SMART FOOD STORAGE

Mohamed Boutaleb, Victoria Lima Fornasero

Scuola Universitaria Professionale della Svizzera Italiana, Manno, Svizzera

mohamed.boutaleb@student.supsi.ch

Abstract

La situazione attuale dovuta alla pandemia ci ha portati ad uscire di casa solo in casi di estrema necessità e ad avere molta cautela nei piccoli gesti quotidiani, come andare al supermercato.
Con una combinazione di sensori e attuatori siamo riusciti a creare la Smart Food Storage 4.0. Il suo obiettivo principale è monitorare tutti i prodotti da conservare nel freezer, frigo, dispensa o cantina. Questo sistema tiene conto di tutti gli alimenti che vengono aggiunti alla dispensa, la loro quantità, descrizione, barcode, tipologia e peso inoltre traccia anche il consumo di questi alimenti. Con il supporto del software sviluppato in java si riesce a monitorare la quantità di alimenti che si ha a disposizione nello storage in tempo reale. Grazie al software sviluppato il sistema è in grado di segnalare anomalie delle singole categorie (temperatura massima, q.tà massima, peso massimo e necessità di rifornimento) attraverso l'uso della GUI in particolare LCD utilizzando messaggi e colori. Il sistema permette grazie all'uso di Grafana di tener traccia dei consumi di ogni singola tipologia di prodotto per permettere di programmare il riordino tenendo in conto del consumo settimanale/mensile.

1 Introduzione

L'emergenza sanitaria causata dal Coronavirus non è finita. In molti paesi ci saranno ancora molte fasi da affrontare per combattere questa pandemia. Siamo ormai tutti consapevoli di come il nostro stile di vita è cambiato radicalmente. Ora più che mai diventa importante adattarsi a questi cambiamenti e fare scelte strategiche e programmate. Infatti anche il mondo dell'industria si è adeguato molto rapidamente a questi cambiamenti andando in contro alle esigenze dell'essere umano per migliorarne il benessere e la salute. Infatti per ridurre al minimo il rischio di contagio e i tempi di attesa si è pensato a un modo per

programmare l'acquisto della spesa alimentare. Per l'appunto basti pensare a quante volte ognuno di noi si è trovato a fare la fila al supermercato (alle volte con tempi con molto lunghi) per comprare della spesa per poi accorgersi tornando a casa di non avere rifornito una certa tipologia di prodotti, o magari di aver comprato un prodotto che non utilizza molto spesso. Ed è proprio da queste esigenze che è nata l'idea dello Smart Food Storage, un progetto che usando Raspberry PI, Java, Grafana e influxDB riesce a tener traccia di tutta la spesa disponibile e che più si consuma per essere notificati quando questa sta per esaurire per poi programmare il riordino. Tutto ciò è stato possibile grazie all'uso di sensori e attuatori che identificano il prodotto e stabiliscono la categoria di appartenenza e tengono traccia dei prodotti consumati.

2 Analisi Requisiti

2.1 Prodotto

Per rappresentare il più possibile la realtà, sono state scelte delle caratteristiche per il prodotto che fossero più adatte ad un sistema database NoSQL. All'inizio la struttura di ogni prodotto era basilare ed era rappresentata da barcode, e description. Per aderire ai requisiti del progetto, sono stati aggiunti delle ulteriori informazioni alla classe riguardanti i dettagli che ogni prodotto potrebbe avere, ossia quantity, weight e consummation. Classe che abbiamo deciso essere astratta e poter poi essere ereditata.

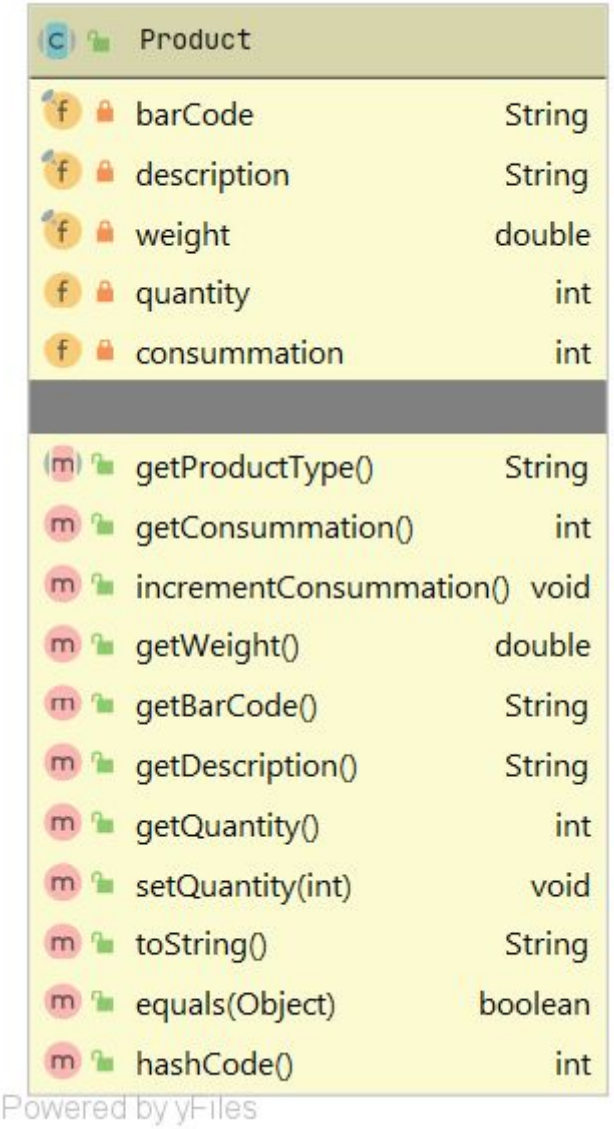

Figura 1 UML classe Prodotto

Ci siamo resi conto che era opportuno garantire che ogni prodotto venisse salvato nella categoria a cui poteva appartenere. Ci siamo attivati creando delle interfacce che rappresentassero le caratteristiche di questi prodotti e delle loro categorie. Alcune di queste interfacce (vedi Fig. 2) sono solo delle *marker interface*.

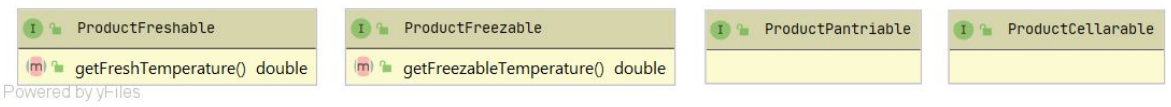

Figura 2 - UML classe Interfacce Prodotti

Per rispettare i requisiti abbiamo scelto un massimo di due tipologie di prodotto per ogni categoria, per un totale di 7 tipologie di prodotti. Ognuno di questi potrà appartenere al massimo a due categorie (vincolo imposto da noi).
Inoltre ogni prodotto avrà delle caratteristiche che il nome stesso suggerisce. Ovviamente queste tipologie di prodotti sono state stabilite secondo delle nostre preferenze e da alcuni standard presi da Wikipedia.
Di seguito i nomi dei prodotti e le categorie a cui possono appartenere:

- **AboveZeroProduct:** prodotti da dispensa e frigo
- **BelowZeroPoduct:** prodotti da congelatore
- **DisposableProduct:** prodotti da dispensa
- **FermentedProduct:** prodotti da cantina
- **LiquidProduct:** prodotti da congelatore e frigo
- **MultiUseProduct:** prodotti da cantina o dispensa
- **SolidProduct:** prodotti da frigo o dispensa

Per generare delle caratteristiche a questi prodotti ovvero: descrizioni e quantità, ci siamo appoggiato sull'uso della classe *ProductUtils* che semplicemente contiene dei campi statici final di array di stringhe per le descrizioni di ogni tipologia di prodotto, e dei metodi per ottenere randomicamente una descrizione per quel prodotto, una quantità e infine una temperatura (per i prodotti da frigo e da freezer) che quel prodotto deve mantenere.

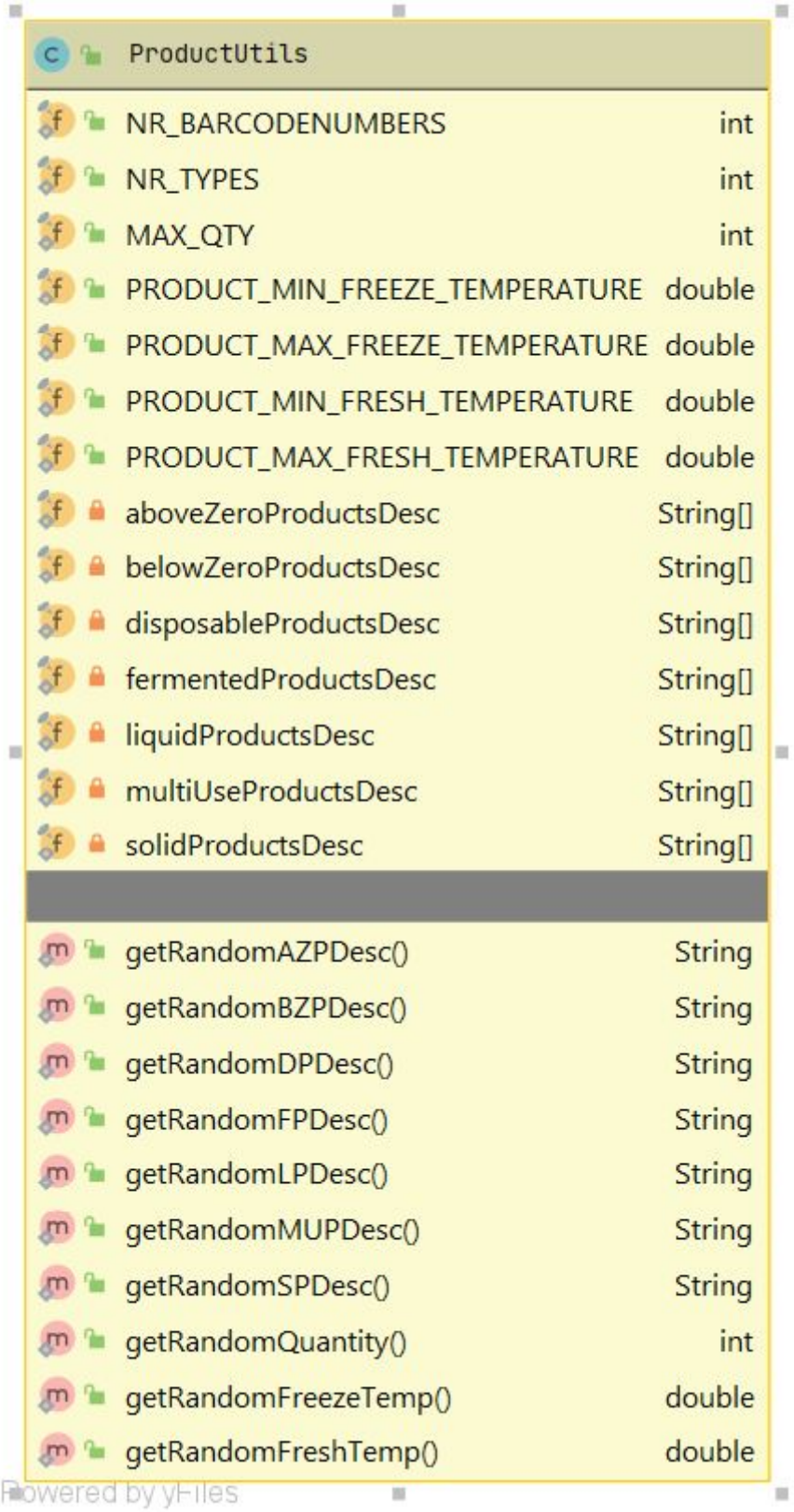

Figura 3 - UML ProductUtils

2.2 Repository e storage

Da una profonda analisi si riesce a denotare come ogni prodotto ha bisogno di essere stoccato in una precisa categoria dove potrà poi essere conservato senza che questo subisca alterazioni. Infatti si è pensato a creare delle categorie partendo dalle diverse proprietà che un prodotto può avere. Le categorie scelte e che più rappresentavano la realtà sono 4.

- **Freezer**: per i prodotti da congelamento
- **Fridge**: per i prodotti da frigo
- **Cellar**: per i prodotti da cantina
- **Pantry**: per i prodotti da dispensa

Per queste categorie abbiamo creato delle classi che ne permettano la gestione implementando interfacce che espongono i metodi più importanti. Le classi che rappresentano le categorie sono le seguenti:

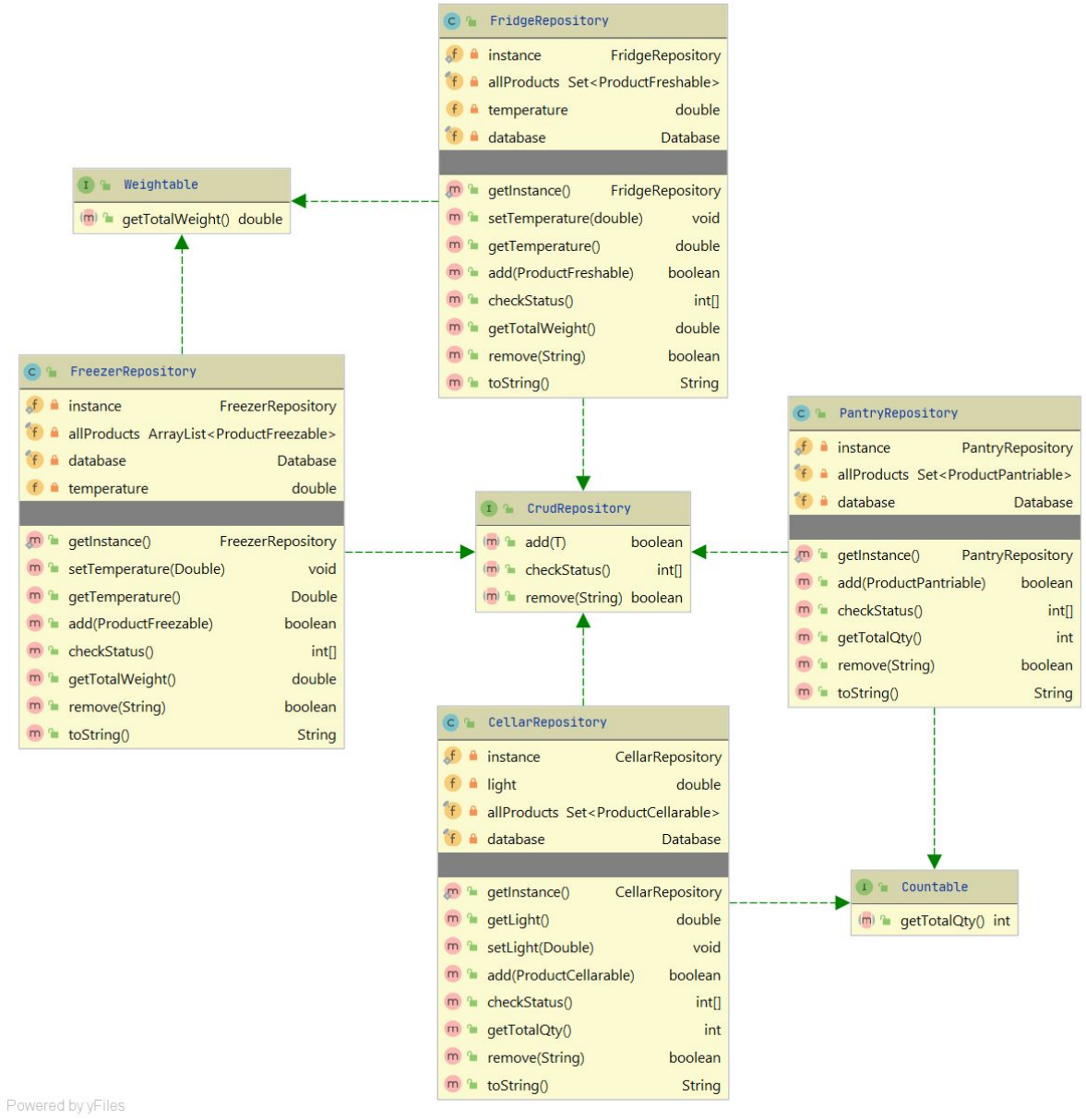

Figura 4 - UML Repositories

Come si può vedere (vedi Fig. 4) al centro della figura c'è un interfaccia che tutti i repository implementano. Questa espone tutti i metodi che servono per salvare i prodotti all'interno di una struttura dati e poterli poi aggiungere al corrispettivo measurements nel database (infatti ogni repository possiede un istanza della classe Database).

Infine abbiamo scelto dei vincoli per ogni repository a livello di quantità e peso.

Questi limiti sono stati stabiliti a nostra discrezione, e sono liberamente configurabili dall'utente finale. Anche questi sono stati gestiti con una classe all'interno del package utils.

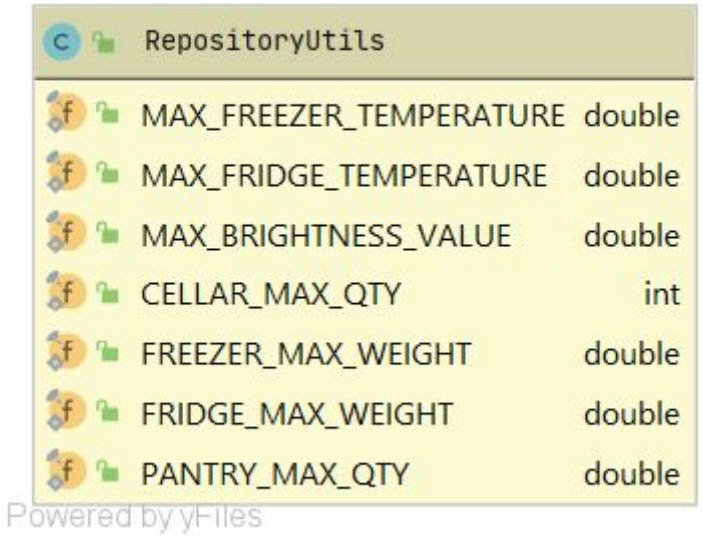

Figura 5 - UML RepositoryUtils

Ogni categoria previamente scelta, abbiamo detto implementare un'interfaccia denominata *CrudRepository*. Usando i generics siamo riusciti a generalizzare per ogni Repository il tipo che doveva gestire e accettare all' interno della struttura(ad es. *Fridge implements CrudRepository<ProductFreshable>*). Inoltre, abbiamo usato delle interfacce che espongono i metodi per stabilire i vincoli (a nostra scelta) per ogni Repository che a seconda del loro vincolo implementeranno *Countable* o *Weightable*. I limiti di peso e quantità sono stabiliti all'interno della classe *RepositoryUtils,* anche questi sono liberamente configurabili dall'utente finale.

3 Scelte tecniche

Per simulare il più possibile una stazione di smistamento prodotti abbiamo di deciso scegliere dei sensori che possono soddisfare lo schema che ci siamo prefissati. Questo schema, supporta 2 sensori di movimento, uno all'entrata - per catturare i prodotti che entrano attraverso una treadmill automatizzata - uno all'uscita - per i prodotti che vengono consumati.

3.1 Sensori e Attuatori Considerati

Per riuscire aderire a tutte le richieste della SmartFoodStorage, abbiamo analizzato le scelte disponibili nella Guida dei Sensori e abbiamo individuato quelli che soddisfavano maggiormente le nostre richieste, ossia:

- **2 Ultrasonic Ranger Sensor:** per recuperare la distanza degli oggetti sia all'entrata che all'uscita.
- **2 Temperature Humidity Sensor:** per visualizzare i valori di temperatura e umidità misurati sia per il frigorifero che per il congelatore.
- **Random Barcode Reader:** per generare codici a barre sia all'entrata che all'uscita
- **4 Lcd RGB:** per visualizzare gli status di ogni repository (attraverso messaggi) e segnalare con il colore uno status buono o cattivo
- **LoadCell:** per recuperare il peso del prodotto
- **Button:** per fermare tutto il sistema in caso di emergenza.
- **Light Sensor:** per visualizzare il valore della luminosità della cantina

3.1.1 Ultrasonic Ranger Sensor

Questo sensore all'interno della SmartFoodStorage ci permette di rilevare che un prodotto si è avvicinato, sotto la soglia stabilita di 20cm - questo range è stato scelto al fine di evitare errori nel rilevare i prodotti. Abbiamo scelto di implementare due sensori, uno all'entrata e un'altro all'uscita per far si che non ci siano sovrapposizioni di prodotti se uno deve uscire e un altro entrare contemporaneamente, evitando così falle del sistema.

3.1.2 Temperature Humidity Sensor

Sensore implementato per permettere il monitoraggio della temperatura sia all'interno del Fridge che Freezer, per segnalare tramite l'LCD la sovratemperatura e non permettere che un prodotto vada male, o scongeli quando non dovrebbe. La scelta dei valori massimi di questi sensori è stata presa, per simulare il più possibile la temperatura reale all'interno di un frigorifero e un congelatore, che corrispondono a 10 e 0 gradi rispettivamente.

3.1.3 Random Barcode Generator

Questo simulatore ci permette di ottenere dei valori random di barcode da file per poterli poi assegnare ai singoli prodotti in entrata e uscita - tenendo conto che a volte può succedere che un barcode generato per un prodotto in uscita possa non esistere nello storage, questo nella realtà non succederebbe mai. Per rendere la demo più leggera abbiamo voluto tenere al massimo 12 barcode all'interno del file. Questo per permettere di testare al meglio l'applicazione e tracciare i consumi.

3.1.4 Lcd

Abbiamo optato per questo attuatore per la sua molteplice funzione, sia quella di cambiare sfondo del display permettendo la simulazione di un led, sia la possibilità di disporre un messaggio. Ne abbiamo implementato 4, uno per ogni categoria della SmartFoodStorage, dove ogniuno di questo diventerà rosso o verde a seconda che uno solo dei requisiti previamente stabiliti non vengono rispettati(es. Temperatura o Quantità/Peso).

3.1.5 LoadCell

Questo sensore è usato solo per pesare ogni prodotto. Se questo dovesse essere già presente nello Storage allora si aumenta il peso di quest'ultimo In caso il Repository sia già abbastanza pieno allora questo verrà segnalato nell'LCD e il prodotto non verrà aggiunto allo storage.

3.1.6 Button

Questo attuatore è stato implementato come *Emergency Stop*, nel caso dovesse accadere un evento straordinario, imprevedibile o estraneo alla sfera d'azione dell'utente finale, si potrà azionare per bloccare tutto il sistema di gestione dello storage.

3.1.7 Light Sensor

Il sensore ci permette di monitorare la luminosità della cantina. Abbiamo optato di usarlo poiché ci sono molti alimenti che sono fotosensibili come il vino e l'olio d'oliva, per questo motivo la soglia usata è bassa, precisamente 40.

3.1.8 Sensori e Attuatori Rifiutati

Siccome il programma è una simulazione della realtà e i dati vengono generati casualmente, abbiamo optato per non implementare il sensore Buzzer alla nostra Smart Food Storage. Inoltre abbiamo anche considerato l'utilizzo di Led ma dopo una breve analisi, il display Lcd ci è sembrato più opportuno visto la possibilità di impostare lo sfondo (nel nostro caso rosso/verde), scartando così l'utilizzo degli attuatori di luminosità.

3.2 Analisi Display Lcd e Temperatura e Umidità

Figura 6 – Display lcd

Come detto precedentemente, abbiamo implementato 4 Lcd al fine di riuscire a monitorare lo stato di ogni categoria. Partendo dall'LCD in alto a sinistra, lo stato del freezer viene segnalato Empty o Full a seconda che sia pieno o vuoto e over-temperature se dovesse superare la temperatura stabilita (nel nostro caso 0 gradi). Non è stato gestito il caso in cui il frigo/freezer abbiano un consistente abbassamento della temperatura, questo perché a nostra discrezione non avviene molto spesso nella realtà. Lo stesso viene applicato per il Fridge, che possiamo notare in questo caso, che è "OK", ovvero, rispetta tutte le soglie precedentemente citate.

In basso a sinistra abbiamo lo stato del Pantry. In questo caso l'unico stato anomalo (non avendo nessun'altra caratteristica) che può avere è solo quello di essere vuoto o pieno.

lo stesso si applica per la Cellar, che in questo caso è vuota e ha una luminosità eccessiva.

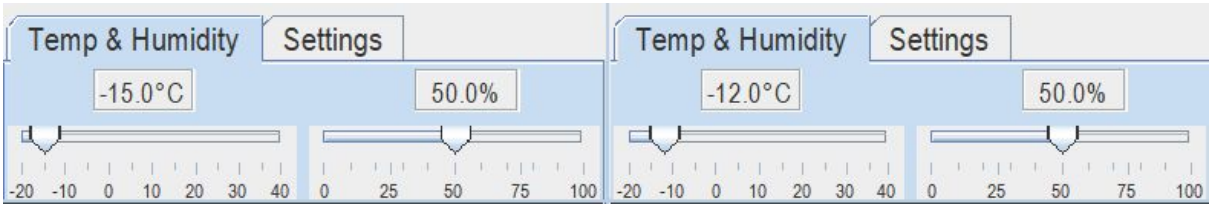

Figura 7 - Temperatura del freezer e del fridge

Per quanto riguarda la temperatura e umidità (vedi Fig. 7) vengono generati dei valori random dal sistema in un certo range da noi prestabilito che sfora il massimo di 10 - questo per rappresentare al meglio un sistema reale che non sarà mai preciso.

4 Procedure di acquisizione

Con l'obiettivo di una dimostrazione più facile del processo di acquisizione dei prodotti per la nostra Smart Food Storage, abbiamo optato per costruire 2 flowchart sotto elencati.

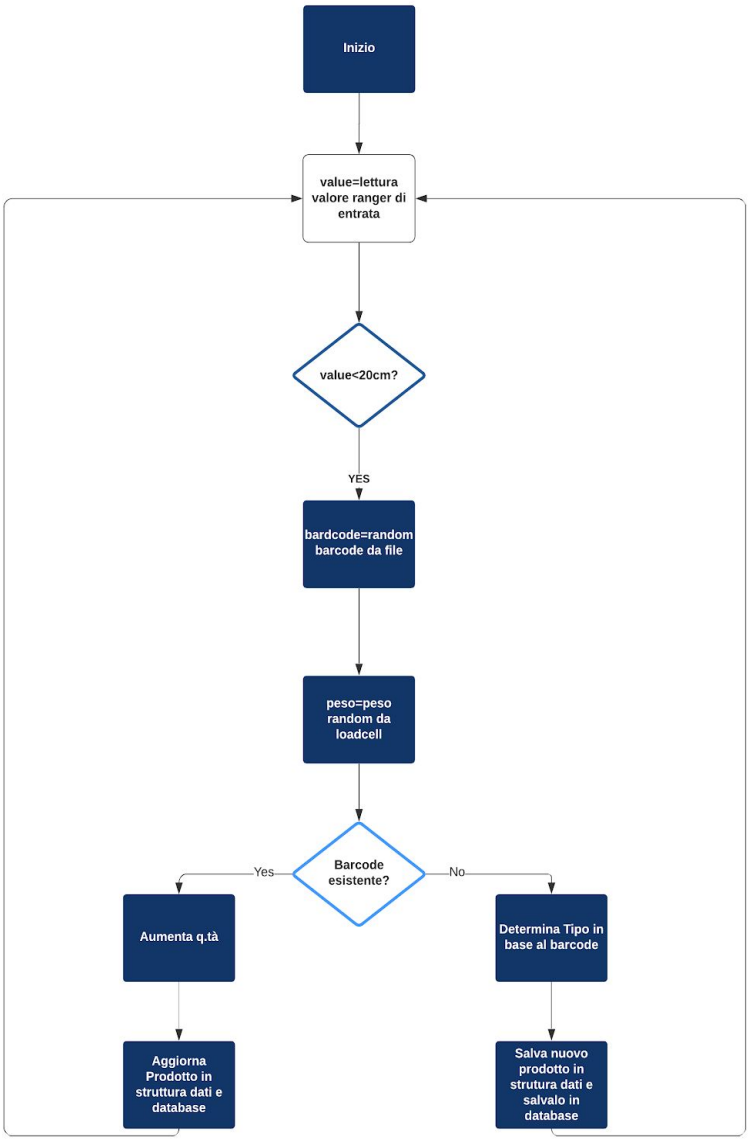

Figura 8 - Flowchart di entrata.

La procedura di acquisizione degli alimenti che entrano nella nostra dispensa automatizzata è molto semplice; come primo step si legge il valore dell'UltrasonicRangerSimulator di entrata, lo step seguente verifica se questo valore è minore di 20 centimetri, se lo è si legge un nuovo valore dal BarcodeGenerator e dal LoadCell e si controlla se un prodotto con lo stesso barcode esiste nello Storage presente a runtime attraverso una struttura dati (Set). Se questo esiste allora aggiorno la quantità (sia in struttura dati che su db) altrimenti stabilisco prima la tipologia a cui appartiene e poi lo salvo sia nella struttura dati che nel database nel suo rispettivo measurement.

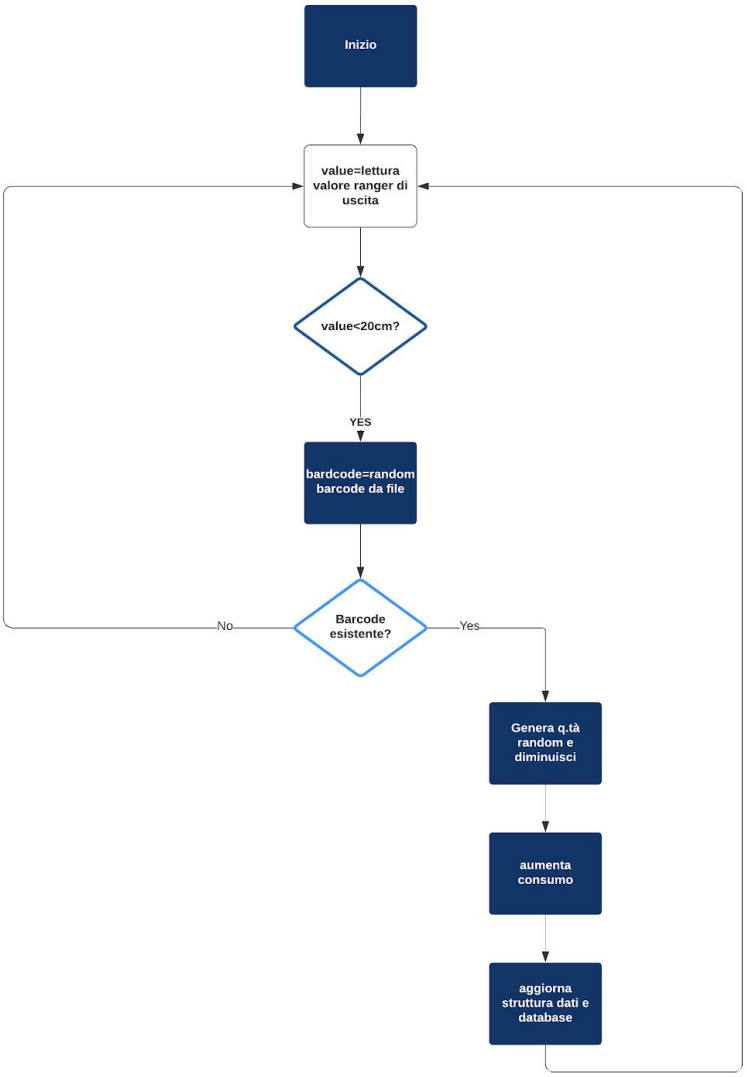

Figura 9 - Flowchart di uscita.

La procedura descritta da questo flowchart assomiglia molto a quella di entrata, tuttavia si parte con la lettura del valore del *Ranger* di uscita, che per continuare la procedura deve ottenere un valore che sia minore di 20 centimetri, altrimenti torna alla sua lettura. Il procedimento rimane invariato finché arriva alla verifica del barcode, se questo è già stato registrato nella dashboard diminuiamo la quantità disponibile (solo se questa risulta essere maggiore di 0), incrementiamo il consumo di questo prodotto e aggiorniamo la struttura dati e il database, altrimenti se il barcode generato non appartiene a nessun prodotto presente nello storage (cosa possibile visto che i barcode sono generati a random, questo nella realtà non succederebbe mai), allora si ritorna alla lettura del sensore di uscita.

5 Struttura progetto e modello dati

5.1 Struttura Progetto

Per poter rendere il progetto il più pulito e strutturato ci siamo accorti che era necessario suddividere il progetto in due moduli:

- Backend
- Frontend

Questa scelta è stata fatta per poter delegare la responsabilità ad ogni modulo. Ci siamo imposti come requisito anche la suddivisione di ogni uno di questi moduli in degli strati per separare ancor di più le responsabilità.
Il backend ha le seguenti responsabilità:

- Interfacciarsi con il frontend
- Logica generazione dei prodotti e delle loro caratteristiche
- Salvare, modificare e rimuovere i prodotti creati
- Stabilire la connessione a influxDB e creare i measurements

Mentre per quanto riguarda il frontend, le cariche che deve gestire sono le seguenti:

- Connettersi alla GUI inizializzando i logger e il GROVEPI
- Ottenere tutti i valori generati randomicamente dai sensori
- Visualizzare l'output degli stati dei repository negli lcd della GUI
- Gestire la stazione di entrata e uscita dei prodotti

Figura 10 - UML Moduli Progetto

Per suddividere ancor di più le responsabilità di ogni modulo abbiamo creato dei layer per operazione nel seguente modo:

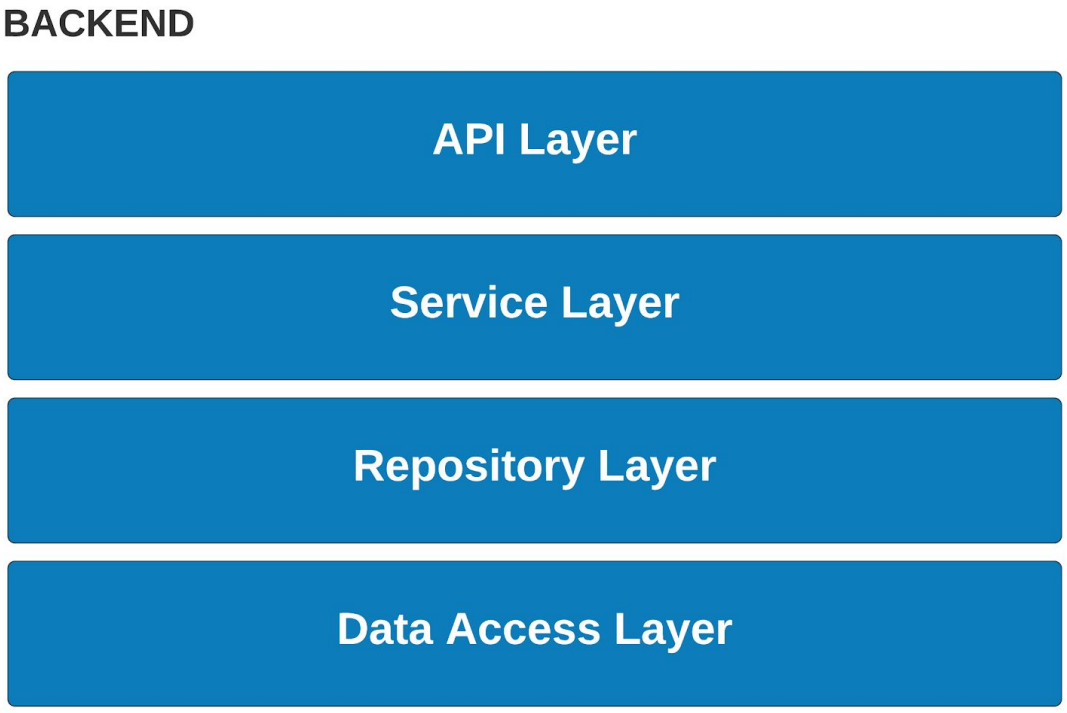

Figura 11 - Architettura a Strati Progetto

Partendo dall'alto troviamo l'API utile al frontend per interfacciarsi con il backend. Qui troviamo la classe StorageController che serve esegue tutte le chiamata allo strato inferiore. Di seguito un diagramma UML della classe:

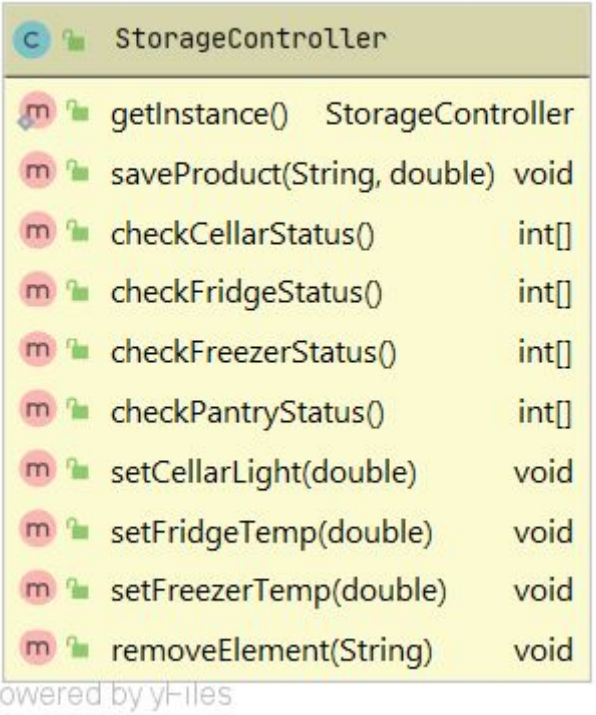

Figura 12 - UML StorageController

Scendendo giù troviamo tutti i servizi che e la logica dell'assegnazione della tipologie ad un prodotto partendo dal barcode e le chiamata ai controlli delle *Repository*. Per la logica dell'assegnazione della tipologia al prodotto abbiamo usato il check digit del barcode usando l'operatore modulo.

Figura 13 - Esempio barcode

All'ultimo livello troviamo invece l'interazione con i database. Qui vengono creati i measurements che approfondiremo nel prossimo capitolo e infine i metodi per salvare i dati.

5.2 Modello dati

A livello di java la libreria di Influx non ci permette di eseguire delle query, perciò abbiamo deciso di sviluppare delle strutture dati che permettano di tener traccia dei prodotti che abbiamo all'interno dello storage. In particolare abbiamo usato all'interno di ogni categoria dei *Set*. Qui i prodotti si identificano fra di loro attraverso il barcode.

Prima abbiamo parlato di come abbiamo soddisfatto i requisiti dei prodotti creando 7 tipologie. Il backend è progettato per essere estendibile pertanto è possibile creare delle tipologie in più. Per lo scopo dell'esercizio ne abbiamo creati solo 7. Di seguito le gerarchie dei prodotti:

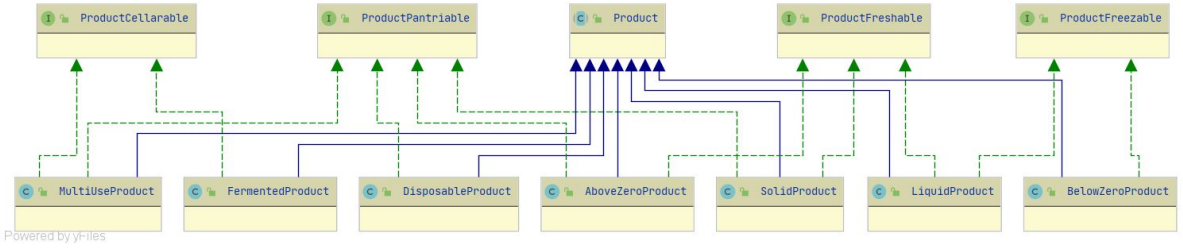

Figura 13 - UML e gerarchia di tutte le tipologie dei prodotti

Per quanto riguarda il frontend le operazioni svolte coinvolgono principalmente la GUI e l'interaction con l'utente. Questa infatti si occupa di inizializzare tutti i sensori e gli attuatori e ottenere i valori generati randomicamente e infine interrogare il backend sugli status dei *Repository* ed interpretarli nei rispettivi LCD.
Per la generazione della temperatura del frigorifero e del congelatore grazie all'aggiornamento della libreria dei simulatori in 1.7 generiamo dei valori random che superano la soglia massima di ogni *Repository* di un valore di 10 (scelto da noi) in modo di andare ogni tanto fuori soglia. La probabilità che un valore di un *Repository* cambi è del 33%. Questo perché prima di modificare gli status dei *Repository* generiamo un valore random tra 0 e 9 e applichiamo successivamente l'operatore modulo e se questo risulta essere uguale a 0 allora generiamo i valori random per i *Repository*.

6 Struttura database

Per ogni categoria abbiamo creato un measurement con l'obiettivo di monitorare tutti prodotti che entrano ed escono dalla Smart Food Storage. I campi che abbiamo scelto di implementare nei measurements sono praticamente gli stessi che rappresentano la classe Prodotto e la tipologia del prodotto in forma di stringa:

- **Barcode**
- **Quantity**

- **Description**
- **Weight**
- **Product Type**
- **Consummation**
- **Temperatura (solo Freezer/Fridge)**
- **Light (solo Cellar)**

Per il *Fridge* e *Freezer* si aggiunge un field denominato temperature, che va a dimostrare la temperatura attuale al loro interno.
Lo stesso vale per la cantina che anch'essa possiede il campo illuminazione che mostra il valore di luminosità corrente nella cantina.
Se uno dei Repository è pieno, a livello di sistema rifiutiamo i prodotti.

7 Struttura grafana

Come richiesto nel progetto al fine di monitorare il consumo dell'utente, abbiamo creato 7 pannelli, 1 per tipologia di prodotto. Questi pannelli si aggiornano in tempo reale ottenendo i dati forniti dal database monitorando i consumi.

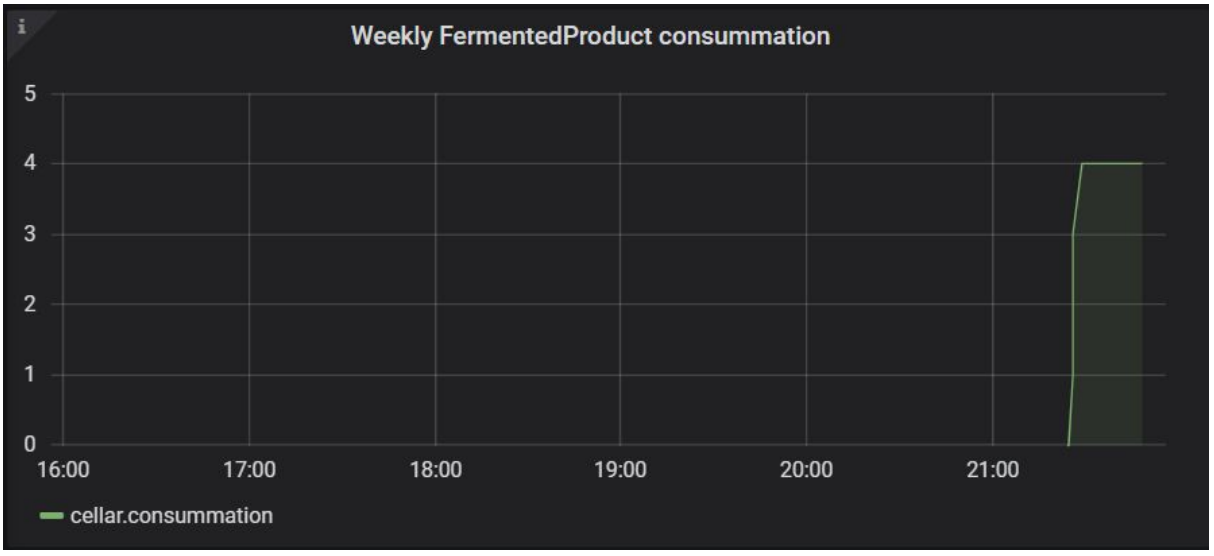

Figura 14 - Grafana modello di consumazione per tutte le tipologie dei prodotti

7.1 Query

Per la costruzione delle nostre query abbiamo optato per prendere in considerazione la somma della consumazione delle diverse tipologie di prodotto e abbiamo deciso di effettuare le query raggruppate per minuto.

Per lo scopo di fare test e debugging della demo, abbiamo deciso di stabilire che 2 minuti equivale a un'intera settimana e 5 minuti ad un mese.

7.1.2 Solid Product

Questa tipologia di prodotto si può trovare in due ubicazioni differenti, perciò ci vogliono due comandi per ottenere un solo pannello con tutte le informazioni necessarie.

```
SELECT sum(last("consummation")) FROM "autogen"."fridge" WHERE ("product_type" = 'SolidProduct') AND $timeFilter GROUP BY time(1m), barcode
```

```
SELECT sum(last("consummation")) FROM "pantry" WHERE ("product_type" = 'SolidProduct') AND $timeFilter GROUP BY time(1m), barcode
```

7.1.3 Multi Use Product

La logica utilizzata per la costruzione di questo pannello è la uguale a quella spiegata precedentemente per i prodotti solidi, ma le ubicazioni relative ai prodotti di multi uso sono dispensa e cantina.

```
SELECT sum(last("consummation")) FROM "cellar" WHERE ("product_type" = 'MultiUseProduct') AND $timeFilter GROUP BY time(1m)
```

```
SELECT sum(last("consummation")) FROM "pantry" WHERE ("product_type" = 'MultiUseProduct') AND $timeFilter GROUP BY time(1m), barcode
```

7.1.4 Fermented Product

Questa tipologia di prodotto si trova in una sola ubicazione, perciò serve solo un comando per richiamare i dati necessari (vedi Fig 14).

```
SELECT sum(last("consummation")) FROM "cellar" WHERE ("product_type" = 'FermentedProduct') AND $timeFilter GROUP BY time(1m), barcode
```

7.1.5 Disposable Product

Questo comando ci permette di osservare in un pannello la somma degli utilizzi dei prodotti della dispensa.

```
SELECT sum(last("consummation")) FROM "pantry" WHERE ("product_type" = 'DisposableProduct') AND $timeFilter GROUP BY time(1m), barcode
```

7.1.6 Below Zero Product

Questo comando ci permette di osservare in un pannello la somma della consumazione dei prodotti del congelatore.

```
SELECT sum("consummation") FROM "freezer" WHERE ("product_type" = 'BelowZeroProduct') AND $timeFilter GROUP BY time(1m), barcode
```

8 Conclusioni

Stante le limitazioni discusse e quello che è lo scopo del progetto abbiamo ottenuto un applicazione che soddisfa tutti requisiti richiesti. Un applicazione che può essere molto utile di questi tempi, dove un periodo di incertezza come questo può portarci a fare ancora le file soprattutto in paesi che stanno o devono ancora affrontare la Fase 2. Aldilà di questa situazione lo Smart Food Storage può essere usato anche da privati o aziende per ottimizzare e programmare i propri rifornimenti della spesa e sapere cosa viene consumato di più. Affrontando diversi compromessi siamo riusciti a vincere tutte le sfide e a creare una dispensa automatizzata e intelligente, che permette di monitorare, l'entrata e uscita dei prodotti, associarli ai barcode, ottenere peso, quantità, e il consumo sia settimanale che mensile di tutti i prodotti che possono appartenere ad una casa/azienda.
La Smart Food Storage è il futuro, in cui non avremmo più bisogno di guardare dentro la dispensa per sapere se mancano dei prodotti, ma ci penserà lei ad avvertirci quando viene a mancare qualcosa.