Smart Super Market

Bossi Tommaso, Nicola Landro

Parte I

Scenario applicativo

Lo scenario trattato da questo progetto consiste nella gestione e nel monitoraggio si un supermercato.

Gli scaffali sono dotati di lettori per rilevare la presenza dei prodotti (ciascuno dotato di un tag RFID), identificando quando uno specifico prodotto viene prelevato o riposto. In aggiunta, il banco frigo presenta un sensore di temperatura con relativo termostato per riportare la temperatura.

Anche i carrelli hanno dei sensori per rilevare la presenza dei prodotti scelti. Ciascun carrello presenta un'interfaccia che mostra la lista dei prodotti prelevati con la spesa totale e permette di effettuare il pagamento tramite NFC. Inoltre, i carrelli sono localizzati in modo da mostrare ai gestori la distribuzione delle persone all'interno del supermercato.

Agli ingressi/uscite sono presenti dei totem antitaccheggio che rilevano prodotti non pagati e dotati di un allarme.

È presente un interfaccia per gli addetti marketing del supermercato grazie alla quale sono in grado di abbassare o alzare i prezzi dei prodotti.

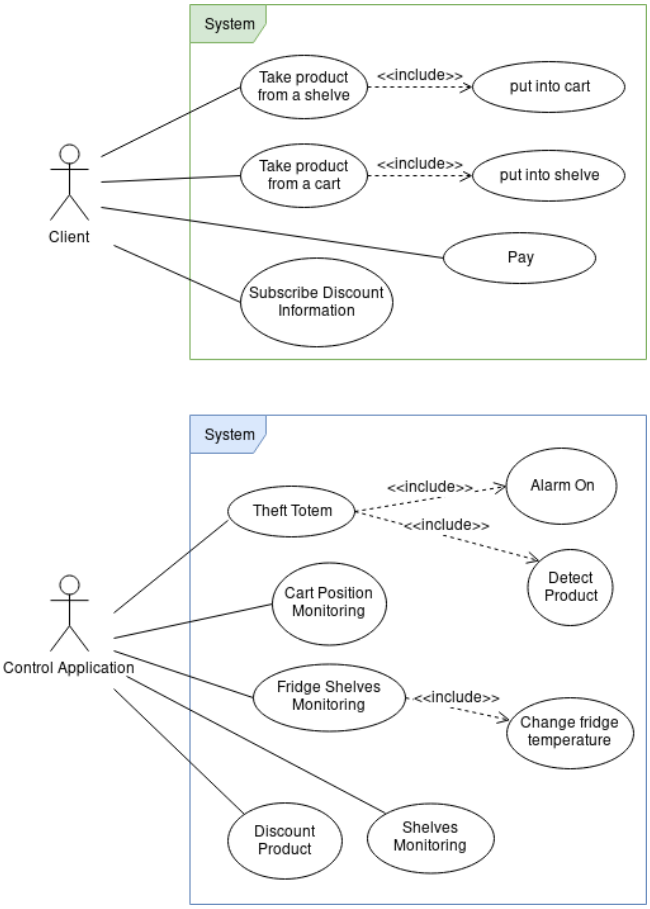

Figura 1: Use case diagram. Vengono indicate le azioni che possono svolgere i clienti del supermercato e le azioni svolte dall'applicazione di controllo.

1 Entità coinvolte

Prendendo come riferimento la figura 2, il supermercato è composto da cinque scaffali (S1, ..., S5) e da due totem antitaccheggio (T1, T2).

Gli scaffali sono dotati di due sensori:

- un rilevatore di prodotti prelevati dallo scaffale (id: `Sx_Add` dove x è il numero dello scaffale); il sensore è analogico e il valore fornito corrisponde all'id del prodotto rilevato.

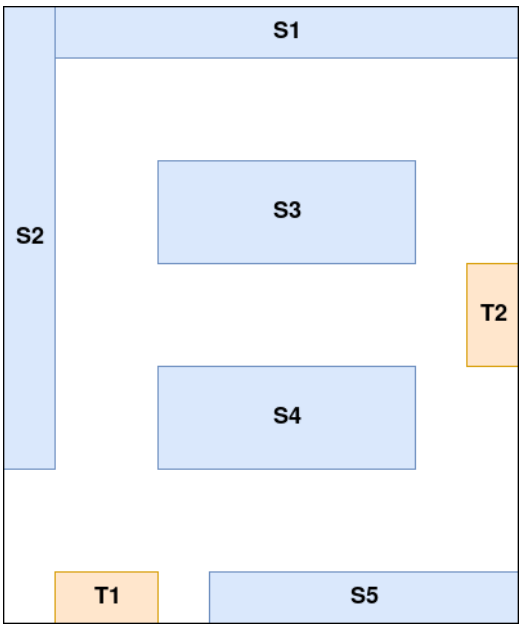

Figura 2: Schema del supermercato.

- un rilevatore di prodotti riposti sullo scaffale (id: `Sx_Rem` dove x è il numero dello scaffale); il sensore è analogico e il valore fornito corrisponde all'id del prodotto rilevato.

Inoltre, S1 viene considerato come banco frigo, ed è per questa ragione dotato di un ulteriore sensore analogico per la temperatura (id: `S1_Temp`).

I totem sono dotati di due sensori:

- un rilevatore di prodotti (id: `Tx_Detector` dove x è il numero del totem); il sensore è analogico e il valore fornito corrisponde all'id del prodotto rilevato.
- un allarme (id: `Tx_Alarm` dove x è il numero del totem); il sensore è digitale e il valore indica se il segnale di allarme è attivo o disattivo.

Il supermercato ha a disposizione venti carrelli (Cart_0, ..., Cart_19), ciascuno dotato di quattro sensori:

- un rilevatore di prodotti aggiunti al carrello (id: `Cart_x_Add` dove x è il numero del carrello); il sensore è analogico e il valore fornito corrisponde all'id del prodotto rilevato.
- un rilevatore di prodotti rimossi dal carrello (id: `Cart_x_Rem` dove x è il numero del carrello); il sensore è analogico e il valore fornito corrisponde all'id del prodotto rilevato.
- un sensore di posizione (id: `Cart_x` dove x è il numero del carrello); il valore fornito indica la posizione del carrello all'interno del supermercato ed è espresso come coppia di coordinate x, y.
- un lettore NFC (id: `Cart_x_NFC` dove x è il numero del carrello) utilizzato per effettuare il pagamento dei prodotti presenti nel carrello.

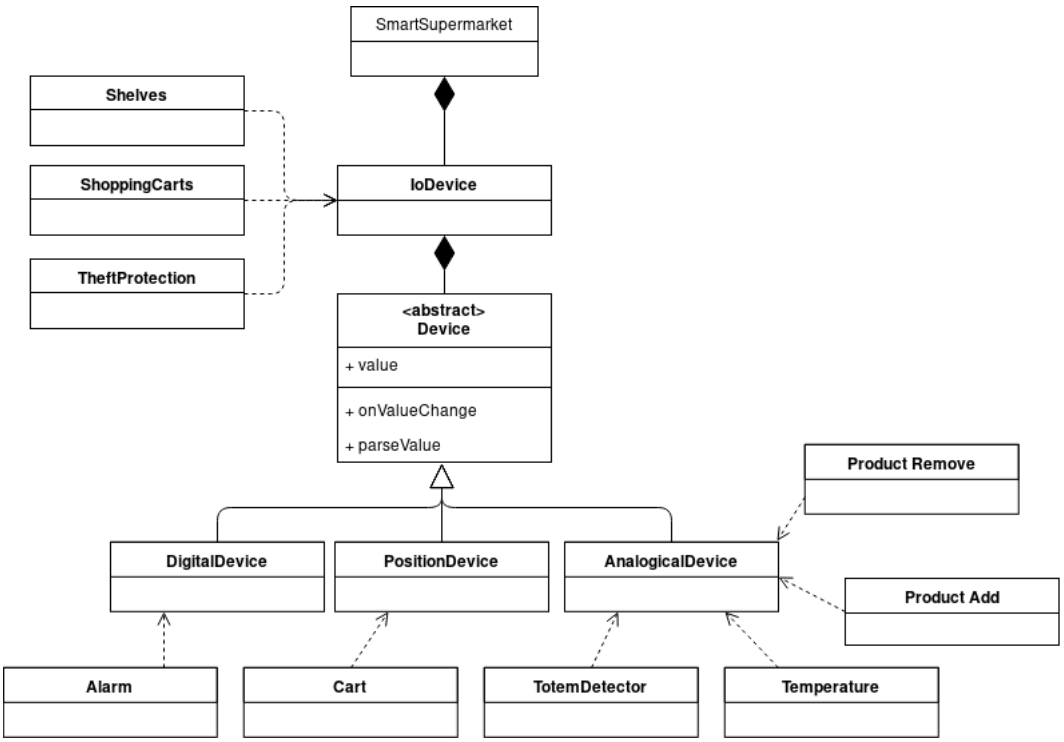

Figura 3: Class diagram.

2 Protocolli di comunicazione

2.1 Sensori simulati

Tutti i sensori illustrati nella sezione 1 sono simulati tramite dei server TCP. Nello specifico si tratta di tre server che simulano:

1. gli scaffali (in ascolto sulla porta 9090);
2. i carrelli (in ascolto sulla porta 9091);
3. i totem (in ascolto sulla porta 9092).

Un'interfaccia che mostra il valore corrente di ciascun sensore è disponibile tramite un server HTTP attivo sulla porta 3000.

Il protocollo di comunicazione adottato dai tre server è il medesimo. Tale protocollo è composto da due possibili messaggi:

1. `SET <id> <value>` per impostare il nuovo valore del sensore; `<id>` è l'id del sensore e `<value>` è il nuovo valore da assegnare.
2. `GET <id>` per ottenere il valore corrente del sensore; `<id>` è l'id del sensore.

Il server risponde a tali messaggi con il messaggio `OK <id> <value>` dove `<id>` è l'id del sensore mentre `<value>` è il nuovo valore appena assegnato oppure il valore richiesto.

Nel caso il comando inviato non sia corretto il server risponde con il messaggio `ERROR <code> <msg>` dove `<code>` è un codice associato al tipo di errore mentre `<msg>` è un messaggio esplicativo.

A seconda del tipo di dispositivo, `<value>` può assumere valori differenti:

- un dispositivo digitale ammette valori `true` oppure `false`;
- un dispositivo analogico ammette un numero floating point;
- un dispositivo di posizione ammette valori nella forma `<x>,<y>` dove sia `<x>` sia `<y>` sono due numeri floating point.

2.2 Gestione pagamenti

Al fine di simulare il pagamento tramite NFC, è stato implementato un server a cui connettersi per effettuare la transazione. Tale server comunica in TCP utilizzando messaggi in formato JSON.

Il server accetta messaggi nella forma

```
{
    request : "payment",
    sender : <senderId>,
    payment : <encryptedPayment>
}
```

dove `<senderId>` è l'id di un account registrato sul server, mentre `<encryptedPayment>` è un campo cifrato (con la chiave associata all'account indicato) che contiene le informazioni sulla transazione da effettuare. Il campo payment decifrato è nella forma

```
{
    from : <fromId>,
    to : <toId>,
    amount : <amount>,
    timestamp : <timestamp>
}
```

dove `<fromId>` e `<toId>` sono rispettivamente l'id dell'account pagante e pagato, `<amount>` è la somma pagata e `<timestamp>` è la data di pagamento.

Il server risponde con il messaggio

```
{
    payment : <encryptedPayment>
}
```

dove `<encryptedPayment>` è lo stesso campo cifrato illustrato in precedenza, a cui è stato aggiunto un campo `result` che indica se l'operazione è andata a buon fine.

Allo scopo di semplificare la simulazione, il server ha un comportamento deterministico: se l'id del pagante è un numero pari il pagamento va a buon fine altrimenti fallisce.

2.3 MQTT

Sono stati definiti alcuni topic MQTT:

1. `supermarket/alert/<Tx>`: avvisi sui furti rilevati dal totem Tx (dove x è il numero del totem).
2. `supermarket/shelf/S1/temperature`: informazioni sulla temperatura rilevata nel banco frigo S1.

3. `supermarket/product/<Pid>/price`: notifiche sulla variazione di prezzo del prodotto Pid.

4. `supermarket/product/<Pid>/availability/<Sx>`: notifiche sulla variazione di disponibilità del prodotto Pid (dove Pid indica l'id del prodotto) nello scaffale Sx (dove x è il numero dello scaffale).

Parte II

Flussi realizzati

3 Scaffali

3.1 Controllo della temperatura

Il flusso per il controllo della temperatura si occupa di mantenere la temperatura del banco frigo S1 stabile sui 2°C. I cambiamenti di temperatura vengono indicati nel grafico presente nella dashboard e pubblicati sul topic `supermarket/shelf/S1/temperature` (vedi sezione 2.3).

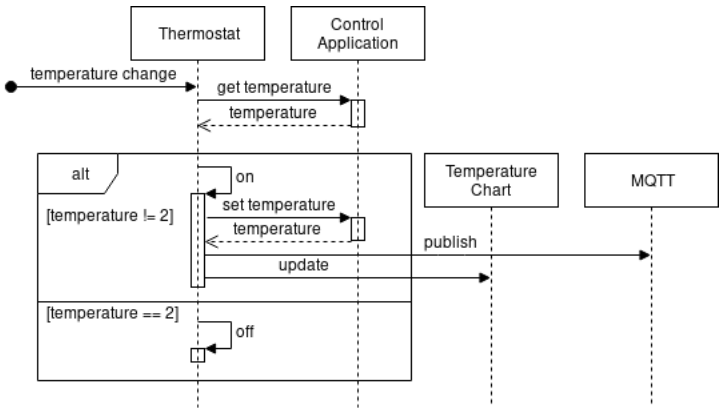

Figura 4: Sequence diagram inerente alla gestione della temperatura.

3.2 Aggiunta e rimozione dei prodotti

Il flusso che controlla l'aggiunta e la rimozione di prodotti dagli scaffali si occupa di aggiornare la disponibilità dei prodotti sul database e di pubblicare la nuova quantità disponibile sul topic `supermarket/product/<Pid>/availability/<Sx>` (vedi sezione 2.3).

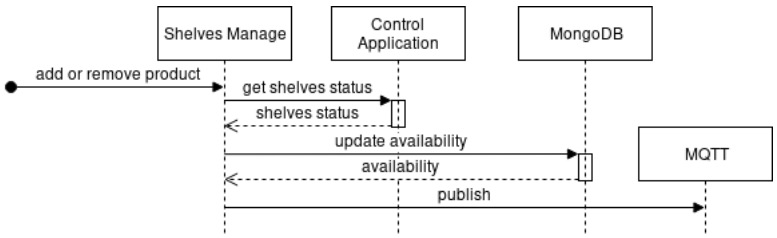

Figura 5: Sequence diagram inerente all'aggiunta e rimozione dei prodotti dagli scaffali.

4 Totem

4.1 Rilevazione prodotti rubati

Quando un prodotto non pagato viene rilevato dai totem antitaccheggio viene azionato un allarme per 5 secondi e viene pubblicata l'informazione sul topic `supermarket/alert/<Tx>` (vedi sezione 2.3).

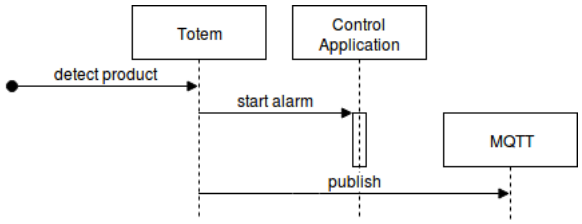

Figura 6: Sequence diagram inerente alla gestione dei tentativi di furto.

5 Carrelli

5.1 Spostamenti casuali dei carrelli

Questo flusso di supporto serve a rendere più evidente il funzionamento della mappa con i carrelli. Questo flusso si occupa di spostare casualmente i carrelli all'interno del supermercato.

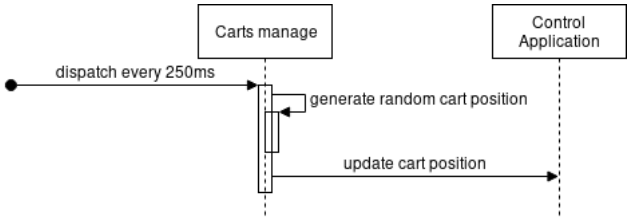

Figura 7: Sequence diagram inerente allo spostamento casuale dei carrelli.

5.2 Creazione mappa con la posizione dei carrelli

Questo flusso si occupa di richiedere la posizione dei carrelli e la disposizione degli scaffali per poter generare e mantenere aggiornata la mappa dei carrelli e la heatmap che mostra le zone più frequentate del supermercato.

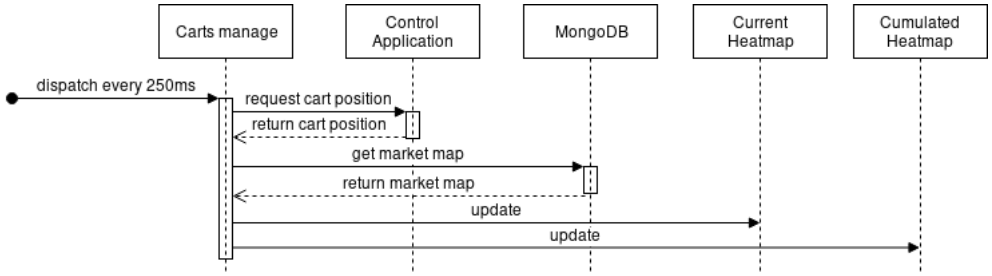

Figura 8: Sequence diagram inerente alla generazione della mappa dei carrelli.

5.3 Aggiunta e rimozione dei prodotti

Il flusso che controlla l'aggiunta e la rimozione di prodotti dai carrelli si occupa di aggiornare la lista dei prodotti prelevati sul database e di aggiornare lo schermo del carrello che mostra tale lista.

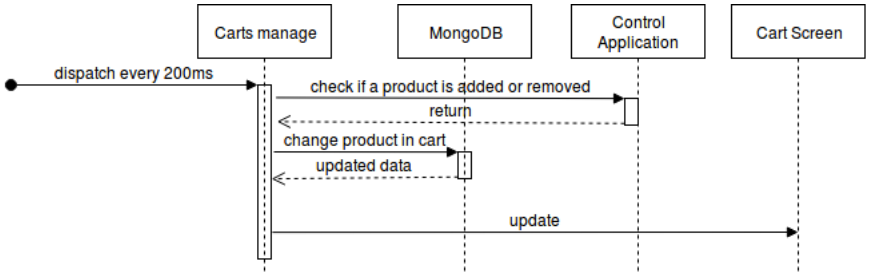

Figura 9: Sequence diagram inerente all'aggiunta e rimozione dei prodotti dai carrelli.

5.4 Gestione dello schermo con la lista prodotti

Ogni carrello è dotato di uno schermo che mostra la lista dei prodotti inseriti con i relativi prezzi. Ogni volta che viene aggiunto un prodotto al carrello (o rimosso) le informazioni sullo schermo vengono aggiornate.

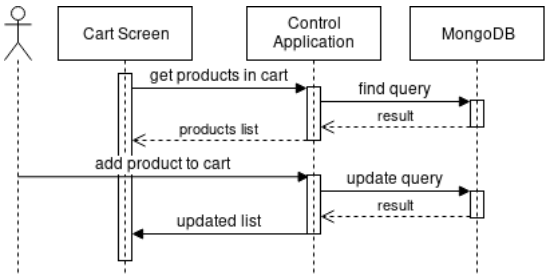

Figura 10: Sequence diagram inerente alla gestione dello schermo del carrello.

5.5 Gestione del pagamento

Per poter effettuare un pagamento è necessario aver inserito almeno un prodotto nel carrello. Selezionando il bottone di pagamento sull'interfaccia del carrello, viene attivato il lettore NFC per poter effettuare la transazione. La transazione viene mandata al server di pagamento quindi il risultato dell'operazione viene mostrato a video.

I messaggi con il server sono cifrati con AES.

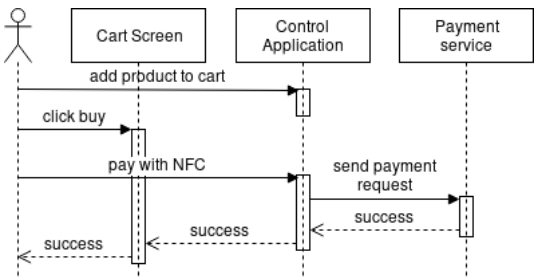

Figura 11: Sequence diagram inerente alla gestione del pagamento tramite NFC dal carrello.

6 Gestione prodotti

6.1 Form per l'aggiornamento dei prezzi

La form per aggiornare i prezzi è una semplice interfaccia grafica pensata per consentire al gruppo marketing del supermercato di aggiornare i prezzi dei prodotti. È possibile cercare nel listino i prodotti e poi inserendo id prodotto e prezzo si potrà aggiornare con successo. Questo scatena un messaggio MQTT a tutti i sottoscritti e anche una chiamata al server del *Smarketbot* firmata, che causa un brodcast a tutti gli iscritti al bot che hanno in precedenza scritto `/subscribe`, come spiegato nella sezione 8.

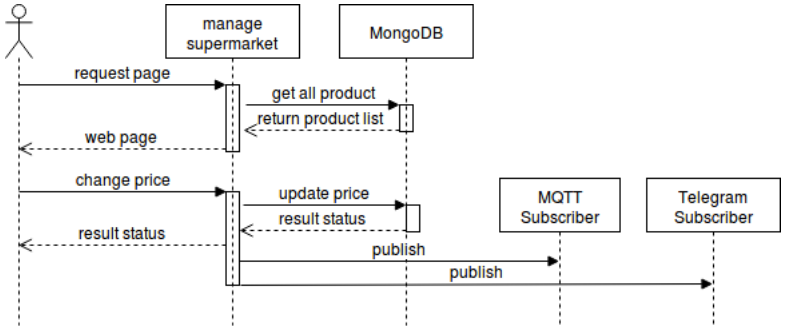

Figura 12: Sequence diagram inerente all'aggiornamento dei prezzi.

7 Subflows

Per rendere più facili e comprensibili i flow e per mettere a fattor comune parti di flow utilizzati in più punti, abbiamo creato alcuni subflow.

7.1 TCP Set e Get

Sono due flussi utilizzati per cambiare o ottenere i valori dei vari sensori. In questi flussi è racchiusa la logica per decidere quale delle tre connesioni TCP utilizzare e per comporre la richiesta. In caso si verifichi un errore, questo viene stampato nei log di debug.

7.2 TCP fake payment

Questo flusso è in grado di simulare il meccanismo di pagamento criptando i dati e inviandoli al server per poi decriptare la risposta e gestirla per informare l'utilizzatore del risultato della transazione. Per i dettagli del protocollo vedi la sezione 2.2.

7.3 Query Mongo

Sono stati realizzati vari subflow che utilizzano MongoDB ed eseguono le seguenti azioni: ottenere la lista di tutti i prodotti del supermercato o dei carrelli, ottenere la mappa del supermarket; aggiornare la disponibilità dei prodotti o il loro prezzo e cambiare i prodotti presenti in un carrello.

7.4 Every 250 ms

Questo è un piccolo subflow di supporto che è stato utilizzato in più punti per velocizzare la simulazione. Ogni secondo genera un evento che viene mandato a vari nodi delay allo scopo di produrre quattro eventi distanziati di 250 millisecondi.

Parte III

Dettagli implementativi

I vari server sono stati tutti implementati in Node.js e sono attivi sulle porte:

- 9090, 9091 e 9092 - i server TCP relativi ai sensori discussi nella sezione 2.1;
- 3000 - il server HTTP che fornisce un'interfaccia per visualizzare i valori dei sensori;
- 10000 - il server TCP che simula un sistema di pagamento.

La versione di Node.js utilizzata è la 9.11.1. Per l'esecuzione del progetto si rimanda alla sezione 10.

8 Smarketbot

In aggiunta alle notifiche sull'aggiornamento dei prezzi tramite MQTT si è deciso di sviluppare Smarketbot, un semplice bot per Telegram[1] in grado di notificare la medesima informazione a chiunque abbia l'applicazione installata.

Per utilizzare Smarketbot su Telegram bisogna accedere alla url http://t.me/NicTomBot dal proprio dispositivo con installato Telegram. Dal bot sono possibili due azioni:

- `/subscribe` con la quale ci si sottoscrive per ottenere informazioni dal bot;
- `/unsubscribe` con la quale è possibile smettere di ricevere informazioni dal bot. Tutti gli altri messaggi inviati al bot saranno ignorati.

Il bot è stato sviluppato e installato a parte per poter garantire un server sempre attivo. Tale server è un Orange Pi PC 2[2], con sistema operativo Armbian, questo è stato possibile grazie alla compatibilità di Node.js con l'architettura ARM.

Anche il bot è sviluppato in Node.js ed il codice sorgente è disponibile al link https://github.com/nicolalandro/BrodcastChatbot.git.

9 Dataset con prodotti reali

Per avere dei prodotti che abbiano nomi, reparti e prezzi realistici si è deciso di prenderli da un supermercato reale.

[1] https://telegram.org/, è disponibile l'app per Android e iOS, l'applicazione per PC e anche una web app.
[2] http://www.orangepi.org/orangepipc2/

È stato dunque usato Web Scraper[3], uno web scraper gratuito disponibile come plugin per Chrome. Usando tale strumento è stato possibile ottenere i dati dal supermercato Il Gigante. La sitemap utilizzata è la seguente:

```
{
  "_id": "ilgigantebig",
  "startUrl": [
    "https://www.cosicomodo.it/
        ilgigante/varallo-pombia/pages
        /tutte-le-categorie"
  ],
  "selectors": [
    {
      "id": "reparto",
      "type": "SelectorLink",
      "selector": "ul.list a",
      "parentSelectors": [
        "_root"
      ],
      "multiple": true,
      "delay": 0
    },
    {
      "id": "element",
      "type": "SelectorElement",
      "selector": "div.figcaption",
      "parentSelectors": [
        "reparto"
      ],
      "multiple": true,
      "delay": 0
    },
    {
      "id": "title",
      "type": "SelectorText",
      "selector": "h3.name a",
      "parentSelectors": [
        "element"
      ],
      "multiple": false,
      "regex": "",
      "delay": 0
    },
    {
      "id": "price",
      "type": "SelectorText",
      "selector": "strong.current",
      "parentSelectors": [
        "element"
      ],
      "multiple": false,
      "regex": "",
      "delay": 0
    }
  ]
}
```

Tale sitemap è in grado di estrarre i nomi e i prezzi dei prodotti presenti nella prima pagina di ogni reparto. Non c'è stato bisogno di aggiungere più pagine per reparto perché i prodotti superavano in numero il migliaio, quindi abbiamo ritenuto di avere abbastanza prodotti.

Successivamente con uno script python abbiamo trasformato il prodotto del web scraper in JSON (formato più consono all'utilizzo desiderato), aggiungendo un campo "`discount`" randomizzato tenendo conto del prezzo del prodotto nonché un campo "`availability`" anch'esso con valore casuale.

10 Come eseguire il server

Per eseguire il progetto è sufficiente lanciare lo script `start.sh` presente nella cartella principale del progetto. Questo script fa partire i server e Node-RED, inserisce il dataset in MongoDB ed apre firefox con le pagine principali.

[3] https://chrome.google.com/webstore/detail/web-scraper/jnhgnonknehpejjnehehllkliplmbmhn

Per evitare incompatibilità con Node-RED (che utilizza una versione di Node.js più vecchia), i server sono stati isolati in un container docker che espone solo le porte indicate in precedenza. Per questo, lo script, prima di far partire i server, si occupa di scaricare ed installare Docker se non è già presente sul sistema e di costruire il container. Questa modalità di esecuzione è stata testata sulla macchina virtuale con ubuntu utilizzata a lezione.

Se invece si desidera eseguire il progetto su un altro supporto è possibile provare ad eseguire lo script aggiungendo l'argomento `--no-docker`. Perché tale modalità funzioni correttamente è necessario che sul sistema sia installata l'ultima versione di Node.js.