Date: 9/23/21
Author: Freeman Smith
Reviewer:

Introduction:
The internet of things (IOT) allows for devices like sensors and appliances to communicate with a central service in real time. This design will utilize IOT devices for a smart store that allows customers to shop for products without need human employees.

Overview:

This design specifies the implementation of the Store Model Service. The Store Model Service manages customers, devices, including sensors and appliances, and the configuration of the stores. The Stores are separated by Aisles and each Aisle is composed of shelves. The shelves contain an inventory of products, but the inventories are managed by the store. The inventories manage the products. When customers are in a store, their location is monitored by devices, specifically sensors, which are managed by the Store Model Service. The Store Model Service also sends commands and messages to appliances such as turnstiles, speakers, and robot assistants which interact with customers. Customers in the store can use a basket which holds their products.

Requirements:

The Store Model Service is primarily responsible for managing the state of the store domain objects including:

- Store
- Aisle
- Shelf
- Inventory
- Product
- Customer
- Basket
- Sensors
- Appliances

Store
The Store is used to model a store instance. Note that the Store 24X7 is a cloud-based service and must be able to manage multiple Stores. A Store contains:

- Globally unique identifier (e.g. store100)
- Store Name (e.g. “Harvard Square Store”)
- Address (street, city, state) (e.g., “ 1400 Mass Avenue, Cambridge, MA 02138”)
- Zero or more customers
- One or more aisles and shelves
- Inventory
- Zero or more Sensors and Appliances Devices

Aisle

An Aisle is a location within the store where shelves can be placed.
Aisle attributes include:

- Number - the assigned aisle number, e.g., aisle_1
- Name - the name of the aisle, e.g., dairy
- Description - aisle description, e.g., milk, cheese, eggs, and other dairy products
- Location, either floor or storeroom

Shelf

A Shelf is a platform within an aisle within a store for inventory to be placed.
Shelf Attributes include:

- Identifier - the shelf identifier, e.g., shelf_1
- Name - the name of the shelf, e.g., milk
- Level - the height of the shelf (high, medium, low)
- Description - description of shelf contents, e.g., skim and whole milk, soy and almond
- milk
- Temperature - values: frozen, refrigerated, ambient, warm, hot. default value: ambient

Inventory

The Inventory is used to define the products available for sale within the store and where they are located. Inventory location is specified with store aisle and shelf. Inventory maintains the count of the product which must remain >= 0 and <= capacity. The inventory contains the following attributes:

- Inventory_id
- Location, Store:Aisle:Shelf
- Capacity, the maximum number of product items that can fit on the shelf
- Count, the current count of product items on the shelf.
- Product Id

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.

- Size (weight and/or volume)
- Category, the type of product, e.g., produce, dairy, deli, frozen meals
- Unit Price in blockchain currency of Units
- Temperature, values include: Frozen, Refrigerated, Ambient, Warm, Hot, default value:
- Ambient

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 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.

- Customer Identifier
- Customer First Name
- Customer Last Name
- Type (Registered or Guest)
- Email Address
- Account Blockchain Address used for billing
- Current Location
- Time last seen (updated when location updates)

Note that a registered Customer can be recognized by all stores.

Basket

The Basket represents a shopping basket used by the customers to carry product items taken from the shelves of the store. A basket is assigned to registered customers as they enter the store. A Basket has the following attributes:

- Basket Identifier
- A list of Products that are contained in the basket with a product-specific count.

Sensor

Sensors are IoT devices and capture and share data about the conditions within the store. Examples of Sensors include:

- Microphone: a listening device for receiving voice commands from customers
- Camera: monitors location of customers

Each sensor records data specific to its type. The data recorded by the sensor is automatically sent to the Store 24X7 System. Each sensor has a unique identifier. Sensors are also located within an aisle of the store. In summary, Sensors have the following features.

- Unique identifier
- Name
- Aisle, location within the store
- Sensor type

Appliance

An Appliance is similar to a Sensor since it is able to record and share data about itself or its surroundings. An Appliance differs from a Sensor since it can also be controlled. Examples of Appliances include:

- Speaker (talk to customers)
- Robot, listen to customers, talk to customers, perform tasks
- Turnstile, listen to customers, talk to customers, open and close turnstile

Store Model Service

The Store Model Service provides a top-level Service interface for provisioning stores. It also supports controlling the store appliances (Turnstiles, Robots, Speakers). Any external entity that wants to interact with the Store Model Service, must access it through the public API of the Store Model Service.

The Store Model Service provides a service interface for managing the state of the stores.

The API supports commands for

- Defining the store configuration
- Showing the store configuration
- Creating/Simulating sensor events

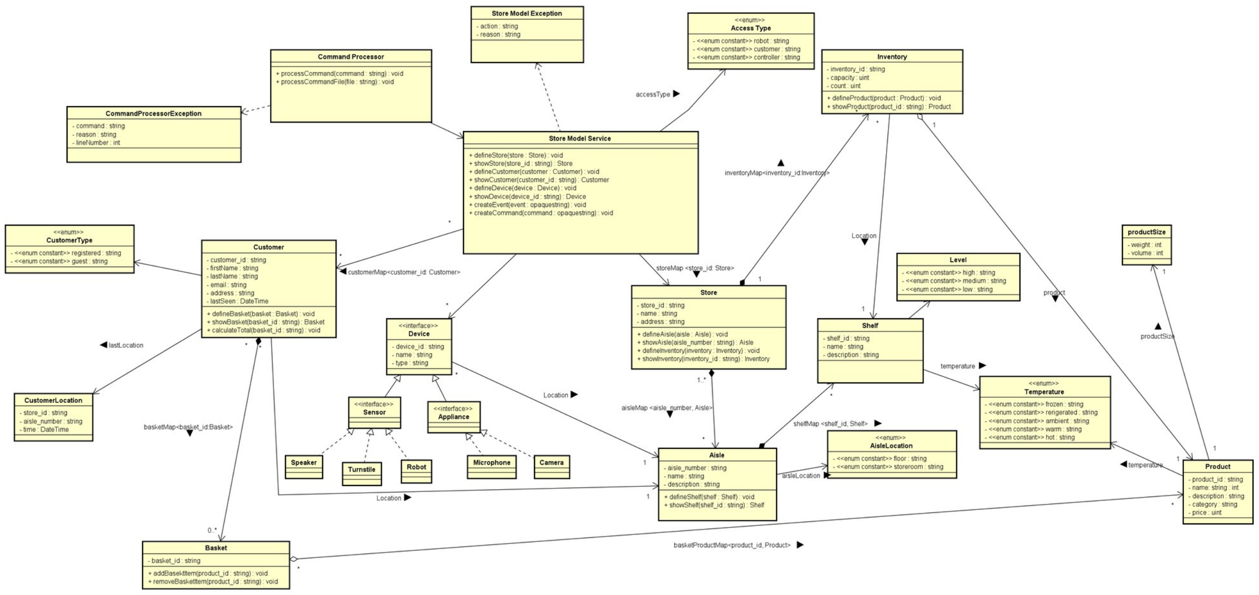
- Sending command messages to appliances
- Accessing sensor and appliance state
- Monitoring and supporting customers

All API methods should include an auth_token parameter that will be used later to support access control.

Command API

The Store Model Service supports a Command Line Interface (CLI) for configuring stores. The commands can be listed in a file to provide a configuration script. The CLI should use the service interface to implement the commands.

Class Diagram:


Store Model Exception
- action : string
- reason : string
<<enum>> Access Type
- <<enum constant>> robot : string
- <<enum constant>> customer : string
- <<enum constant>> controller : string
Command Processor
+ processCommand(command : string) : void
+ processCommandFile(file : string) : void
CommandProcessorException
- command : string
- reason : string
- lineNumber : int
Inventory
- inventory_id : string
- capacity : uint
- count : uint
+ defineProduct(product : Product) : void
+ showProduct(product_id : string) : Product
accessType
Store Model Service
+ defineStore(store : Store) : void
+ showStore(store_id : string) : Store
+ defineCustomer(customer : Customer) : void
+ showCustomer(customer_id : string) : Customer
+ defineDevice(device : Device) : void
+ showDevice(device_id : string) : Device
+ createEvent(event : opaquestring) : void
+ createCommand(command : opaquestring) : void
inventoryMap<inventory_id:Inventory>
<<enum>> CustomerType
- <<enum constant>> registered : string
- <<enum constant>> guest : string
Customer
- customer_id : string
- firstName : string
- lastName : string
- email : string
- address : string
- lastSeen : DateTime
+ defineBasket(basket : Basket) : void
+ showBasket(basket_id : string) : Basket
+ calculateTotal(basket_id : string) : void
customerMap<customer_id: Customer>
Location
productSize
- weight : int
- volume : int
Level
- <<enum constant>> high : string
- <<enum constant>> medium : string
- <<enum constant>> low : string
storeMap <store_id: Store>
Store
- store_id : string
- name : string
- address : string
+ defineAisle(aisle : Aisle) : void
+ showAisle(aisle_number : string) : Aisle
+ defineInventory(inventory : Inventory) : void
+ showInventory(inventory_id : string) : Inventory
product
productSize
<<interface>> Device
- device_id : string
- name : string
- type : string
lastLocation
Shelf
- shelf_id : string
- name : string
- description : string
temperature
CustomerLocation
- store_id : string
- aisle_number : string
- time : DateTime
<<interface>> Sensor
<<interface>> Appliance
Location
aisleMap <aisle_number, Aisle>
<<enum>> Temperature
- <<enum constant>> frozen : string
- <<enum constant>> rerigerated : string
- <<enum constant>> ambient : string
- <<enum constant>> warm : string
- <<enum constant>> hot : string
basketMap<basket_id:Basket>
shelfMap <shelf_id, Shelf>
<<enum>> AisleLocation
- <<enum constant>> floor : string
- <<enum constant>> storeroom : string
Speaker
Turnstile
Robot
Microphone
Camera
Aisle
- aisle_number : string
- name : string
- description : string
+ defineShelf(shelf : Shelf) : void
+ showShelf(shelf_id : string) : Shelf
aisleLocation
temperature
Product
- product_id : string
- name: string : int
- description : string
- category : string
- price : uint
Location
Basket
- basket_id : string
+ addBasektItem(product_id : string) : void
+ removeBasketItem(product_id : string) : void
basketProductMap<product_id, Product>
0..*
1..*
1
*

Class Dictionary

Product

The product class represents an individual product with a unique id, name, description, size in weight, category, price in units, and an enumerated temperature. The products are updated by Inventory.

Properties

Property Name	**Type**	Description
product_id	String	A unique id for each product
Name	String	Name of the product

Description	String	A description of the product
Size	productSize	The weight and or volume
Category	String	Type of product (deli, meat, fruit, etc.)
Price	Uint	Price in block chain units
Temperature	Enum	Either frozen, refrigerated, ambient, warm, or hot. Default: ambient

productSize

The product size is saved as integers of weight and ounces.

Properties

Property Name	**Type**	Description
Weight	uint	The weight in grams
Volume	uint	The volume in ounces

Inventory

The inventory tracks the count of a specific product. It also manages the products, sits on a shelf and is managed by the store.

Properties

Property Name	**Type**	**Description**
inventory_id	String	Unique ID
Capacity	Uint	Max number of that product allowed
Count	Uint	Current number of that product

Associations

Association Name	**Type**	**Description**
Product	Product	The product being managed by the inventory
Location	Shelf	The shelf where the inventory is located, this shelf object contains aisle, which contains store

Methods

Method Name	**Signature**	**Description**
defineProduct	(Product: Product):void	Defines a new product and

		begins to track its inventory.
ShowProduct	(product_id: string): Product	Given a product ID, retrieves the Product Object

<u>Shelf</u>
The shelf is an object that is stored within an aisle. It holds an inventory of products.

Properties

Property Name	**Type**	**Description**
shelf_id	String	Unique ID
Name	String	Name of the shelf (e.g. milk)
Level	Enum	Either high, medium, or low
Description	String	A description of shelf contents
Temperature	Enum	Either frozen, refrigerated, ambient, warm, or hot. Default: ambient

<u>Aisle</u>
The aisle manages shelves. Aisles are within stores and are used as the locations for devices and customers.

Properties

Property Name	**Type**	**Description**
aisle_number	String	Unique ID
Name	String	Name of the aisle (e.g. soup)
Description	String	A description of aisle contents
aisleLocation	Enum	Floor or storeroom

Associations

Association Name	**Type**	**Description**
ShelfMap	Shelf	A hashmap of shelf objects with shelf_id as a key

Methods

Method Name	**Signature**	**Description**
defineShelf	(Shelf: Shelf):void	Defines a new shelf and begins to track its inventory.
showShelf	(shelf_id: string): Shelf	Given a shelf ID, retrieves the Shelf Object

<u>Store</u>

The store is managed by the Store Model service. It manages inventory and aisles.

Properties

Property Name	**Type**	**Description**
store_id	String	Unique ID
Name	String	Name of the aisle (e.g. soup)
Address	String	Address of the store

Associations

Association Name	**Type**	**Description**
AisleMap	Aisle	A hashmap of Aisle objects with aisle_number as a key
InventoryMap	Inventory	A hashmap of Inventory objects with inventory_id as a key

Methods

Method Name	**Signature**	**Description**
defineAisle	(Aisle: Aisle):void	Defines a new Aisle and begins to track its inventory.
showAisle	(aisle_number: string): Aisle	Given a Aisle number, retrieves the Aisle Object
defineInventory	(Inventory: Inventory):void	Defines a new Inventory and begins to track its products.
showInventory	(inventory_id: string): Inventory	Given a Inventory ID, retrieves the Inventory Object

<u>Basket</u>

The basket is managed by the Customer and contains products

Properties

Property Name	**Type**	**Description**
basket_id	String	Unique ID

Associations

Association Name	**Type**	**Description**
basketProductMap	Product	A hashmap of products objects in the basket with product_id as a key

Methods

Method Name	**Signature**	**Description**
AddBasketItem	(product_id: string):void	Adds a product item to the

		basket
removeBasketItem	(product_id: string):void	removes a product item to the basket

Customer
The customer carries Products in a Basket. They are managed by the Store Model Service and sensors track which aisle they are in.

Properties

Property Name	**Type**	**Description**
customer_id	String	Unique ID
firstName	String	Customer first name
lastName	String	Customer last name
Type	Enum	Registered or guest
Email	String	Customer email address
Address	String	Customer billing address
LastSeen	DateTime	The last time a sensor picked up the customer's location

Associations

Association Name	**Type**	**Description**
Location	Aisle	The Aisle customer was last seen
Basket	Basket	The Basket objects belonging to the customer

Methods

Method Name	**Signature**	**Description**
defineBasket	(Basket: Basket):void	Defines a new Basket and begins to track its products
showBasket	(basket_id: string): Basket	Given a Basket number, retrieves the Basket Object
CalcultateTotal	(basket_id: string): int	Calculate the total cost of contents in a customer's basket

CustomerLocation
There are many different possible devices, but they will be divided into either Sensors or Appliances, which both inherit from the Device class. The devices are managed by the Store Model Service.

Properties

Property Name	**Type**	**Description**

store_id	String	ID of the store
aisle_number	String	The aisle number in the store
Time	DateTime	Time that the customer was last seen

<u>**Device**</u>

There are many different possible devices, but they will be divided into either Sensors or Appliances, which both inherit from the Device class. The devices are managed by the Store Model Service.

Properties

Property Name	**Type**	**Description**
device_id	String	Unique ID
name	String	Name of the device
Type	String	Robot assistant, speaker, turnstile, etc.

Sensor

Sensors are things like microphones or cameras, IOT devices that gather information from customers in some form. They are managed and can received simulated events from the Store Model Service.

Appliance

Sensors are things like robots or assistants, IOT devices that can interact with customers in some form. They are managed and can received simulated events from the Store Model Service. They can also receive commands from the Store Model Service.

<u>**Store Model Service**</u>

Associations

Association Name	**Type**	**Description**
storeMap	Store	A hash map of all the stores in the service
DeviceMap	Device	A hashmap of all the devices in the service
CustomerMap	Customer	A hashmap of all the customers in the service

Methods

Method Name	**Signature**	**Description**
defineStore	(Store: Store):void	Defines a new Store
showStore	(store_id: string): Store	Given a Store number, retrieves the Store Object

defineCustomer	(Customer: Customer):void	Defines a new Customer
showCustomer	(sustomer_id: string): Customer	Given a Customer number, retrieves the Customer Object
defineDevice	(Device: Device):void	Defines a new Device and manages it
showDevice	(device_id: string): Device	Given a Device number, retrieves the Device Object
createEvent	(event: opaqueString):void	Creates a new Event in one of the devices
createCommand	(command: opaqueString): void	Sends a command to one of the appliances

Store Model Exception
Handles exception from the store model service and stores the action and reason for the exception.

Properties

Property Name	**Type**	**Description**
action	String	What threw the exception
reason	String	Why the action caused the exception

<u>Command Line Processor</u>

<u>Command Syntax:</u>
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 <store_id>:<aisle_number> name <name> description <description>
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 <store_id>:<aisle_number>:<shelf_id> name <name> level (high |

medium | low) description <description> [temperature (frozen | refrigerated | ambient | warm | hot)]

# 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_id>

# Update the inventory count, count must >= 0 and <= capacity
update inventory <inventory_id> 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 <basket_id> product <product_id> item_count <count>

# Remove a product item from a basket
remove_basket_item <basket_id> product <product_id> item_count <count>

# Clear the contents of the basket and remove the customer association

clear_basket <basket_id>

# Get the list of product items in the basket, include the product_id and count
Show basket_items <basket_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>
Note: Sensor events will be defined as part of the store controller module (i.e., assignment 3).
For now, treat it as an opaque string.

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>
Note: Appliance events and commands will be defined as part of the store controller module
(i.e., assignment 3). For now, treat it as an opaque string.

Command Line Processor
Handles exception from the store model service and stores the action and reason for the exception.

Methods

Method Name	**Signature**	**Description**
processCommand	(command:string):void	Process a single command. The output of the command is formatted and displayed to stdout. Throw CommandProcessorException on error
processCommandFile	(commandFile:string):void	Process est of commands from a file. Throw

		CommandProcessorException on error

Command Processor Exception

Handles exception from the command line processor and stores the command, line number and reason for the exception.

Properties

Property Name	**Type**	**Description**
command	String	What command threw the exception
reason	String	Why the action caused the exception
LineNumber	Int	The line number of the command

Testing

Implement a test driver class called TestDriver that implements a static main() method. The main() method should accept a single parameter, which is a command file. The main method 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".
A test command file will be provided, but you should create your own test file and process that. Include the test input and output with your submission.

Risks

Customers locations are being tracked, so their data is especially sensitive. There is not any security or authentication system. Also, the store model service interacts with actual moving appliances. The robot assistants could damage property or injure customers if there were no safety features. Finally, the customer accounts are connected to their payment ledger. Financial data like this is of course very sensitive and there is no security to prevent hackers from accessing customer's financials.