Store Model Service Design Document

Date: 09/23/2019
Author: Adaeze Ezeh
Reviewer(s): Jonathan Dredge

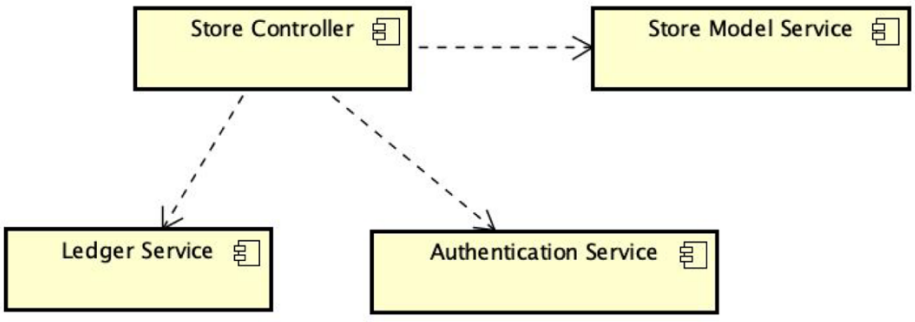
Introduction

This document describes the design of the Store Model Service. It highlights the requirements, use cases, implementation, testing procedure, and risks associated with building the Store Model Service. Like the Store Model Service Requirement document provided, this document also highlights the requirements of the Store Model, but it delves into greater details about relevant information for the successful implementation of the Store Model Service.

Overview

The Store Model Service is an important part of the Store24X7 System Architecture. It is responsible for managing the domain entities of the Store24X7 System comprising the store, aisles, shelves, inventory, products, customers, baskets, and various devices. The Store Model Service is responsible for initializing all of these entities. It also provides an API for interacting with the domain entities listed, as well as supports querying and updating of entities states.

The overall Store 24X7 System Architecture comprises four components—Store Controller, Store Model Service, Ledger Service and Authentication Service. The Store Controller interacts and updates the entities in the Store Model Service. It is responsible for monitoring the sensors and controlling the appliances in the store. It is responsible for listening and responding to voice commands from customers, and also general questions on the state of the store. The Ledger Service manages transactions, accounts, and blocks that make up the Blockchain used to processes purchase transactions in the Store 24X7 System. Lastly, the Authentication Service controls access and manages the authentication of users to the store. Its Entitlement Service identifies users through facial and voice recognitions, giving access to users to control appliances.



Caption: UML Components showing services for the Store 24X7 System, (Image from: CSCI E-97 Assignment 2 - Store 24X7 System Architecture Document)

Requirements

This section provides a summary of the requirements for the Store Model Service that this design document addresses. Please note that it does not contain a full list of all the properties and associations because those are listed in the Requirements document. Also, all entities have unique identifiers.

Customers

1. Customers must enter or exit the store through the turnstiles and are either registered or guests (not registered), adults or children.
2. Guest cannot take products out of the store
3. Once in the store, customer is assigned a basket which they can put products in.
4. Customers can interact with appliances in the store such as the robot assistants or speak directly into sensors such as microphones to make inquiries such as the location of a particular product or their account balance.
5. Customers locations are monitored and stored while within the store
6. When a customer leaves the store, their last time seen is stored

Store

1. There can be multiple stores with unique identifiers in the Store Model Service
2. Parts of the store include the floor or the store room.
3. A store contains aisles (and everything in an aisle), baskets, and customers in it

Aisles

1. Aisles must be initialized and located within a store.
2. Aisles contains shelves.
3. Customers, sensors, and appliances are located within aisles.

Shelves

1. Shelves must be initialized and located within aisles.
2. They contain inventory of the store's products.
3. Shelves have 3 level—high, medium, and low.
4. Shelves have 5 temperatures—frozen, refrigerated, ambient, warm, and hot—of which ambient is the default temperature.

Inventory

1. Inventory must be initialized and located on a shelf.
2. An Inventory contains a particular product.
3. The number of products in an inventory must not exceed its capacity.
4. When a customer places a product in their basket, the number of products in inventory reduces by the number of said products in the customer's basket.

Products

1. Products are in inventory placed on shelves in either the store floor or room.
2. Products can also be placed in customer's baskets
3. Products have 5 temperatures—frozen, refrigerated, ambient, warm, and hot—of which ambient is the default temperature.
4. Products are priced in blockchain currency of Units.

Baskets

1. Once a customer enters the store, a basket is automatically assigned to the customer.
2. A basket has a unique identifier which is the same as the customer's identifier and contains a list of products put into it by that assigned customer.

Devices

1. Devices are generally sensors but some of these sensors are appliances and they each have unique identifiers.
2. Devices must be initialized with aisles in the store.
3. Sensors are of different types such as microphones, cameras etc.
4. Appliances are of different types such as robot assistants, turnstiles, speakers etc.
5. Sensors and appliances collect data and automatically send it to the Store 24X7 System while appliances—in addition—communicate with customers and can be controlled by the Store Controller Service.

Store Model Service

1. The Store Model Services has a collection of stores
2. It implements the behaviors outlined by the Store Model Service Interface.
3. The Store Model Service configures the store and all the entities within the store.
4. It contains a list of products that can be sold at any store and customers that can visit the various stores
5. It is used by the Controller Service to monitor, access, show, control and update the store entities and customers upon request.
6. It is also used by the Controller Service to send commands to appliances and create events for sensors

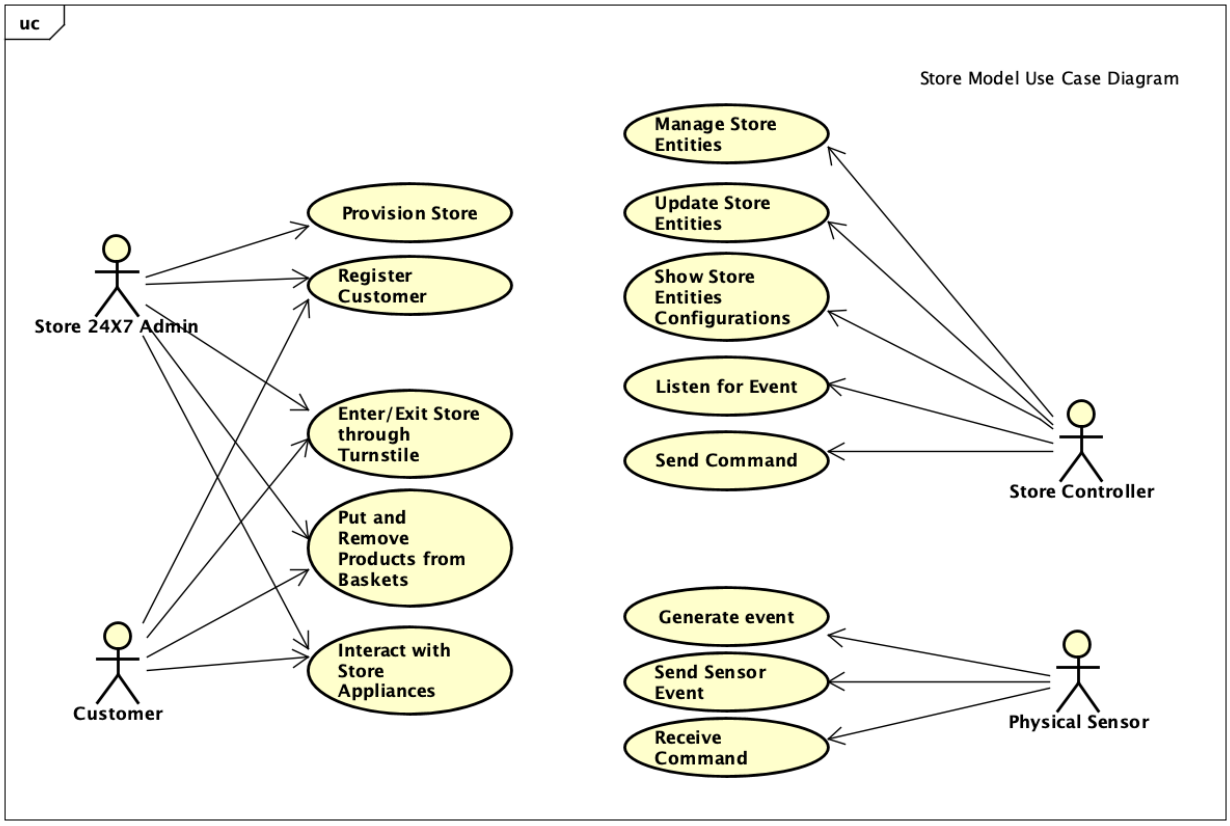
Not Supported/Required

Persistence

The Store Model System will be maintained in memory and not require persistence to limit the scope of this implementation.

Use Cases

This following use cases are supported by the Store Model System:



Actors:

The actors of the Store Model System include Store 24X7 Admin, Customers, Physical Sensors and the Store Controller.

Store 24X7 Admin

The Store Admin is responsible for provisioning the store, aisles, shelves, inventory, devices, and products. Although, not idea, the Store 24X7 Admin also can update inventory id the robot assistants are unavailable. The Admin can also perform all the use cases available to the Customer.

Customer

The customer is a user shopping at the store and is sort of a secondary actor who is allowed by the Store System to perform some use cases. Registered customers are able to enter the store through the turnstile, put and remove items from their baskets, and exit the store with products in their basket, provided there have sufficient balance in their account to cover the total due. Unregistered customers are not allowed to leave the store with products in their basket.

Customers can also interact with appliances i.e. the robot assistant or the store model service by asking for the account balance or for the location of a product within the store.

Store Controller

While Store Model Service is responsible for managing the store and all of its domains' configurations, the Store Controller monitors the sensors and controls all the appliances within the store. It accesses the Store Model Service operations through the provided API and changes the state of the store based on inputs from the sensors in the store. It supports customers by assigning them a basket ,updating and showing the contents of the basket, tracking their location and last seen date-time, checking out and transaction processing etc. It is essentially the brain of the Store 24X7 System. It also queries the state of the store and answers general questions about the store.

Physical Sensor

The physical sensors in the store are responsible for collecting data and sending data as sensor event. Appliances which are also sensors with an added functionality of receiving commands, not only collect and send data to the Store Controller but also receive commands and carry them out. For example, when a physical sensor like a camera see that a customer has changed locations within the store, it sends that information to the Store Controller which then updates the customer's location.

Use Cases:

Provision Store

Provisioning of a store includes initializing the store and all the entities within it—aisle, shelves, products, inventories, and devices. The initialization of each of these entities is detailed below:

- Store: Initializing the store includes setting the id, name and description of the store. In addition, the total count of customers, maximum occupancy, total number of available baskets, daily revenue sum, and list of aisle, sensors and appliances are initialized
- Aisle: Initializing the aisle includes defining the aisle within a store, and with a number identifier, name, and description. A list of shelves within that aisle is also initialized
- Shelf: A shelf is initialized within a store and an aisle; thus, it requires the store id and the aisle number. Additional requirements include the shelf id, name, level (low, medium or high), and temperature (frozen, refrigerated, ambient, warm, or hot)—of which ambient is the default temperature
- Inventory: Initializing inventory requires defining the inventory within a specific store, aisle number, and shelf id. The store id, aisle number, shelf id, inventory id, product, capacity of the product, and count of the product are required. The count of the product is usually between 0 and the capacity
- A new product in a store is initialized with the id, name, description, size, category, price, and temperature frozen, refrigerated, ambient, warm, or hot)—of which ambient is the default temperature. Products start off as inventory in the store but can also be placed in baskets by customers

- Creating a sensor or appliance includes initializing of the microphones, sensors, speaker, turnstile, and robot assistants within various locations of the store and aisle numbers, using unique device identifiers, names and types.

Register Customer

Creating a customer will initialize a Customer object with a customer id, first name, last name, type (i.e. registered or guest), email-address, and account.

Enter/Exit Store through Turnstile

As customers enter the store through the turnstile, they are recognized as either registered or unregistered guests, and as adults or children. When a customer attempts to exit the store with products in their basket, only customers with sufficient balances to cover the total cost of products in the basket are allowed to exit through the turnstile. Guests are not allowed to exit the store through the turnstile with products in their baskets.

Put and Remove Product in Basket

Adding to the list of products in a basket for a specified basket id. A customer can take a product from a shelf and place it in their basket. Removing a product from a basket places the object back in inventory on a shelf. This returns the basket and its contents. This returns the basket and its contents.

Interact with Store Appliances

A customer can ask the store model system through the microphones or a robot assistant for the location of a product. An aisle number and a shelf id are returned when the request is made. A customer can also ask for their account balance and the balance for the given account name is returned.

Manage Store

The Store Model Service manages the Store ensuring that the number of customers has not exceeded the maximum occupancy, checks that no inventory has run out, checks for any spills in the store that need clean up, and verifies that the total costs of products sold matched with is in the daily revenue sum. Make sure that customers enter and exit the store through the turnstile one at a time and that the daily revenue sum is updated one transaction at a time.

Update Store Entities

This includes updating store entities such as assigning a basket to a customer who enters a store, updating the customer’s location, updating the basket and inventory as a customer puts

products into their basket, and clearing the basket. Once a customer enters the store, they are immediately assigned a basket with a unique identifier. When a customer takes a product from a shelf and places it in their basket or removes a product from their basket, this takes care of updating the basket and inventory, or clearing the basket as needed. Updating the inventory requires increasing or decreasing the count for a specific inventory id. The count must be between 0 and the maximum capacity for the product. Also, as a customer moves around the store, the Store Model service will update their position using the information from the sensors. When a customer exits the store, it will save the time the customer was ‘last seen’ at.

Show Store Entities Configuration

This is a summarized use case for showing the configurations of the store, aisle, shelf, inventory, baskets, product or device, provided the entity identifier is provided. It returns the respective details of the specific entity with its locations. It also handles showing the details of a specified customer.

Listen for Event

The Store Controller listens for events from the sensors in the store so that it can act on it. The sensor sends the data to the Store Controller as event which them lead to altering the state of different entities in the store by the controller.

Send Command

The Store Controller send commands to the appliances in the store and the appliance, in turn, perform the tasks required by the commands.

Generate Event

Physical sensors around the store generate events based on the conditions in the store. Sensor events will be treated as opaque strings for now and clearly defined in the next assignment.

Send Sensor Event

These physical sensors send the events they generate to the Store Controller. Sensor events will be treated as opaque strings for now and clearly defined in the next assignment.

Receive Command

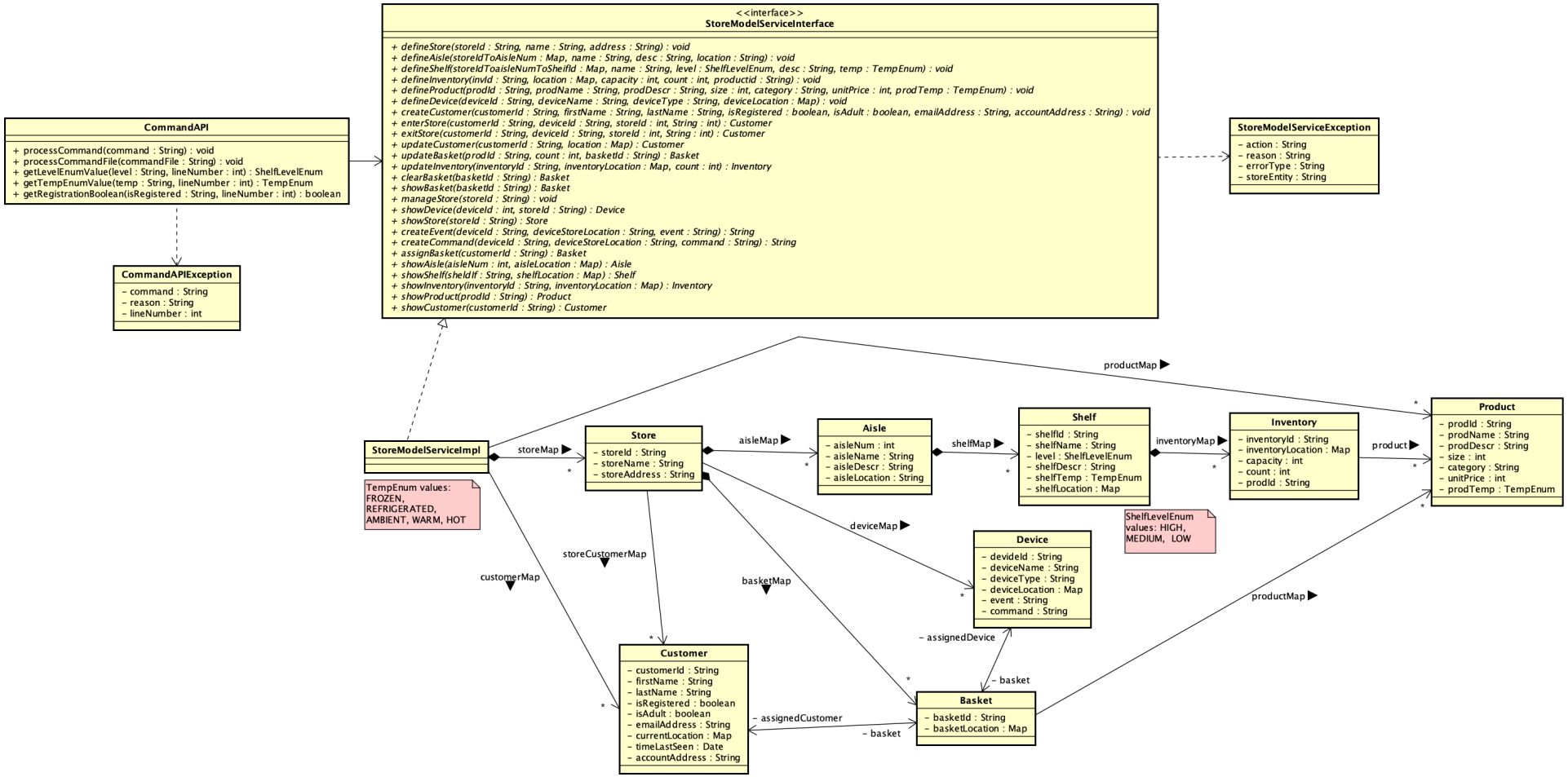
Appliances, which are also sensor receive commands from the Store Controller and then carry out the necessary task required by the commands. Appliance commands will be treated as opaque strings for now and clearly defined in the next assignment.

Implementation

This section of the document will describe the implementation details for the Store Model Service system.

Class Diagram

The following class diagram defines the Store Model implementation classes defined in this design and contained within the package “com.cscie97.store.model”.



Class Dictionary

This section specifies the class dictionary for the Store Model System. The classes must be defined within the package “com.cscie97.store.model”.

Store

Store is used to create an instance of the store with is managed and provisioned by the Store Model Service. It is initialized with a unique identifier, a name and description. It contains aisles, shelves, inventory with products and customers shopping. The Store 24X7 manages multiple stores at the same time.

Properties

Property Name	Type	Description
storeId	String	Unique identifier for the store
storeName	String	Name of the store
storeAddress	String	Address of the store

Associations

Association Name	Type	Description
aisleMap	Map<int:Aisle>	Map of Aisle number to Aisle in the store. The use of map is to take advantage of the easier search properties in Java Maps
storeCustomerMap	Map<String:Customer>	Map of all customers in currently in the store (CustomerId to Customer).
basketMap	Map<String:Basket>	Map of basket_number to Basket available in the store. The number of baskets cannot exceed the maximum capacity.
deviceMap	Map<String:Device>	Map of device_id to Device within the store

Aisle

Aisle represents the aisles in a store where shelves are placed. They can only exist within a store and deleting a store, gets rid of the aisles. It is initialized with a unique number, name, description and location.

Properties

Property Name	Type	Description
aisleNum	int	A unique number assigned to an aisle.
aisleName	String	A unique name of an aisle.
aisleDescr	String	A description of an aisle's contained product category e.g. milk, tea etc.
aisleLocation	String	Location of the aisle – either on the store floor or store room.

Associations

Association Name	Type	Description
shelfMap	Map<String:Shelf>	A map of the shelves and their associated shelf numbers located within that aisle.

Shelf

Shelf represents the platform on an aisle where inventory is placed within a store. Shelves belong on aisle and deleting an aisle in a particular store gets rid of the shelves within the aisle.

Properties

Property Name	Type	Description
shelfId	String	Unique identifier for shelf instance
shelfName	String	Name for shelf instance
shelfLevel	ShelfLevelEnum	The height of the shelf—high, low are medium.
shelfDescr	String	Description of the shelf's contents e.g. nuts, bread etc.
shelfTemp	TempEnum	The temperature of the shelf—frozen, refrigerated, ambient, warm, and hot—of which ambient is the default temperature. This is from a Temperature enum.
shelfLocation	Map<int:String>	Location of a shelf stored as a map of the aisleNumber to shelfId

Associations

Association Name	Type	Description
inventoryMap	Map<String:Inventory>	A map of all inventory located on a shelf.

Inventory

Inventory represents a class used to define the products available for sale within a store and where they are located. The location of an inventory uses storeId, aisleNumber and shelfId. Inventory maintains the count of the products which must remain between 0 and the maximum capacity of the shelf for that product.

Properties

Property Name	Type	Description
inventoryId	String	Unique identifier for an inventory
inventoryLocation	Map<String:int:String>	Location of the inventory showing map of the storeid to aisleNum to shelfId
capacity	int	Maximum capacity of a specific product the shelf can hold
count	int	The actual available number of a product in the store for sale
prodId	String	The product id of the product in the inventory

Associations

Association Name	Type	Description
product	Product	The product in the inventory.

Product

Product represents an abstraction of the different kinds of products that are sold in the store. Products are kept as inventory on shelves located either on the store floor or in the store room. Customers can take products from shelves and put in their baskets. When a registered customer leaves the store with products in their basket, the products are used to compute the bill for the customer.

Properties

Property Name	Type	Description
prodId	String	Unique identifier for product
prodName	String	Name of the product
prodDescr	String	Description of the product
size	Int	Size of the product (weight)
category	String	Category showing the type of the product e.g. dairy, nuts, etc.
unitPrice	Int	The unit price of the product in block chain currency units
prodTemp	TempEnum	The temperature of the product—frozen, refrigerated, ambient, warm, and hot—of which ambient is the default temperature. This is from a Temperature enum class.

Customer

Customer represents a person that shops at the store. A customer contains a unique identifier, and other attributes such as first name, last name, customer type, email address, current location, and last seen time. A customer can be created by the Store Admin or a User. The Store Model Service keeps a list of customers created within its system, while each store in the System keeps a list of customers in the store. Thus, upon creation, a customer is added to a map of customers in the Store Model Service. When a customer enters the store through the turnstile, they are automatically assigned a basket and their location updated as they process through the store.

Properties

Property Name	Type	Description
customerId	String	Unique Identifier for customer assigned upon creation of a customer
firstName	String	First name of the customer
lastName	String	Last name of the customer
isRegistered	Boolean	A boolean used to represent if a customer is either registered or a guest (unregistered) i.e. true if

		registered and false otherwise.
isAdult	Boolean	A boolean that is true if the customer is an adult and false if the customer is a child
emailAddress	String	The email address of the customer
currentLocation	Map<String:int>	The current location of the customer stored as a store-to-aisle map. It is updated as sensors track the position of the customer throughout the store. When a customer is outside a store, it is set to null.
accountAddressId	String	The account address of the blockchain account for customer in the ledger that is used for payments.
timeLastSeen	Date	When a customer exits the store, the date and time is recorded.

Basket

Basket represents a shopping basket used by customers to put products taken from the inventory on shelves in the store. Once a customer enters a store, they are assigned a basket. When a product is put in a basket by a customer, the basket is updated to include that product and in turn, the inventory in the store is updated. A basket has a unique identifier which is identical to the customer's id, a list of products, and the customer id it is assigned to.

Properties

Property Name	**Type**	**Description**
basketId	String	Unique Identifier for basket assigned upon creation
basketLocation	Map<String:int>	The location of the basket, which is either assigned to a customer, left at the turnstile or with a robot assistant. It is stored as a store-to-aisle map.
assignedCustomer	Customer	The customer whom the basket is assigned to.

assignedDevice	Device	The device/appliance which the basket can be assigned to. If a customer leaves a basket of products at the turnstile due to insufficient funds for the purchase, the store controller assigns the basket to a robot assistant to put the products back on the correct shelves.

Associations

Association Name	Type	Description
productMap	Map<String:Product>	Map of product id to count of product in basket, put by assignedCustomer

StoreModelServiceImpl

StoreModelServiceImpl implements the operations of the Store Model Service Interface, thus provides a top-level service interface for provisioning the store. External entities interacting with the Store Model Service accesses it through its public API.

Associations

Association Name	Type	Description
storeMap	Map<String:Store>	This is a map of storeIds to stores provisioned in the Store Model Service
customerMap	Map<String:Customer>	This is a map of customerIds to customers within the Store Model Service system. Customers can go to any stores within the system.
productMap	Map<String:Product>	This is a map of prodIds to products in the Store Model Service that can be sold in any stores.

Methods

Method Name	Signature	Description
defineStore	(storeId : String, name : String, address : String) : void	Initializes the store with a unique id, name and address
defineAisle	(storeIdToAisleNum : Map, name : String, desc : String, location : String) : void	Initializes the aisles within a store, giving an aisle number, name, description, and location

defineShelf	(storeIdToaisleNumToSheifId : Map, name : String, level : ShelfLevelEnum, desc : String, temp : TempEnum) : void	Initializes a shelf within a store, and aisle, using a shelfId, name, level, description, and temperature. The temperature is from the Temperature enum class.
defineInventory	(invId : String, location : Map, capacity : int, count : int, productid : String) : void	Initializes inventory within a particular location (store, aisle and shelf) giving a unique identification, capacity, count and productId.
defineProduct	(prodId : String, prodName : String, prodDescr : String, size : int, category : String, unitPrice : int, prodTemp : TempEnum) : void	Defines a product using a unique identifier, a name, description, size, unit price and temperature. The temperature is from the Temperature enum class.
defineDevice	(deviceId : String, deviceName : String, deviceType : String, deviceLocation : Map) : void	Instantiates a device (a sensor or appliance) within the store on an aisle, using a device id, name, and type.
createCustomer	(customerId : String, firstName : String, lastName : String, isRegistered : boolean, isAdult : boolean, emailAddress : String, accountAddress : String) : void	Creates a customer using a unique identifier, first and last name, email address, and type.
enterStore	(customerId : String, deviceId : String, storeId : int, String : int) : Customer	Changes the location of the specified customer and returns a string to identify customer.
exitStore	(customerId : String, deviceId : String, storeId : int, String : int) : Customer	Changes the location of the customer, checks for total of products in the customer's basket and updates the customer's TimeLastSeen.
updateCustomer	(customerId : String, location : Map) : Customer	Updates the customer's location and returns a string to identify the customer
assignBasket	(customerId : String) : Basket	Assign a customer a basket when they enter the store. It returns the basket details showing the customer has been assigned the basket
updateBasket	(prodId : String, count : int, basketId : String) : Basket	Adds or subtracts from the products in the basket.
updateInventory	(inventoryId : String,	Updates the specified inventory in

	inventoryLocation : Map, count : int) : Inventory	the store and returns the updated inventory
clearBasket	(basketId : String) : Basket	Clears out the products in the basket and returns the basket
showBasket	(basketId : String) : Basket	Shows the products in the basket
manageStore	(storeId : String) : void	Manages the Store by checking if any baskets with products where left at the turnstile by guests, checks for any spills in the store that need clean up, inventory that need to be restocked etc. This should ideally trigger events and commands. Due to the lack of requirements, at the moment, for the Store Controller, its functionality is a little limited. This function will most likely evolve based on the requirements on the next assignment.
showDevice	(deviceId : int, storeId : String) : Device	Shows the specified device i.e. a sensor or an appliance
showStore	(storeId : String) : Store	Shows the store and details about it – the aisles, shelfs, product list, number of customers, sensors and appliances.
showAisle	(aisleNum : int, aisleLocation : Map) : Aisle	Shows the specified aisle's number, name, description, and location
showShelf	(sheldIf : String, shelfLocation : Map) : Shelf	Shows the specified shelf's name, level, description, temperature, and location.
showInventory	(inventoryId : String, inventoryLocation : Map) : Inventory	Shows the specified inventory's location, capacity, count and product information.
showProduct	(prodId : String) : Product	Shows the specified product's name, description, size, category, unit price, and temperature
showCustomer	(customerId : String) : Customer	Shows the specified customer's first name, last name, email, type, and account.
createEvent	(deviceId : String, deviceStoreLocation : String,	Creates an event for a sensor using a unique identifier and the event. It

	event : String) : String	returns a string to describe the event.
createCommand	(deviceId : String, deviceStoreLocation : String, command : String) : String	Creates a command for an appliance using a unique identifier and the command. It returns a string to describe the command.

StoreModelServiceException

The StoreModelServiceException is returned from the StoreModelSevice in response to an error condition. The StoreModelServiceException shows the action that was attempted, the reason for its failure, the errorType, and the store entity associated with that error.

Properties

Property Name	Type	Description
action	String	Action performed e.g. define store etc.
reason	String	Reason for exception e.g. storeId already exists
errorType	String	This shows the type of error that was generated to give the Store Controller an idea on what needs resolution e.g. an initialization error, a re-stocking error etc.
storeEntity	Class	This is the store entity such as the store, an aisle, a shelf, a product etc. that is involved in the generation of an error. This allows the Controller the potential to fix the issue.

Device

Device represents either a sensor or an appliance. Sensors are IoT devices in the store that capture and share data about the conditions within the store with the Store 24X7 System. Examples include microphones for listening to voice commands from customers and cameras for monitoring the location of customers in the store. They have unique identifiers. Appliances are also IoT devices with the ability to also record and share data. In addition, appliances can be controlled by commands received from either customers or the Store Model Service. Appliances include speakers that talk to customers, robot assistants that listen and talk to customers, and perform tasks, and turnstiles that listen and talk to customers, and open and

close the turnstiles for customers to enter and exit the store. Appliances contain all the attributes in the Device class in addition to a command.

Properties

Property Name	Type	Description
deviceId	String	Unique identifier for device
deviceName	String	Name of device
deviceType	String	Type of device—either a sensor or an appliance
deviceLocation	Map<String:int>	Location of the sensor using a map of storeId to aisleNumber
event	String	A device/sensor event
command	String	A command sent to an appliance

Command API

The Command API is a utility class that supports Command Line Interface (CLI) and is used by the Store Model Service for configuring stores. The configuration script is a file containing listed commands which the CLU then calls the Model Store Service to implement.

The command syntax specifications are as follows:

Define Store

Define a store with the given identifier, name and address.

```
define-store <identifier> storeName <name> storeAddress <address>
```

Show Store

Show the details of a store by printing out the details of the store including the id, name, address, active customers, aisles, inventory, sensors, and devices.

```
show-store <identifier>
```

Define Aisle

Define an aisle within the store with a given name, description, and location

```
define-aisle <store_id>:<aisle_number> aisleName <name> aisleDescr
<description> location (floor | store_room)
```

Show Aisle

Show the details of the aisle, including the name, description and list of shelves.

```
show-aisle <store_id>[:<aisle_number>]
```

Define Shelf

Define a new shelf within the store

```
define-shelf <store_id>:<aisle_number>:<shelf_id> shelfName <name>
shelfLevel (high | medium | low) shelfDescr <description>
[shelfTempEnum (frozen | refrigerated | ambient | warm | hot )]
```

Show Shelf

Show the details of the shelf including id, name, level, description and temperature

```
show-shelf <store_id>[:<aisle_number>[:<shelf_id>]]
```

Define Inventory

Define a new inventory item within the store, on a shelf in an aisle with an id, capacity, count, and product

```
define-inventory <inventory_id> location
<store_id>:<aisle_number>:<shelf_id> capacity <capacity> count <count>
product <product_id>
```

Show Inventory

Show the details of the inventory

```
show-inventory <inventory_id> location
<store_id>:<aisle_number>:<shelf_id>
```

Update Inventory

Update the inventory count where the count must a value between 0 and the capacity

```
update-inventory <inventory_id> count <increment or decrement> location
<store_id>:<aisle_number>:<shelf_id>
```

Define Product

Define a new product with an id, name, description, size, category, unit price, and temperature

```
define-product <product_id> prodName <name> prodDescr <description>
size <size> category <category> unit_price <unit_price> [prodTempEnum
(frozen |refrigerated | ambient | warm | hot )]
```

Show Product

Show the details of a product

```
show-product <product_id>
```

Create Customer

Create a new customer with an id, first name, last name, type, email and blockchain ledge account

```
define-customer <customer_id> first_name <first_name> last_name
<last_name> isregistered (true|false) isadult (true|false)
email_address <email> account <account_address>
```

Update Customer

Update the location of a customer

```
update-customer <customer_id> location <store:aisle>
```

Show Customer

Show the details of the customer

```
show-customer <customer_id>
```

Enter Store

Show the details of the customer

```
enter-store <customer_id> device <device_id> store <store>
```

Exit Store

Show the details of the customer

```
exit-store <customer_id> device <device_id> store <store>
```

Assign Basket

Assign a basket to a customer, create a new basket if necessary

```
assign-basket <customer_id>
```

Show Basket

Show the customer associated with a basket and the list of products in the basket, include the product_id and count

```
show-basket <basket_id>
```

Update Basket

Add or subtract a product item to or from a basket

```
update-basket <basket_id> prodId <product_id> count <count>
```

Clear Basket

Remove all products from the basket

```
clear-basket <basket_id>
```

Define Device (Sensor or Appliance)

Define device of any type using an id, name, type and location

```
define-device <device_id> deviceName <name> deviceType (microphone |
camera| speaker | robot | turnstile) location <store>:<aisle>
```

Show Device (Sensor or Appliance)

Show details of a device of any type sensor using its unique id

```
show-device <device_id> storelocation <store_id>
```

Create Event

Create an event for a sensor or appliance

```
create-event storelocation <store_id> <device_id> event <event>
```

Create Command

Send an appliance a command

```
create-command storelocation <store_id> <device_id> command <command>
```

Sample Script:

```
# Define a store
define-store store001 storeName "costco" storeAddress "77 Elma St. San Luis,
CA"

# Show the details of a store
show-store store001

# Define an aisle within the store
define-aisle store001:1 aisleName aisl01 aisleDescr "dairy" location "floor"

# Show the details of the aisle, including the name, description and list of
shelves.
show-aisle store001:1

# Define a new shelf within the store
define-shelf store001:1:shelf001 shelfName aaa shelfLevel medium sheldDescr
"dairy" shelfTempEnum refrigerated

# Show the details of the shelf including id, name, level, description and
temperature
show-shelf store001:1:shelf001

# Define a new product
define-product milk001 prodName milk prodDescr "white liquid deliciousness"
size 12 category "dairy" unit_price 3 tempTempEnum refrigerated
```

```
# Show the details of the product
show-product milk001

# Define a new inventory item within the store
define-inventory inventory001 location store001:1:shelf001 capacity 20 count
11 product milk001

# Show the details of the inventory
show-inventory inventory001 location store001:1:shelf001

# Update the inventory count, count must >= 0 and <= capacity
update-inventory inventory001 count 6 location store001:1:shelf001

# Define a new customer
define-customer cust00001 first_name ann last_name fred isRegistered true
isAdult true email_address "afred@gmail.com" account ann

# Show the details of the customer before updating location
show-customer cust00001

# Update the location of a customer
update-customer cust00001  location store001:1

# Show the details of the customer after updating location
show-customer cust00001

# Assign a basket to a customer, create new basket if needed
assign-basket cust00001

# show basket information for a basket showing customer, list of products,
product_ids and count
show-basket cust00001

# Add  or subtract a product item to or from a basket
update-basket cust00001 product milk001 item_count 2

# Remove all products from the basket
clear-basket cust00001

# Add or subtract a product item to or from a basket
update-basket cust00001 product milk001 item_count 4

# Get the list of product items in the basket, include the product_id and
count
show-basket cust00001

# Define device of type sensor
define-device device001 name front_door_camera type camera location
store001:1

# Show sensor details using its unique id
```

```
show-device device001 storelocation store001

# Define device of type appliance
define-device device002 name r2d2 type robot location store001:1

# Show device details
show-device device002 storelocation store001

# Define device of type appliance
define-device device003 name robotcashier1 type turnstile location store001:1

# Show device details
show-device device003 storelocation store001

# Create a sensor or appliance event, this simulates a sensor/appliance event
create-event device001 storelocation store001 event "customer entered store
through turnstile"

# Send the appliance a command
create-command device002 storelocation store001 command "clean up on aisle 2
needed"

# Call manage-store - you should see a message about restocking inventory
manage-store store001
```

Methods

Method Name	Signature	Description
processCommand	(command:String) : void	Processes a single command. The output of the command is formatted and displayed to stdout. It also throws an CommandAPIException on error.
processCommandFile	(commandFile:String) : void	Processes a set of commands provided in a given commandFile. It throws an CommandAPIException on error.
getLevelEnumValue	(level : String, lineNumber : int) : ShelfLevelEnum	This takes in a string representing the shelf level and returns the enum value. It throws a CommandAPIException on error or with an unrecognized command.
getTempEnumValue	(temp : String, lineNumber : int) : TempEnum	This takes in a string representing the temperature and returns the enum value. It throws a CommandAPIException on error or with an unrecognized command.
getRegistrationBoolean	(isRegistered : String, lineNumber: int): boolean	This takes in a string showing if a customer is registered or not and returns

		the matching boolean. It throws a CommandAPIException on error or with an unrecognized command.

CommandAPIException

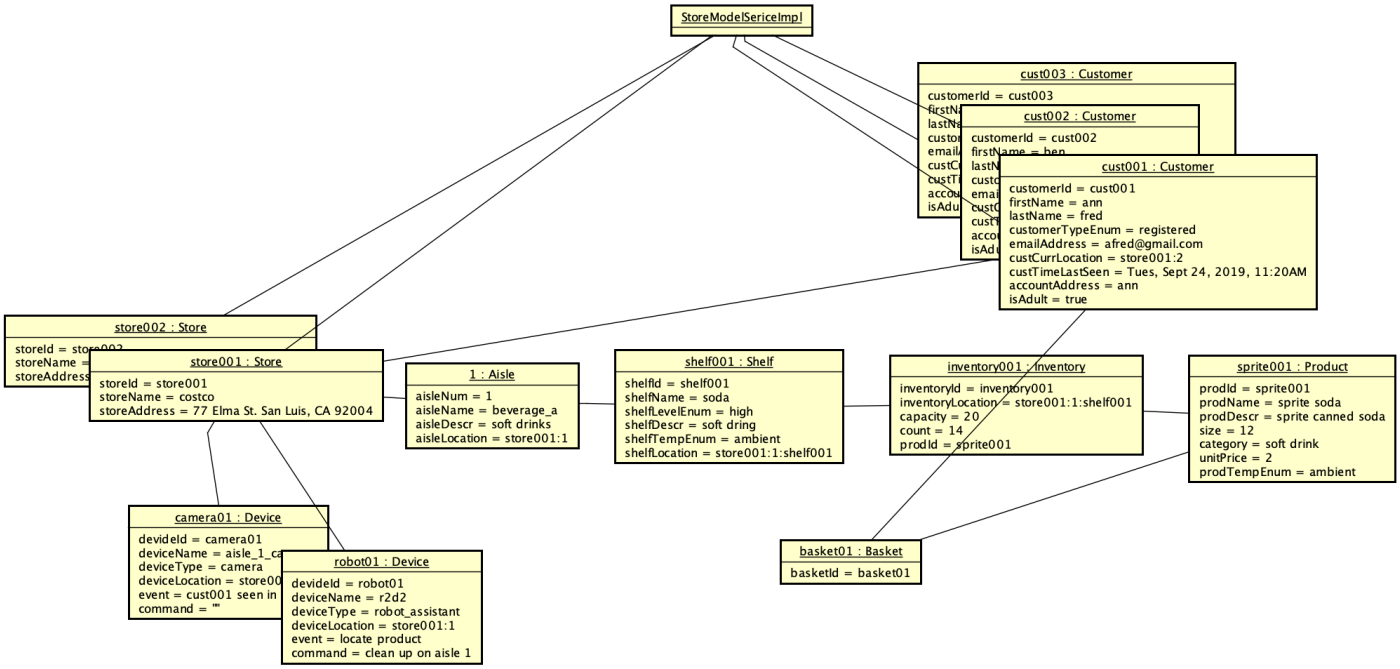
The CommandAPIException is returned from the Command API methods in response to an error condition. The CommandAPIException shows the command that was attempted and the reason why it failed. When commands are read from a file, it includes the line number of the failed command

Properties

Property Name	Type	Description
command	String	Command performed e.g. define store etc.
reason	String	Reason for exception e.g. storeId already exists
lineNumber	int	Line number of the command from a file input.

Implementation Details

For additional clarity, the instance diagram provided below shows a Store Model Service with 2 stores and 3 customers—2 of which are currently in one of the stores and assigned baskets.



The core component of this design is the Store Model Service Interface with contains all of the operations for the Store 24X7 System accessed by the provided Command API. The Model Service Class implements the operations in the Store Model Service Interface. This interface makes it possible for new implementations of an operation to be added to the Model Service Class without interrupting the normal functionality of the System—provided the interface does not change. These interface operations are responsible for managing the store and all the entities within the store such as the aisles, shelves, inventory, products, devices, customers and baskets.

Regarding the choice of data structures, a lot of the data structures used in this design are maps because of the ease with which you can search for an item. The identifiers for most of the store entities are mostly strings to account for a greater range of alphanumeric combinations and supports the specification around these ids being unique.

The Store Model Service can manage multiple store provided they are initialized with unique identifiers. Once a store is initialized, aisles must be initialized within the store, shelves must be initialized within an aisle, inventory must be initialized on a shelf, and 1 product can be kept in an inventory. When an inventory is initialized, the product's temperature needs to match the shelf's temperature, or the Store Model Service will throw an exception. This means that deleting a store would delete all of the aisles within it, and in turn, the shelfs and inventory. When a customer is created, they are added to customer map in the Store Model Service. Customers enter the store through the turnstiles. Customers can go to any stores in the Store Model Service. Once a customer enters the store, the Store Controller assigns a basket to the customer and the customer can put products into or take products out of the basket.

Once in the store, the customer is added to the map of customers in that store. Customers can visit any store managed by the Store Model Service but can only be in one store at a time. Deleting a store from the Store Model Service will not cause the customers in it to be deleted as there are part of the larger service. The cameras in the store, track the location of every customer in the store and send this information to the controller which updates the customers' locations. Customers can interact with the microphones in the store.

When a registered customer attempts to leave the store through the turnstile, the Store Controller opens the turnstile, automatically processes the purchase transaction using the Ledger Service API, provided the customer has sufficient balance in their account. Otherwise, the customer cannot leave the store with products worth more than is in their account. The customer simply leaves the basket of products at the turnstile, generating an event, and the Store Controller calls the robot assistant to come put the products away on the shelves. This means that both robots and customers can have baskets. Unregistered customers cannot leave the store with any products in their baskets. And the turnstiles can technically also have the baskets, which then triggers the Store Controller to assign the basket to a robot. It is important to pay attention to this design detail to avoid any risks.

Devices with are either sensors or appliances belong in stores, so deleting a store from the Store Model Service gets rid of the devices within it. For now, the events and commands that a generated by the sensors or sent to the appliances, respectively will be treated as opaque strings till the Store Controller is implemented.

Exception Handling

Two types of exceptions are provided in this design namely:

1. The StoreModelServiceException which is returned by the StoreModelService in response to an error condition and captures its properties in the response—the action that was attempted, the reason for its failure, the type of error, and the store entity related to the action. This additional bit of information is to improve the Store Controller's ability to resolve issues that arise in the store.
2. The CommandAPIException which is returned from the Command API methods in response to an error condition and captures its properties in the response—the command that was attempted and the reason why it failed. When commands are read from a file, the CommandAPIException includes the line number of the failed command.

Testing

A TestDriver class that implements a static main() method should be created at the start of the implementation phase. This should allow for functional testing of the individual units provided the CommandAPI is implemented early. Both the CommandAPI and the StoreModelService classes are designed to throw exceptions that capture the intended action or command and the reason for failure, aiding in debugging any error s encountered in the implementation.

The main() method should take in a command file as its single parameter and will call the CommandProcessor.processCommandFile(file:string) method, passing in the name of the provided command file. The TestDriver class should be defined within the package “com.cscie97.store.test”.

These are the aspects of the design that are tested by the test script:

- Defines store entities—store, aisle, shelf, customer, product, device, and inventory correctly,
- Catches and correctly handles errors regarding creating entities with id’s that already exist,
- Shows the configurations of the different entities correctly,
- Correctly assigns a basket to a customer upon entry to a store
- Correctly updates the map of products in a basket as a customer
- Does not allow a guest to leave the store with products in their basket i.e. throws the right exceptions
- Correct handling of errors and throws exceptions for behaviors outside the requirements
- Correct handling of unrecognizable commands

Regarding performance, implementing a custom JUnit load runner using existing test can help determine if the API designed is too slow.

Risks

The current design restricts products to being in only 2 places—in inventory or in a basket. It also allows for both customers and robot assistants to be assigned a basket. The idea behind this freedom is to mimic the behaviors of customers in a real-life store. Special care is to be taken in implementing and testing this feature to avoid unwanted risk regarding managing the baskets as the design for the Store 24X7 System grows. In addition, the baskets do not have capacity. Later versions of this may be modified to account for the volume of the basket available relative to the size of products in it.

Similar to the Ledger Service, the memory implementation leaves the system vulnerable to losing the state of the store entities and customers. Authentication will need to be implemented soon to properly identify registered customers who can exit the store with products.