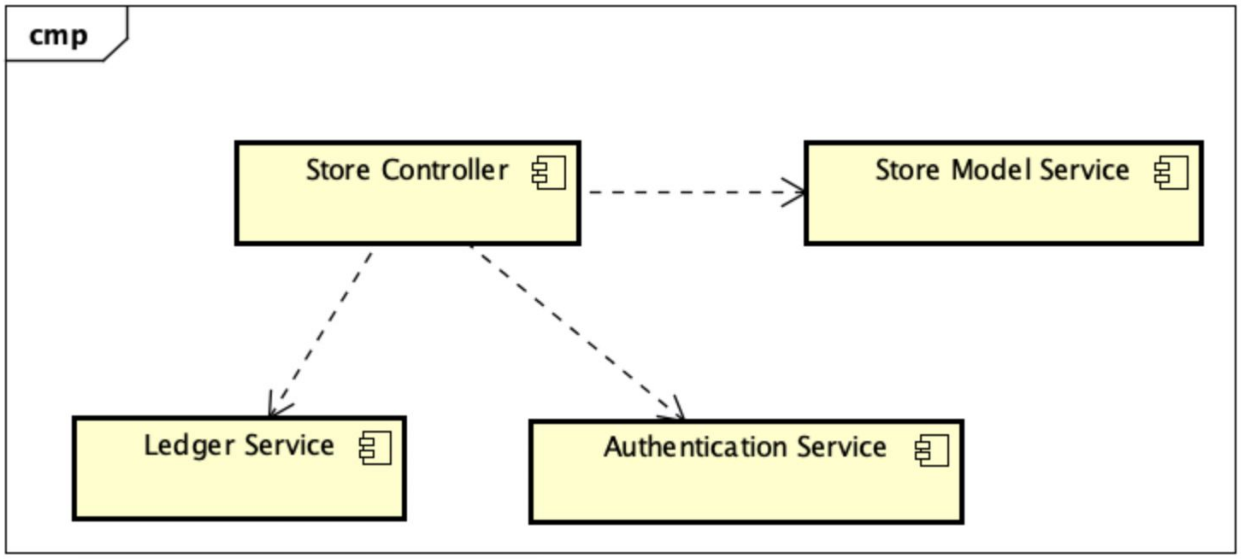
Author: Santiago Iraola
Date: 11/01/2019

Introduction

This document defines the design of the Store Model Service.

Overview

The current service to be designed is part of a bigger system, the Store24X7 which allows physical stores to leverage the power of AI, cameras and sensors to create a fully automated experience for the customer. The implementation of such system will overall increase customer experience and reduce costs.
As mentioned above the Store Model Service (referred to SMS) responsibility is to manage the state of all the elements of the store domain, including the sensors and appliances.



The Store24x7 system will have four main modules. The Store Model Service will primarily be consumed by a Store Controller service that will detect events in the store and request the SMS to update state of the domain objects.

Requirements

The SMS needs to handle the state of all domain objects of the 24x7 store in a safe and efficient way. The **domain objects** needed for this design are the following:

- **Store**: physical store where the customer would interact
- **Customer**: a user of our store
- **Basket**: cart that holds items a customer currently plans to buy
- **Product**: representation of the products and their information
- **Aisle**: physical space in the store
- **Shelf**: physical space in the store where items will be placed
- **Inventory**: products currently in store
- **Sensor**: representation of devices such as cameras or microphones that will be recording data.
- **Appliance**: devices that will not only record data but perform different tasks

The SMS API needs to be able to handle the following:

Define any of the above mentioned objects
SMS can receive requests via the defined API and create new instances of these objects.

Show the state of objects
SMS can show the current state of a given object.

Modify state of objects
Some objects such as Customers, Baskets or Inventory will need to update state when events are read by the sensors. The SMS system will be able to make changes accordingly.

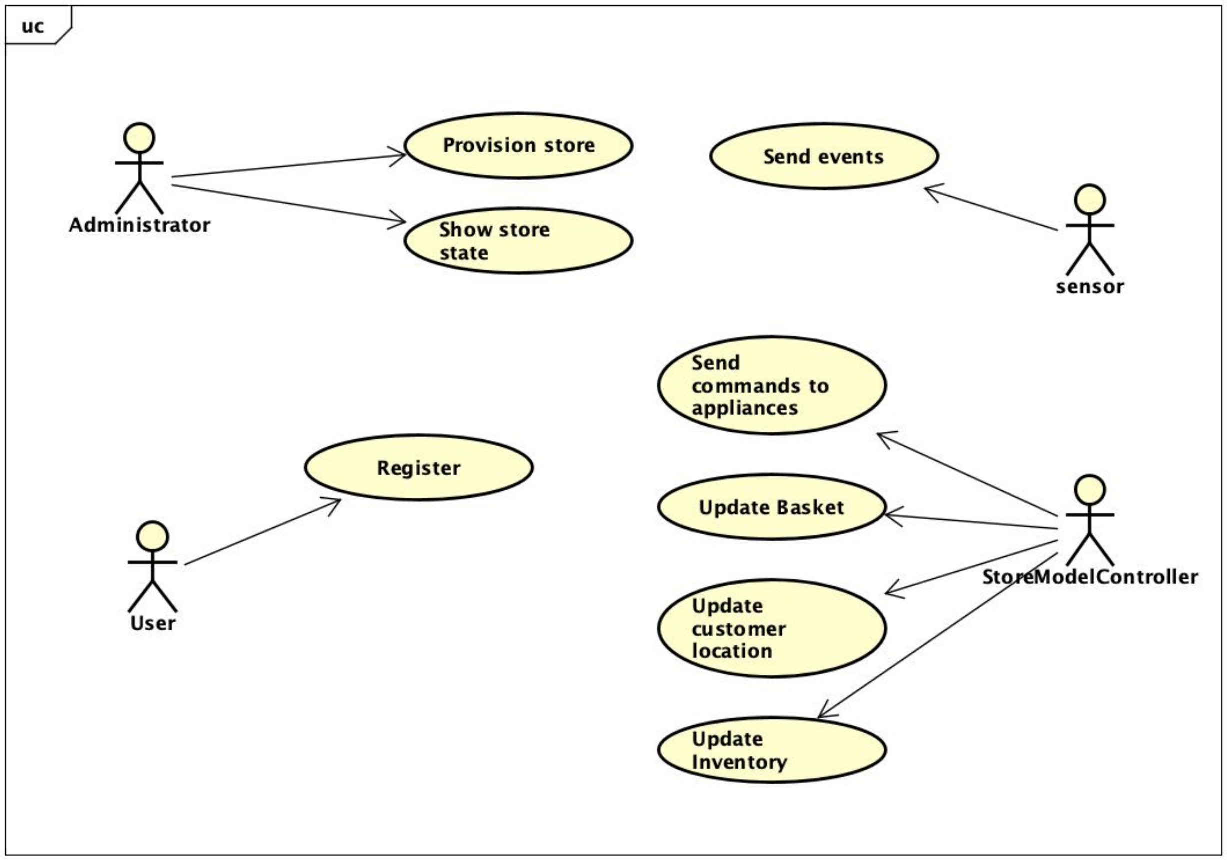
Create Sensor or Appliance events
Will stimulate sensors or appliances via events.

Not required

Persistence
All of the data will be maintained in memory, there is no requirement to use any sort of permanent database.

REST API
The system will live in its entirety in one JVM, there is no need to implement or design a RESTful API in order for other modules to interact with the SMS.

Use Cases
The following Use Case diagram describes the use cases supported by the Store Model Service.



Actors

Administrator
Administrator is in charge of provisioning the store. Will define all of the domain objects in the store (for the full list please see the requirements or class diagram). It will also be able to query the state of any of the objects by their unique identifier.

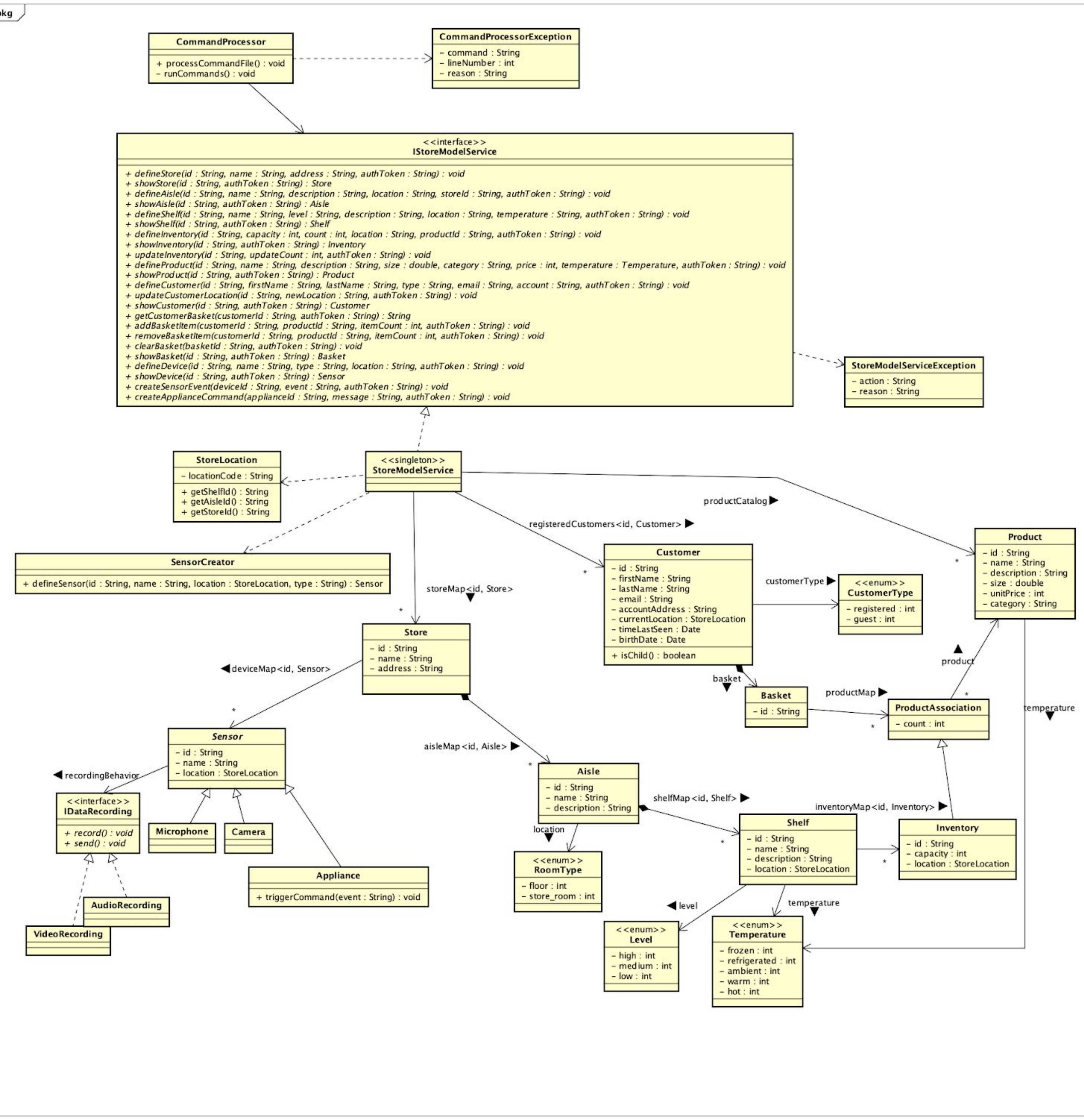
User
User represents the customers within the store and will only interact with the SMS when registering themselves and their Ledger Account.

Sensor
Sensors also include appliances that are present within the store and will record and send events to the SMS.

Store Model Controller
Store Model Controller is the main user for this service. It will send commands to appliances such as robots, turnstiles or speakers for them to perform actions. It will also notify the SMS when the state of any of its objects need to be updated such as the inventory or the basket.

Implementation

Class Diagram


pkg
CommandProcessor
+ processCommandFile() : void
- runCommands() : void
CommandProcessorException
- command : String
- lineNumber : int
- reason : String
<<interface>>
IStoreModelService
+ defineStore(id : String, name : String, address : String, authToken : String) : void
+ showStore(id : String, authToken : String) : Store
+ defineAisle(id : String, name : String, description : String, location : String, storeId : String, authToken : String) : void
+ showAisle(id : String, authToken : String) : Aisle
+ defineShelf(id : String, name : String, level : String, description : String, location : String, temperature : String, authToken : String) : void
+ showShelf(id : String, authToken : String) : Shelf
+ defineInventory(id : String, capacity : int, count : int, location : String, productId : String, authToken : String) : void
+ showInventory(id : String, authToken : String) : Inventory
+ updateInventory(id : String, updateCount : int, authToken : String) : void
+ defineProduct(id : String, name : String, description : String, size : double, category : String, price : int, temperature : Temperature, authToken : String) : void
+ showProduct(id : String, authToken : String) : Product
+ defineCustomer(id : String, firstName : String, lastName : String, type : String, email : String, account : String, authToken : String) : void
+ updateCustomerLocation(id : String, newLocation : String, authToken : String) : void
+ showCustomer(id : String, authToken : String) : Customer
+ getCustomerBasket(customerId : String, authToken : String) : String
+ addBasketItem(customerId : String, productId : String, itemCount : int, authToken : String) : void
+ removeBasketItem(customerId : String, productId : String, itemCount : int, authToken : String) : void
+ clearBasket(basketId : String, authToken : String) : void
+ showBasket(id : String, authToken : String) : Basket
+ defineDevice(id : String, name : String, type : String, location : String, authToken : String) : void
+ showDevice(id : String, authToken : String) : Sensor
+ createSensorEvent(deviceId : String, event : String, authToken : String) : void
+ createApplianceCommand(applianceId : String, message : String, authToken : String) : void
StoreModelServiceException
- action : String
- reason : String
StoreLocation
- locationCode : String
+ getShelfId() : String
+ getAisleId() : String
+ getStoreId() : String
<<singleton>>
StoreModelService
productCatalog ▶
registeredCustomers<id, Customer> ▶
SensorCreator
+ defineSensor(id : String, name : String, location : StoreLocation, type : String) : Sensor
storeMap<id, Store> ▼
Customer
- id : String
- firstName : String
- lastName : String
- email : String
- accountAddress : String
- currentLocation : StoreLocation
- timeLastSeen : Date
- birthDate : Date
+ isChild() : boolean
customerType ▶
<<enum>>
CustomerType
- registered : int
- guest : int
Product
- id : String
- name : String
- description : String
- size : double
- unitPrice : int
- category : String
Store
- id : String
- name : String
- address : String
◀ deviceMap<id, Sensor>
Sensor
- id : String
- name : String
- location : StoreLocation
◀ recordingBehavior
<<interface>>
IDataRecording
+ record() : void
+ send() : void
Microphone
Camera
Appliance
+ triggerCommand(event : String) : void
AudioRecording
VideoRecording
basket ▼
Basket
- id : String
productMap ▶
▲ product
ProductAssociation
- count : int
temperature ▼
aisleMap<id, Aisle> ▶
Aisle
- id : String
- name : String
- description : String
location ▼
<<enum>>
RoomType
- floor : int
- store_room : int
shelfMap<id, Shelf> ▶
Shelf
- id : String
- name : String
- description : String
- location : StoreLocation
inventoryMap<id, Inventory> ▶
Inventory
- id : String
- capacity : int
- location : StoreLocation
◀ level
<<enum>>
Level
- high : int
- medium : int
- low : int
temperature ▼
<<enum>>
Temperature
- frozen : int
- refrigerated : int
- ambient : int
- warm : int
- hot : int
*

Class Dictionary

StoreModelService
The SMS will implement the IStoreModelService that defines all of the operations it needs to handle. The implementation of this object should be singleton.

Associations

Association Name	Type	Description
productCatalog	Map<productId, Product>	All products defined within the SMS
storeMap	Map<storeId, Store>	All of the store instances managed
registeredCustomers	Map<customerId, Customer>	All customers registered by the SMS

Methods

All define methods that do not have a unique id or a valid location should throw StoreModelServiceException if id is not unique. Every method in the interface should be able to take an authToken as a String parameter. This will be used for validating access to certain operations defined later on the next design iteration.

Method Name	Signature	Description
showDevice	(id: String, authToken: String): Sensor	return device based on id or null if not present
showStore	(id: String, authToken: String): Store	return store based on id or null if not present
showAisle	(id: String, authToken: String): Aisle	return aisle based on id or null if not present
showShelf	(id: String, authToken: String): Shelf	return shelf based on id or null if not present

showInventory	(id: String, authToken: String): Inventory	return inventory based on id or null if not present
showProduct	(id: String, authToken: String): Product	return product based on id or null if not present
showCustomer	(id: String, authToken: String): Customer	return customer based on id or null if not present
showBasket	(id: String, authToken: String): Basket	return Basket based on id or null if not present
defineStore	(id: String, name: String, address: String, authToken: String): void	Creates a new store and adds it to the storeMap.
defineAisle	(id: String, name: String, description: String, location: String, authToken: String): void	Creates a new aisle and assigns it to a given store based on the location
defineShelf	(id: String, name: String, description: String, location: String, authToken: String): void	Creates a new shelf and assigns it to a given store and aisle based on the location
defineInventory	(id: String, capacity:Int,count: int, location: String, productId: String, authToken: String): void	Creates a new inventory item in a given store:aisle:shelf. That inventory has associated a product based on productId
defineProduct	(id: String, name: String, description: String, size: double, category: String, price: int, temperature: Temperature, authToken: String): void	Creates a new Product and add it into the productCatalog
defineCustomer	(id: String, name: String, lastName: String, type: CustomerType, email: String, account: String, authToken: String): void	Creates a new customer and adds it into the registeredCustomers list
defineDevice	(id: String, name:	Creates a new device and

	String,type: String, location: String, authToken: String): void	assigns it to the appropriate location
updateInventory	(id: String, updateCount: int, authToken: String): void	Update the given inventory the amount stated in updateCount. It can take positive numbers for addition or negative for subtraction. Throw StoreModelServiceException if id is not found
updateCustomerLocation	(id: String, newLocation: String, authToken: String): void	Update given customer location to a new one. Throw StoreModelServiceException if id is not found or newLocation is invalid
getCustomerBasket	(customerId: String, authToken: String): String	Get basket id associated with the customer, create a new basket if the customer does not have one. Throw StoreModelServiceException if id is not found or customer is a guest
addBasketItem	(customerId: String, productId:String, itemCount: int, authToken: String): void	Add itemCount number of new products with productId. Throw StoreModelServiceException if id is not found or productId is invalid or if customer is a guest or basket not initialized.
removeBasketItem	(customerId: String, productId:String, itemCount: int, authToken: String): void	Remove itemCount number of products from basket. If itemCount exceeds basket count remove item from basketMap. Throw StoreModelServiceException if id is not found or basket is not present in customer.

clearBasket	(customerId: String, authToken: String): void	Empties the basketMap of a given customer.

Store
Store represents one particular physical store within the SMS.

Properties

Property Name	Type	Description
id	String	Unique identifier
name	String	Store name
address	String	Store address

Associations

Association Name	Type	Description
aisleMap	Map<aisleId, Aisle>	All the aisles within the store
deviceMap	Map<deviceId, Device>	All devices inside the store

Aisle
Aisle represents sections of a particular physical store within the SMS.

Properties

Property Name	Type	Description
id	String	Unique identifier
name	String	Aisle name
description	String	Description of the aisle

Associations

Association Name	Type	Description
location	RoomType	Section of the store
shelfMap	Map<shelfId,Shelf>	Collection of shelves within the aisle.

Shelf
Shelf represents sections of a particular aisle where inventory is placed.

Properties

Property Name	Type	Description
id	String	Unique identifier
name	String	Shelf name
description	String	Description of the shelf
location	StoreLocation	Location where shelf is present. Indicates store:aisle

Associations

Association Name	Type	Description
temperature	Temperature	Temperature of the shelf
level	Level	Vertical level of the shelf
inventoryMap	Map<inventoryId, Inventory>	Collection of inventory within the shelf

Inventory
Items available in shelves, they are always associated with a product. Extends the ProductAssociation class.

Properties

Property Name	Type	Description
id	String	Unique identifier
capacity	Integer	Count can never be higher than this
location	StoreLocation	Location in a store should contain shelf:aisle:store

ProductAssociation
Class representing the count property and a product reference to be used by the basket and extended by inventory.

Properties

Property Name	Type	Description
count	Int	Number of products

Associations

Association Name	Type	Description
product	Product	Holds a product reference

Product
Product provides an abstraction for the types of products available for sale within the store. Products are placed on shelves as inventory. Customers take products from shelves and place them into their shopping baskets. When leaving the store and passing through the store, the products in the consumer's basket are used to compute the bill for the consumer. Products have the following attributes.

Properties

Property Name	Type	Description
id	String	Unique identifier
name	String	product name
description	String	Description of the product
size	double	Weight of the product in kg
unitPrice	int	Price in blockchain currency units
temperature	Temperature	Temperature of the product

Customer
Customer represents a person who shops at the store. Customers are recognized by the Store24X7 system through facial and voice recognition. Cameras and Microphones located in each aisle of the store monitor the location of all customers. Customers can be either Adults or Children. Customers can be a known and registered or unknown (e.g. guest). All known customers have a name for reference. Guest are not allowed to remove items from the store.

Properties

Property Name	Type	Description
id	String	Unique identifier
firstName	String	first name
email	String	Email used to register
accountAddress	String	Blockchain wallet address
currentLocation	StoreLocation	Location within the store store:aisle

timeLastSeen	Date	Last time location was updated

Associations

Association Name	Type	Description
basket	Basket	Reference to the Customer’s basket. Will get assigned when customer enters the store
customerType	CustomerType	Registered or Guest

Methods

Association Name	Type	Description
isChild()	(): boolean	Computes the birthdate of the customer with current date to see if it is more than eighteen years old

Basket
The Basket represents a shopping basket used by the customers to carry product items taken from the shelves of the store.

Properties

Property Name	Type	Description
id	String	Unique identifier of the basket

Associations

Association Name	Type	Description

productMap	Map<String, ProductAssociation>	Map of products in the basket by productId

Methods

Method Name	Signature	Description
addProduct	(count: Integer, product: Product)	Add a product to the basket, if id is already present in the productMap add amount. If not present in the basket create a new product association in the map

Sensor
Sensors are IoT devices and capture and share data about the conditions within the store such as microphone and cameras. Each sensor records data specific to its type. The data recorded by the sensor is automatically sent to the Store 24X7 System. Sensor should be an abstract class. The two child classes that extend Sensor are cameras and microphones, which should use the VideoRecording and AudioRecording behavior respectively when instantiated.

Properties

Property Name	Type	Description
id	String	Unique identifier of the sensor
name	String	Sensor name
location	StoreLocation	Location given as store:aisle

Associations

Association Name	Type	Description
recordingBehavior	IDataRecording	Behavior in charge of recording and sending data

SensorCreator
Factory class in charge of creating sensors based on the type provided. Determine if it should create Microphone, Camera or Appliance. Appliance should be created when type is *turnstile*, *robot* or *speaker.* For examples see script command examples at the end of the document.

Methods

Method Name	Signature	Description
defineSensor	(id: String, name: String, location: StoreLocation, type: String) : void	Will return a new Sensor object based on the type.

Appliance
An Appliance extends Sensor but it can also be controlled.

Methods

Method Name	Signature	Description
triggerCommand	(event: String) : void	Will execute a command based on an event

StoreLocation
StoreLocation is in charge of validating and formatting String like locations and returning the appropriate ids of the given location.

Methods

Method Name	Signature	Description
getShelfId	() : String	Based on locationCode it returns shelfId or null if its not available.

getAisleId	() : String	Based on locationCode it returns aisleId or null if its not available.
getStoreId	() : String	Based on locationCode it returns storeId or null if its not available.

StoreModelServiceException
The Store Model Service Exception is returned from the SMS API methods in response to an error condition. It captures the action that was attempted and the reason for the failure.

Properties

Property Name	Type	Description
action	String	Action that was performed
reason	String	Reason for the exception

CommandProcessorException
The CommandProcessorException is returned from the CommandProcessor in response to error conditions. It captures the command that was executed and the reason for the failure. If the command was read from a file it should also include the line number.

Properties

Property Name	Type	Description
comand	String	Command executed
reason	String	Reason for exception
lineNumber	int	Line number of the command that errored in the input file.

CommandProcessor
Utility class to feed the SMS with a set of commands using the following syntax.

Store Commands

# Define a store
```
define store <identifier> name <name> address <address>
```

# Show the details of a store, Print out the details of the store including the id, name, address, active customers, aisles, inventory, sensors, and devices. show store <identifier>

Aisle Commands

# Define an aisle within the store
```
define aisle <aisle_number> name <name> description <description>
store <store_id>
location (floor | store_room)
```
# Show the details of the aisle, including the name, description and list of shelves.
```
show-aisle <store_id>:<aisle_number>
```

Shelf Commands

# Define a new shelf within the store
```
define shelf <shelf_id> name <name> level (high | medium | low)
description <description> [temperature (frozen | refrigerated |
ambient | warm | hot )] location <store_id>:<aisle_number>
```
# Show the details of the shelf including id, name, level, description and temperature
```
show shelf <store_id>:<aisle_number>:<shelf_id>
```

Inventory Commands

# Define a new inventory item within the store
```
define inventory <inventory_id> location
<store_id>:<aisle_number>:<shelf_id> capacity <capacity> count
<count> product <product_id>
```
# Show the details of the inventory
```
show inventory <inventory_location>
```
# Update the inventory count, count must >= 0 and <= capacity
```
update inventory <inventory_location> update_count <increment or
decrement>
```

Product Commands

# Define a new product

```
define product <product_id> name <name> description <description>
size <size> category <category> unit_price <unit_price> [temperature
(frozen | refrigerated | ambient | warm | hot )]
```
# Show the details of the product
```
show product <product_id>
```

Customer Commands
# Define a new customer
```
define customer <customer_id> first_name <first_name> last_name
<last_name>
```
type (registered|guest) email_address <email> account <account_address>
# Update the location of a customer
```
update-customer <customer_id> location <store:aisle>
```
# Show the details of the customer
```
show customer <customer_id>
```

Basket Commands
# Get basket_id associated with the customer, create new basket if the customer does not already have a basket associated.
```
get_customer_basket <customer_id>
```
# Add a product item to a basket
```
add_basket_item <customer_id> product <product_id> item_count
<count>
```
# Remove a product item from a basket
```
remove_basket_item <customer_id> product <product_id> item_count
<count>
```
# Clear the contents of the basket and remove the customer association
```
clear_basket <customer_id>
```
# Get the list of product items in the basket, include the product_id and count
```
Show_basket_items <customer_id>
```

Sensor Commands
# Define device of type sensor
```
define device <device_id> name <name> type (microphone|camera)
location <store>:<aisle>
```
# Show device details
```
show device <device_id>
```
# Create a sensor event, this simulates a sensor event
```
create_event <device_id> event <event>
```

Appliance Commands
# Define device of type appliance
```
define device <device_id> name <name> type (speaker | robot |
turnstile)
location <store>:<aisle>
```
# Show device details
```
show device <device_id>
```
# Create an appliance event, this simulates a sensor event
```
create-event <device_id> event <event_description>
```
# Send the appliance a command
```
create-command <device_id> message <command>
```

Implementation Details

The core component of this design is the Store Model Service singleton that via its API it will call methods to manage the state of all domain objects within the 24X7 Store.

Sensors are a big part of the system, there will be an abstract sensor class that will define its behavior on the IDataRecording interface, the goal here is to decouple the behavior of sensors for future extensibility as stakeholders are interested in exploring other types of sensors that would record and send data that could implement this interface. For now the Microphone and Camera are available and they should implement their appropriate recording class. Appliances definitions will come in the near future so there is no need to trigger any sort of behavior for now in the triggerCommand method.

Most of the objects in the store need to be identifiable by location. Because of this, the logic to parse location strings to extract certain unique identifiers such as the shelfId, aisleId and storeId will exist in the StoreLocation class. This will allow us to modify the way locations are defined in the future, even with future information in them and still be compatible with our current design.

All objects should have a toString override to show all of their data in an easy to read fashion as the testing of the service will be done via the CLI until the rest of the modules are ready. Also, when possible, invalid operations should not halt the execution of the program and continue reading commands.

Testing

Implement a TestDriver class with the static main() method that will read scripts containing the commands provided in the CommandProcessor class dictionary. The goal of the TestDriver is to determine all corner cases and validate not only successful runs but that the service returns the appropriate errors when provided with invalid operations.

Risks

Considering that all of the experience will be fully automated there is a risk that sensors will detect events incorrectly that could impact the consumer experience. It is a core priority that consumers do not experience this behavior and if they do that the error always favors the customer (e.g if a product should be charged or not).
Because the whole system will live in memory there is a high risk of outage that will cause the loss of state of the whole 24X7 Store. The whole architecture should move to using permanent storage solutions.
With increasing the number of stores and sensors there will be a high number of requests hitting the Store Model Service API, in order to support this load there should be plans to move to a distributed system that could load balance the requests for better performance.