Smart Dumpster

Alberto Marfoglia, Elia Marcantognini, Tommaso Mandoloni

Indice

1 Hardware

1.1 Schema del circuito

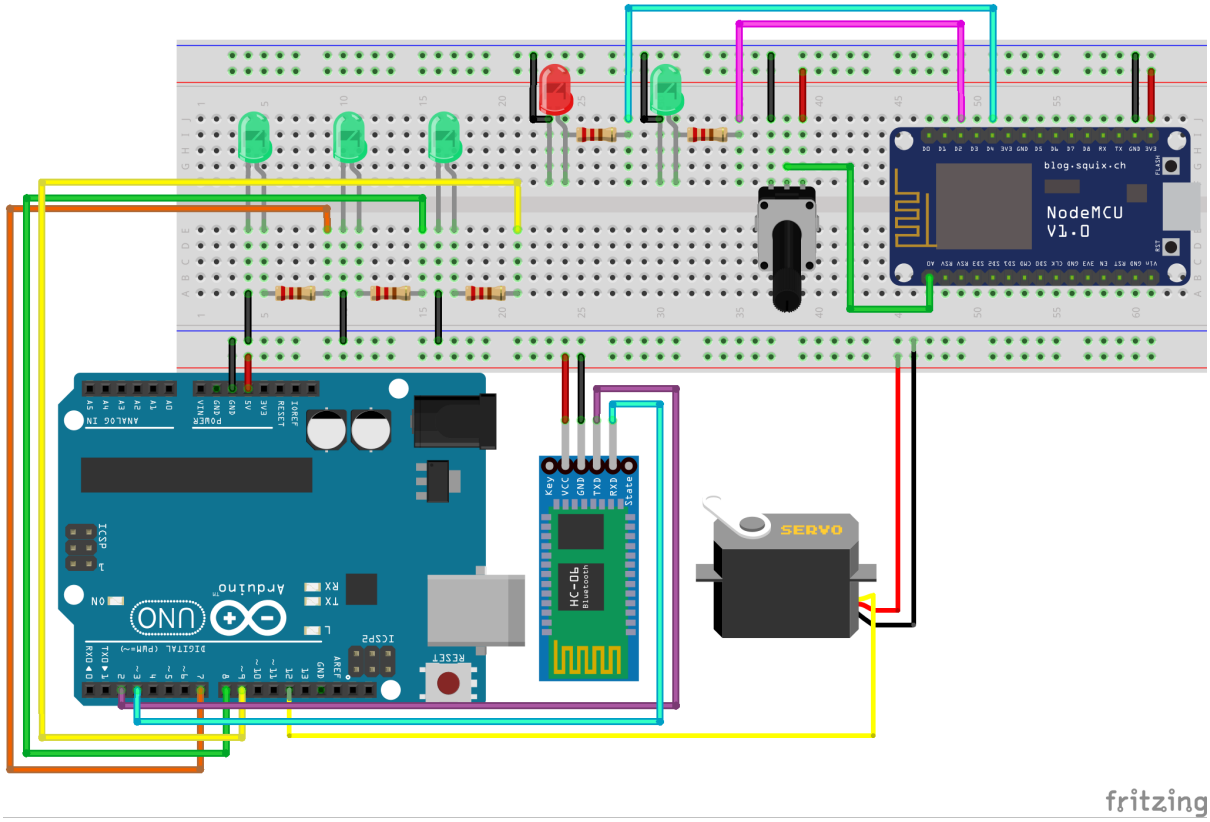

Figura 1: Schema del circuito completo realizzato mediante Fritzing.

1.2 Lista dei sensori e attuatori utilizzati

1.2.1 Arduino Uno r3 e Bluetooth HC-06 - Controller

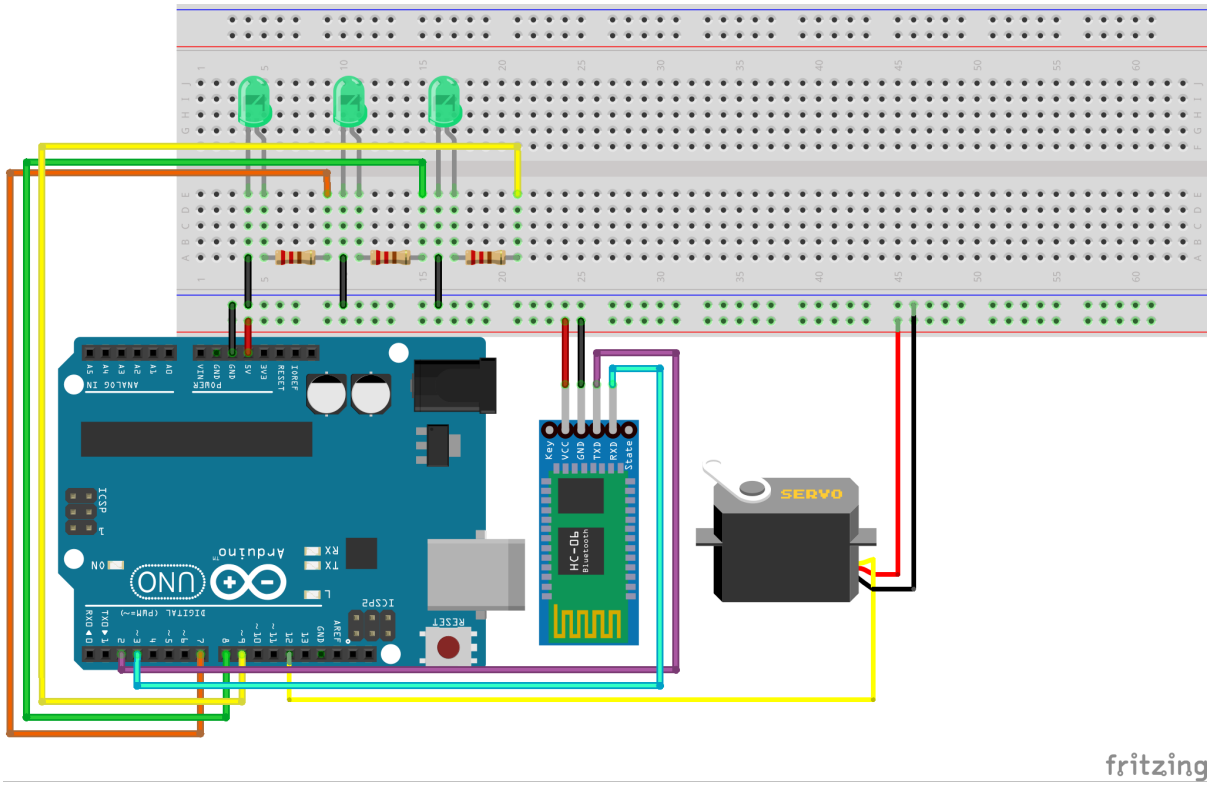

Figura 2: Schema del circuito del controller realizzato mediante Fritzing.

- Servo motore: funzione esclusivamente meccanica, permette di aprire e chiudere il cassonetto;
- Led: utilizzati per segnalare l'apertura del cassonetto e il tipo di rifiuto che l'utente vuole buttare;
- Bluetooth HC-06: utilizzato per implementare la comunicazione tra l'applicazione Android e il Controller.

1.2.2 ESP8266 - Edge

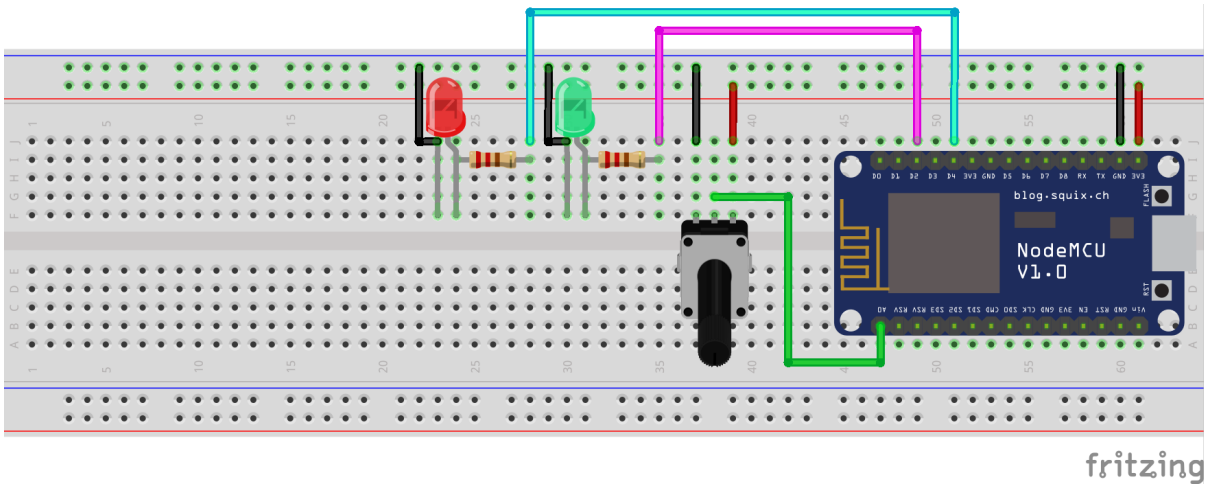

Figura 3: Schema del circuito dell'edge realizzato mediante Fritzing.

- Led: utilizzati per segnalare lo stato del cassonetto;
- Potenziometro: utilizzato per simulare il peso corrente del cassonetto.

2 Software

Tutti i componenti che comunicano con il server utilizzano il protocollo HTTP. Per non avere problemi di compatibilità si è scelto JSON come formato dati. JSON oltre ad essere facilmente leggibile è un **media type** ufficiale di Internet (application/json). Il Service memorizza lo stato del sistema su file con estensione JSON, in questo modo si evitano errori durante la scrittura e la lettura.

2.1 Service

L'applicazione Service, implementata mediante l'ecosistema Node.js, contiene la logica e quindi lo stato dell'intero sistema Smart Dumpster. Il Service offre un'interfaccia REST (API) utilizzata dalle varie componenti per aggiornare/ricevere lo stato globale:

- GET: api/v1/token: ritorna una stringa (token) nel caso in cui il Dumpster sia disponibile;
- GET: api/v1/status: ritorna lo stato del sistema in formato JSON:

  ```
  { available: boolean, quantity: numeric }
  ```

- PUT: api/v1/status: aggiorna lo stato del sistema sostituendolo, una volta validato, con l'oggetto JSON ricevuto all'intero del "corpo" della richiesta;
- GET: api/v1/deposits: ritorna un vettore contenente tutti i depositi raggruppati per data. È possibile specificare specificare una **query** per limitare il numero di depositi:

  ```
  [{ date: boolean, deposits: [quantity: numeric, ... ] }, ... ]
  ```

- POST: api/v1/deposits: aggiunge un nuovo deposito utilizzado la quantità ricevuta e la data corrente.

2.1.1 Node.js

Node.js è un ambiente di esecuzione open-source, basato sul modello di programmazione **event-driven** e consente di realizzare applicazioni lato server utilizzando Javascript. Tra i principali benefici:

- Node.js è scalabile poichè le richeste I/O, di natura bloccanti, sono gestite in maniera asincrona;
- Node.js si basa su Javascript quindi è possibile utilizzare lo stesso linguaggio sia per il backend che per il frontend.

Node.js processa tutte le richieste mediante un singolo thread denominato **event-loop**. Quando Node riceve una richiesta da un client inserisce questa all'interno di una event-queue. L'event-loop esegue indefinitivamente e preleva dalla coda le richieste:

- se la richiesta non necessita di chiamate I/O, Node prepara e invia la risposta al client;
- se la richiesta necessita di operazioni bloccanti I/O, Node affida la richiesta ad un **worker-thread**. Quando l'operazione I/O viene completata, il worker-thread prepara una risposta la quale viene inoltrata al client dall'event-loop mediante l'utilizzo di callback.

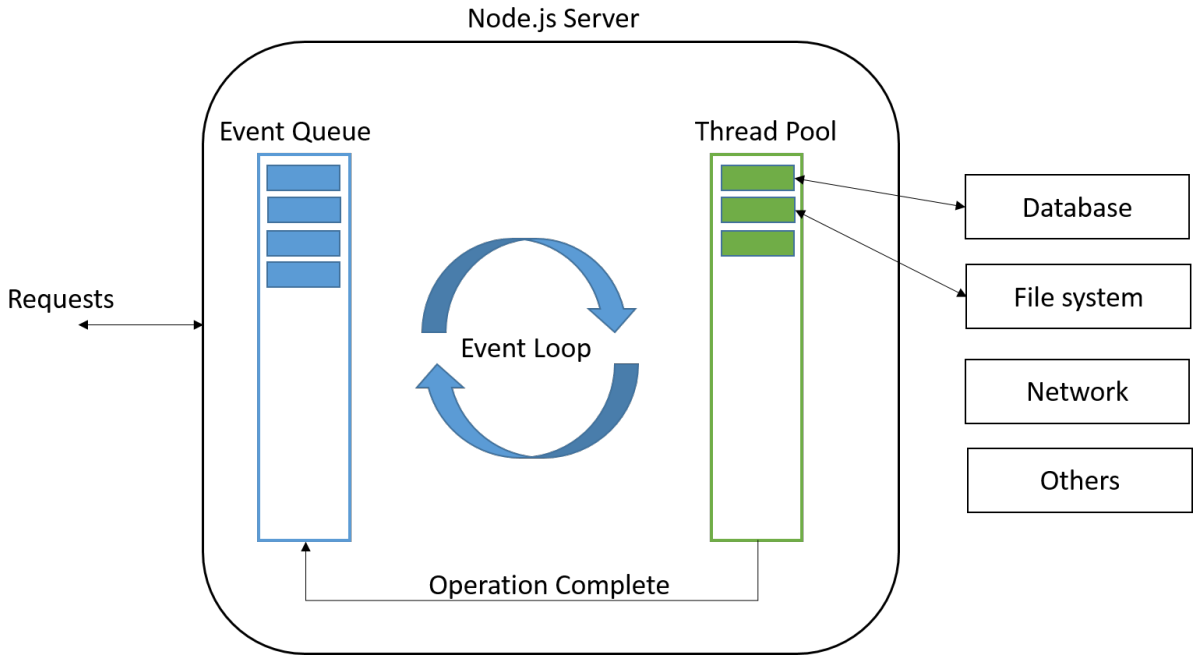

Figura 4: Funzionamento dell'Event Loop su Node.js

2.2 Mobile App

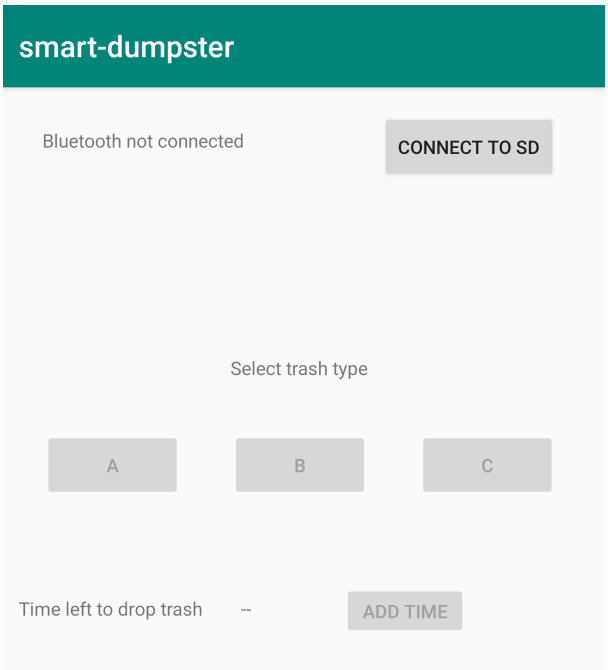

Figura 5: Android Mobile App

L'applicazione mobile è implementata su piattaforma Android ed ha una duplice funzione:

- Comunicazione Bluetooth: attraverso la connessione bluetooth con il controller, ha la possibilità di comunicare quale tipologia di rifiuto l'utente vuole gettare;

- Comunicazione HTTP: attraverso la comunicazione HTTP con il server l'applicazione invia dati inerenti la quantità depositata dall'utente e richiede, per ogni deposito, un token al server.

Il suo funzionamento è semplice, alla richiesta di connessione allo smart dumpster viene fatta una richiesta GET al server e se si riceve un token valido allora viene stabilita la connessione bluetooth allo smart-dumpster. L'utente poi può scegliere il tipo di rifiuto da inserire nel cassonetto e, se il tempo rimanente alla chiusura non è sufficiente, può richiederne un prolungamento. Terminato il deposito viene inviata una richiesta POST al server contenente i dati relativi alla quantità depositata dall'utente. Terminato un deposito l'utente può richiedere al server un ulteriore token per effettuare un altro deposito.
Le richieste GET e POST sono implementate attraverso l'utilizzo di due AsyncTask, rispettivamente GetTokenTask e PostDepositTask, per evitare il rischio di chiamate HTTP bloccanti sul main thread dell'activity.
L'applicazione è quindi così composta:

- MainActivity: classe responsabile del rendering dell'app;
- GetTokenTask: AsyncTask responsabile dell'invio di richieste GET. Attraverso l'utilizzo di un listener è possibile passare il risultato della GET alla MainActivity;
- PostDepositTask: AsyncTask responsabile dell'invio di richieste POST. Invia un body di tipo application/json con la quantita depositata, simulata attraverso una funzione random.

2.3 Controller

Il controller è implementato come macchina a stati finiti sincrona su piattaforma Arduino. È basato fondamentalmente su un unico task, il MainTask, che ha il compito di gestire l'apertura e la chiusura dello smart-dumpster.

2.3.1 MainTask

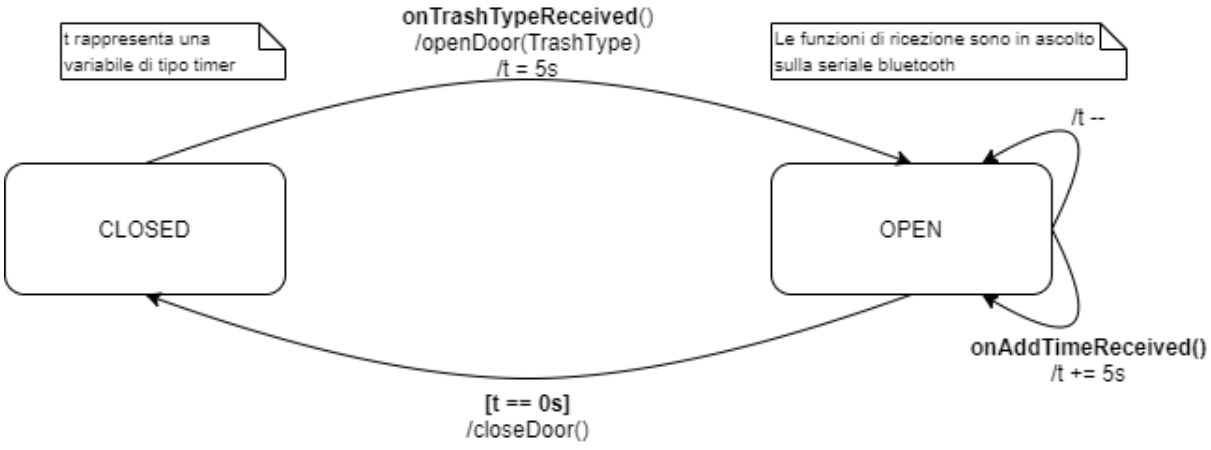

Figura 6: Diagramma a stati finiti del MainTask

Il MainTask, come si può evincere dallo schema, è un semplice task con due stati che agisce su un unico oggetto che è il DumpsterDoor.
Il task parte dallo stato di **close** e quando il controller riceve, tramite bluetooth, il tipo di rifiuto che l'utente vuole depositare passa allo stato open inizializzando un timer di 5 secondi, aprendo

lo sportello e accendendo il led relativo al tipo di rifiuto gettato. Terminato il tempo, che può essere prolungato se il controller riceve un comando di **addTime** dall'app, il task torna allo stato di **closed**.

2.3.2 DumpsterDoor

DumpsterDoor è l'oggetto principale del controller, che rappresenta lo sportello mobile del cassonetto. È composto da un **ServoMotor**, che ha il compito di aprire e chiudere lo sportello, e da un **LedController**, che ha il compito di accendere (e spegnere) il led relativo al tipo di rifiuto ricevuto tramite bluetooth.

2.3.3 BluetoothSerial

BluetoothSerial è l'oggetto predisposto al controllo e alla gestione dei messaggi inviati e ricevuti attraverso il protocollo bluetooth, insieme a MsgService. In particolare, BluetoothSerial, può:

- Inviare il tempo rimanente prima che il portello del cassonetto venga richiuso, funzione utile per far visualizzare nell'applicazione Android il tempo all'utente;
- Ricevere il tipo di rifiuto che l'utente vuole depositare;
- Ricevere il comando di prolungamento del tempo, se l'utente lo necessita.

2.4 Edge

L'edge è stato implementato sfruttando un approcio polling su NodeMCU, piattaforma open-source che include un firmware funzionante tramite il modulo WiFi SoC ESP8266. Ruolo fondamentale di questo componente è verificare e tenere aggiornato lo stato di tutto il sistema tramite interazioni con il Service: infatti è compito suo accendere e spegnere i led che segnalano lo stato globale del Dumpster in base alla quantità di rifiuti depositata, che non deve superare una soglia prestabilita. Il core del sistema è composto da un solo task principale che sfrutta degli oggetti predisposti a semplificare la gestione delle attività che deve svolgere:

- Due oggetti della classe Light, estesione della classe Led, che rappresentano i led di stato del componente stesso, aventi le funzioni di accensione e spegnimento in base allo stato corrente.
- Un oggetto della classe Potentiometer rappresentante il potenziometro per impostare la quantità di rifiuti. Si occupa semplicemente di prelevare il valore attuale del potenziometro o ritornare un valore booleano se quest'ultimo è stato cambiato, settando il nuovo valore aggiornato in caso sia true.
- Un oggetto HTTPClient, fornito dalla libreria ESP8266HTTPClient, che si occupa di effettuare la connessione al server tramite l'indirizzo IP ed eseguire le richieste HTTP di GET e PUT su di esso per ricevere o inviare dati.
- Un oggetto DynamicJsonDocument, fornito dalla libreria ArduinoJson, che permette una gestione semplificata dei documenti in formato JSON sfruttando la serializzazione, facilitando in questo modo lo scambio di messaggi con il server.
- Per quanto riguarda invece la comunicazione WiFi, viene utilizzata la libreria ESP8266WiFi che mette a disposizione dei metodi statici per garantire la corretta connessione alla rete.

Il corretto funzionamento è dettato dal main task che si occupa di interrogare ciclicamente il server ad ogni secondo. Infatti, dopo una prima fase di setup iniziale dove vengono istanziati

gli oggetti, settate le variabili e inizializzata la connessione WiFi, la procedura di svolgimento delle attività segue uno schema ben definito:

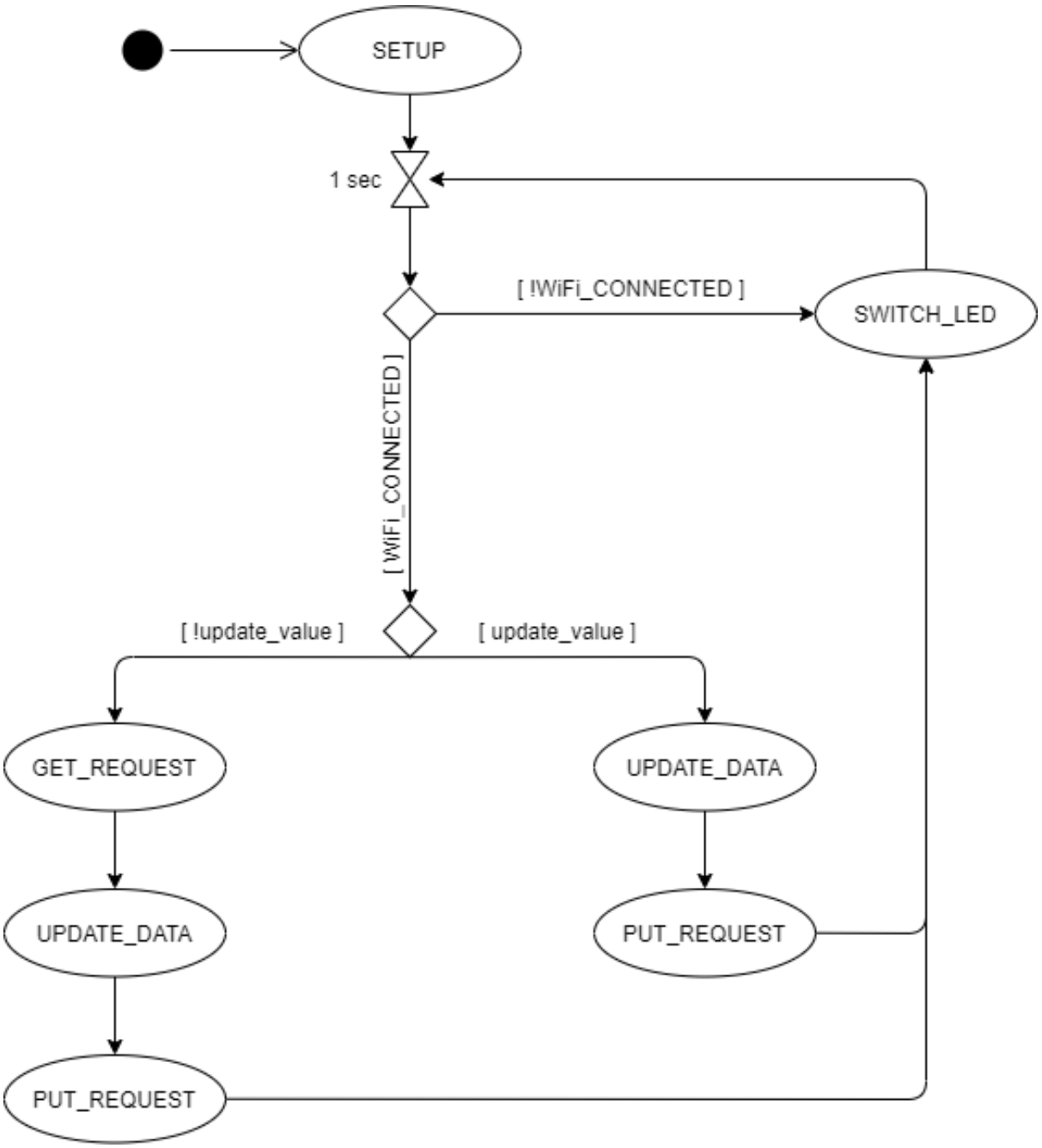

Figura 7: Edge polling procedure

2.5 Dashboard

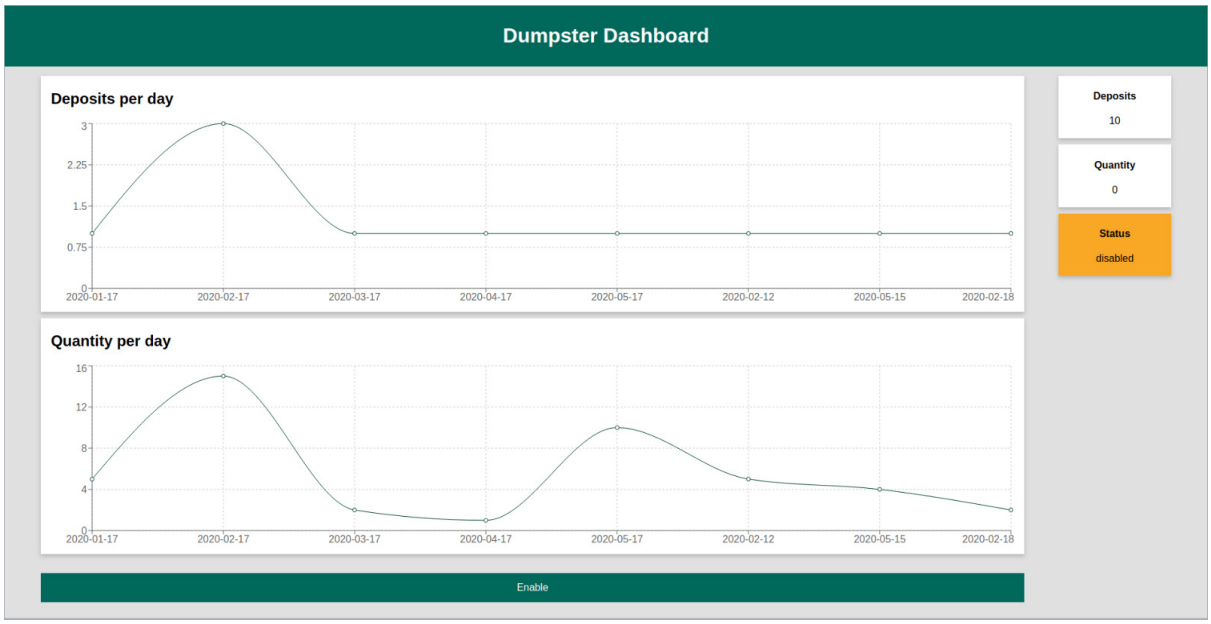

Figura 8: Dashboard - frontend

La dashboard è stata realizzata come un'applicazione **single-page** tramite il framework **React.js**. Grazie React è stato possibile suddividere il codice in componenti (**Component**) in modo da ridurre la complessità del codice. Altri vantaggi del framework possono essere:

- JSX: sintassi estesa di Javascript permette l'inserimento di elementi HTML all'interno di codice Javascript con estrema facilità portando inoltre ad avere un codice più leggibile.
- Virtual DOM: il DOM viene rappresentato virtualmente, tutte le modifiche che vi vengono apportate vengono poi di riflesso applicate anche nel DOM reale. In questo modo si aggiornano specifiche parti risparmiando tempo di calcolo dovuto al rendering dell'intera pagina.
- E' possibile implementare delle componenti con un proprio stato, oppure altre funzionali (stateless) che ricevono le proprietà dalle componenti "genitori". In questo modo la logica dell'intera applicazione può essere centralizzata.

L'applicazione viene compilata e convertita in una versione di Javascript compatibile con la gran parte dei web browser mediante la libreria Webpack. Il bundle compilato viene incluso all'interno di un tag script e servito staticamente da un server Node.js. L'applicazione esegue delle richieste HTTP asincrone. Il responso che se ne ottiene contiene i dati che verranno prima manipolati e successivamente utilizzati per la realizzazione di grafici. Tramite il pulsante mostrato in figura è possibile abilitare e disabilitare il Dumpster in qualsiasi momento a patto che la quantità di spazzatura corrente non superi quella massima.