Politecnico di Torino

Politecnico di Torino

Corso di Laurea in Ingegneria Informatica
Sessione di Laurea Ottobre 2021

Progetto e implementazione di un'architettura innovativa per la gestione e visualizzazione dei dati IoT di una filiera produttiva

Relatori:
Prof. GIOVANNI MALNATI
Prof. FABIO FORNO

Candidato:
SIMONE D'AMILO

A.a. 2020/2021

Sommario

Le aziende al giorno d'oggi si trovano a confrontarsi con molteplici problematiche, tra cui una in particolare è la logistica. Uno dei concetti chiave che ruotano attorno alla logistica di una impresa è quello della tracciabilità, intesa come la possibilità di registrare e identificare un prodotto dalla propria origine fino alla destinazione finale. Il presente lavoro di tesi viene inglobato in un progetto più grande che ha lo scopo di permette a tutte le aziende di una Supply Chain di mettersi in contatto e di scambiare dati in maniera efficiente e veloce. Le informazioni sono quelle relative ad un prodotto e sono fondamentali per ricostruire il suo ciclo di vita. Questa piattaforma prende il nome di XTAP.

La mia attività si concentra sulla possibilità di associare in modo strutturato ai dati di produzione di un prodotto le informazioni raccolte tramite dei sensori ambientali. Questo è il principale vantaggio rispetto alle piattaforme di monitoring IoT già presenti. La mia attenzione si concentra soprattutto sulla visualizzazione dei dati: l'obiettivo è quello di introdurre all'interno di XTAP la possibilità di creare grafici e dashboard specificando i dati da visualizzare. Nel farlo è necessario essere conformi ad EPCIS, uno standard GS1 che descrive come le informazioni di un prodotto devono essere trasferite tra i diversi attori di una Supply Chain. L'unità base di EPCIS è un evento, una struttura contenente 5 dimensioni: What, Where, Why, When e How, l'ultima delle quali introdotta con la versione 2.0 di EPCIS.

Per l'implementazione delle funzionalità richieste si è utilizzato React, una libreria JavaScript per lo sviluppo d'interfacce utente. Si è sviluppato una pagina per la creazione dei grafici, all'interno della quale vengono richieste alcune informazioni come il nome del grafico, la finestra temporale, il tipo di dato e i filtri per meglio identificare i dati da visualizzare. Inoltre, è presente un'altra pagina per la visualizzazione delle dashboard presenti nel sistema, dalla quale è possibile raggiungere la pagina necessaria per la visualizzazione e modifica di una singola dashboard. Per la gestione dei grafici si è utilizzato react-vega, una libreria che attraverso un approccio dichiarativo permette di descrivere un grafico con una grande flessibilità.

Durante il presente lavoro di tesi mi sono occupato anche di alcuni aspetti relativi al back-end, come la possibilità di utilizzare la Real-time Aggregation per migliorare le performance in fase di lettura sul database. Questa funzionalità è messa a disposizione da TimescaleDB, un database basato su PostgreSQL e utilizzato per il salvataggio delle risorse dell'applicazione. La Real-time Aggregation consiste nella creazione di viste materializzate che si aggiornano in modo incrementale in background, inoltre, TimescaleDB gestisce in automatico la lettura dei dati che non sono ancora stati materializzati. Un'altra funzionalità introdotta nel back-end è la gestione delle coordinate spaziali tramite PostGIS, un'estensione di PostgreSQL.

In conclusione, è possibile dire che gli obiettivi prefissati nelle prime fasi del presente lavoro sono stati raggiunti in quanto è stata aggiunta in XTAP una funzionalità grazie alla quale l'utente ha la possibilità di creare dei grafici specificando il tipo di dati da visualizzare. Inoltre, è stata introdotta una gestione delle dashboard che permette all'utente di personalizzare la visualizzazione dei grafici salvati; il tutto rimanendo conformi ad EPCIS.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Le aziende al giorno d'oggi si trovano a confrontarsi con molteplici e differenti problematiche, tra cui una in particolare è la logistica. Più nel dettaglio, uno dei concetti chiave che ruotano attorno alla logistica di una impresa è quello della tracciabilità, intesa come la possibilità di registrare e identificare un prodotto dalla propria origine fino alla destinazione finale. Uno degli ambiti dove la tracciabilità è necessaria, non solo per migliorare la qualità del lavoro ma anche perché imposto da leggi nazionali, è quello agroalimentare. Il tema della sicurezza alimentare è un argomento di grande importanza perché si pone sempre più attenzione alla salute dei cittadini e a quello che mangiano. Infatti in un mercato globalizzato come quello dei nostri giorni è fondamentale conoscere il luogo di provenienza di un prodotto e i vari step che lo hanno condotto alla vendita finale. La tracciabilità dei prodotti è fondamentale per le autorità e gli organi di controllo ma non solo, infatti, sono molti i motivi per i quali è necessario tenere traccia della vita di un prodotto. Tra questi, uno dei più importanti è dare la possibilità al cliente finale di avere quante più informazioni possibili sui prodotti che sta per acquistare in modo tale da poter effettuare la scelta migliore. Ciò porta ad un miglior rapporto tra clienti ed azienda perché aumenta la fiducia del consumatore, il quale può facilmente verificare la sicurezza e la genuinità del prodotto. Inoltre, la tracciabilità svolge un altro ruolo fondamentale, cioè quello di garantire l'originalità di un prodotto ed è importante in quanto è una vera e propria attestazione di qualità rivolta al diminuire la contraffazione. I prodotti Made in Italy soffrono da sempre della concorrenza di Paesi sleali che imitano in tutto e per tutto le nostre eccellenze e le mettono in commercio a prezzi decisamente inferiori.

Implementare un sistema di tracciabilità non è solo una necessità, infatti porta anche dei vantaggi strategici che possono essere racchiusi sotto due grandi macro-categorie:

- Efficienza Logistica, che si ottiene grazie alla conoscenza ed al controllo, in ogni punto della filiera, delle caratteristiche del prodotto e della sua movimentazione. Alcuni vantaggi sono relativi ai rifornimenti, alla collaborazione tra aziende, alla gestione ottimale dei mezzi di trasporto, alla riduzione delle emissioni di CO2 e alla riduzione della documentazione cartacea;
- Marketing, dove si parla di tutte quelle informazioni che possono costituire una customer experience per il consumatore finale oppure di informazioni che aumentano la conoscenza che il consumatore ha dell'azienda.

Un altro concetto fondamentale è quello della rintracciabilità, che è contrario alla tracciabilità, cioè è il processo che torna indietro nella catena di produzione di un

prodotto, al fine di ricercare un preciso evento. Entrambi i concetti seguono la logica di una raccolta ordinata di informazioni durante un processo. Quindi, possiamo vedere la tracciabilità come la comunicazione di ogni singolo evento, ciò consente di conoscere le varie fasi di trasformazione, mentre la rintracciabilità è l'archiviazione di queste informazioni per poterle riprendere in un qualsiasi momento.

Il problema dello stato attuale è che i servizi esistenti non sempre permettono di gestire tutte le informazioni in modo adeguato. Per questo motivo, in alcuni casi si preferisce gestire manualmente il ciclo di vita di un prodotto utilizzando dei documenti cartacei, quindi la comunicazione tra le diverse aziende diventa macchinosa e difficile da gestire. Infatti uno degli aspetti fondamentali è la comunicazione delle informazioni tra le aziende della Supply Chain perché in questo modo ogni nodo della catena ha la possibilità di controllare che il prodotto sia gestito in maniera adeguata in tutti gli altri step della sua realizzazione. Non avere una piattaforma completa per la gestione della tracciabilità comporta un costo da parte delle aziende che non sempre è accettabile. Ad esempio non si da la possibilità alle aziende di trovare in anticipo eventuali errori che possono causare il danneggiamento di un particolare prodotto alimentare comportando una grave perdita dal punto di vista economico, non solo non potendo più vendere il prodotto danneggiato, ma parliamo anche di danni di reputazione e del costo necessario per il recupero integrale. Ciò porta ad una difficile ricerca per individuare il lotto da ritirare e soprattutto ad uno studio approfondito per capire le cause del problema.

Uno degli aspetti da considerare quando si vuole ricostruire la 'storia' di un prodotto è quello del monitoraggio delle informazioni relative ad esso attraverso dei dispositivi IoT. Per meglio spiegare questo approccio è necessario introdurre brevemente il concetto di IoT, con il quale si indicano tutti quei dispositivi che appartengono al mondo digitale. Non si parla solo di computer o smartphone, ma anche di tutti gli oggetti che circondano la vita di tutti i giorni. Al giorno d'oggi questi dispositivi permettono di semplificare quelle azioni che fino a pochi anni fa erano complesse e necessitavano dell'intervento umano. E' elevato il numero degli ambiti nei quali i dispositivi IoT sono utilizzati: dalla casa intelligente alla smart factory, dalle auto intelligenti alla smart city e via dicendo.

Entrando nel dettaglio del presente lavoro di tesi, l'obiettivo è quello di includere in una piattaforma già esistente la raccolta tramite dei dispositivi di IoT delle informazioni relative ai prodotti, lo stoccaggio dei dati raccolti e la loro visualizzazione attraverso delle dashboard che potranno essere costruite in base alle esigenze dell'azienda. In particolar modo, la piattaforma sopra citata permette a più aziende di mettersi in comunicazione e di scambiare dati in maniera efficiente e veloce, risolvendo i problemi descritti in precedenza. Inoltre, fornisce degli strumenti molto utili per monitorare ed analizzare la catena di produzione dei prodotti integrando dati provenienti da diverse aziende. Il nome della suddetta piattaforma è XTAP.

Il monitoraggio dei prodotti, come descritto in precedenza, è una delle operazioni principali che devono essere svolte per poter dare una visione completa di quello che è il percorso di un prodotto. Ogni volta che parliamo di monitoraggio facciamo riferimento a tre step che dall'esterno vengono visti come un'unica operazione ma in realtà sono fasi complesse e che necessitano di uno studio approfondito individuale:

- *Raccolta*, in questa fase i protagonisti sono i dispositivi IoT che hanno come scopo quello di raccogliere le informazioni riguardo ad uno specifico luogo o prodotto. Le informazioni posso essere di diverso tipo, alcuni esempi sono temperatura, umidità, illuminazione, velocità del vento, precipitazioni, ecc.

- *Aggregazione e Stoccaggio*, una volta raccolti i dati è necessario processarli ed in particolar modo aggregarli prima di salvarli. Questa operazione è fondamentale

in quanto la quantità di informazioni ricevute dai dispositivi è notevole, quindi si preferisce raggrupparli per evitare di raggiungere delle quantità di dati non gestibili.

- *Visualizzazione*, l'ultima operazione fondamentale è quella della visualizzazione dei dati. L'obiettivo è quello di mettere a disposizione degli utenti della piattaforma un modo semplice e veloce per consultare i dati necessari attraverso dei grafici e delle dashboard.

Da una prima descrizione di queste fasi è possibile capire come il lavoro di ognuna di esse sia rivolto verso delle operazioni specifiche che sono essenziali per il raggiungimento di quello che è l'obiettivo principale: permettere agli utenti della piattaforma di avere una visione d'insieme su quelle che sono le caratteristiche più importanti da considerare quando si vuole creare un prodotto di qualità. All'interno di questo documento si cercherà di entrare più nel dettaglio di queste fasi analizzando quali sono gli strumenti utilizzati e perché si è scelto di utilizzarli.

Capitolo 2

Stato dell'Arte

Nel capitolo precedente è stato introdotto quello che è lo scopo principale del lavoro descritto in questo documento: raccogliere i dati relativi ad un prodotto od un luogo attraverso dei dispositivi di IoT, permettere la loro visualizzazione all'utente e includere questo meccanismo all'interno di XTAP. Per sviluppare questa funzionalità all'interno della piattaforma XTAP ci sono diverse possibilità, in particolar modo si potrebbe sfruttare uno dei tanti servizi disponibili online. Al giorno d'oggi sono tanti i progetti che hanno come obiettivo quello di creare una piattaforma in grado di aiutare gli utenti a svolgere tutte le operazioni del monitoring, che come descritto nel capitolo precedente sono Raccolta, Aggregazione e Stoccaggio, ed infine la Visualizzazione.

Questa abbondanza di piattaforme che gestiscono il monitoring di un'azienda, a partire dalla configurazione dei sensori per arrivare alla visualizzazione dei dati raccolti, è dovuta alla sempre più crescente necessità delle aziende di tenere sotto controllo i propri prodotti e i luoghi di lavoro. Infatti, il monitoraggio IoT permette ad un'azienda di essere sempre pronta ad intervenire per risolvere eventuali problemi che si manifestano in uno dei tanti step nei quali un prodotto necessita di passare. Vediamo alcuni esempi di queste piattaforme per poterne valutare le caratteristiche principali, le differenze ed eventualmente i problemi:

- *eMoldino*, questa piattaforma gestisce tutte le fasi del monitoring e rende tutte le operazioni pressoché automatiche. Partendo dalla raccolta dei dati, l'utente ha il solo compito di posizionare i sensori ed eMoldino si occupa della raccolta e della trasmissione dei dati verso un server, il tutto senza nessun intervento da parte di un umano. Inoltre, la trasmissione dei dati viene protetta grazie ad un meccanismo di cifratura che garantisce la confidenzialità delle informazioni. eMoldino mette anche a disposizione un software facile da utilizzare per visualizzare efficientemente i dati raccolti, e permette ad ogni azienda di personalizzarlo in modo tale da adattarlo alle proprie esigenze. Infine, l'ultima caratteristica di eMoldino è un'analisi dei dati effettuata tramite l'Intelligenza Artificiale, che aiuta le aziende a prendere delle decisioni in modo corretto e sistematico;

- *ThingsBoard*, le caratteristiche principali sono molto simili a quelle di eMoldino, ma la particolarità di ThingsBoard è la possibilità di definire delle catene di regole, che se non rispettate fanno partire una catena di allarmi. Quindi, questo strumento viene utilizzato per controllare continuamente alcuni aspetti che sono di particolare importanza per l'azienda.

Altre piattaforme con caratteristiche simili a quelle precedentemente elencate sono *TheThings*, *WolkAbout* e *KaaIot*. Come è possibile notare sono molte le piattaforme

che aiutano le aziende a costruire in modo semplice ed intuitivo dei sistemi in grado di gestire tutte le fasi del monitoring. Il problema di queste piattaforme non è relativo al come le operazioni del monitoring vengono eseguite, infatti tutti i servizi precedentemente descritti sono ottimi da questo punto di vista. Il problema è che il nostro obiettivo non è solo quello di raccogliere questi dati, ma anche di metterli in relazione con altri dati relativi allo stesso prodotto. Il modo in cui è possibile farlo è descritto da GS1, quindi l'obiettivo è quello di creare una piattaforma conforme a GS1 ed in particolare ad EPCIS, uno standard che descrive come le informazioni relative ad un prodotto e ad un particolare step devono essere trasportate. Infatti, la struttura dei dati raccolti deve essere tale da poterla mettere in relazione con i dati di un prodotto che derivano dagli eventi EPCIS, che come vedremo nel capitolo successivo hanno lo scopo di trasportare le informazioni relative ad un prodotto e ad una particolare azione.

Capitolo 3

GS1 e XTAP

3.1 GS1

GS1 è un'organizzazione non-profit che sviluppa e mantiene standard globali per la comunicazione tra imprese. Uno dei più importanti è il codice a barre, un simbolo presente sui prodotti che può essere scannerizzato elettronicamente, rendendo più semplice la possibilità di tracciare, processare e stoccare un prodotto. I codici a barre hanno un ruolo cruciale nella distribuzione poiché velocizzano il check-out, conducono ad una gestione dell'inventario e della consegna più efficienti e permettono la vendita online su scala globale.

Lo scopo di GS1 è quello di migliorare l'efficienza, la sicurezza e la visibilità delle Supply Chain, attraverso strumenti fisici e digitali. Inoltre, si pone l'obiettivo di aiutare le aziende riducendo i costi necessari per mantenere la tracciabilità dei prodotti grazie ad un sistema automatizzato basato su chiavi univoche globali e informazioni digitali. GS1 è progettato per essere usato in ogni tipo di azienda, infatti è possibile trovarlo nel settore del largo consumo, in quello sanitario ma anche nei trasporti e logistica, ristorazione e settori tecnici. Tutte le modifiche che vengono effettuate al sistema sono introdotte in modo tale da non creare problemi alle applicazioni che utilizzano le versioni precedenti.

L'architettura GS1 è composta da tre livelli, ognuno dei quali con uno scopo ben preciso: Identificazione, Cattura e Condivisione. Introduciamoli brevemente facendo riferimento alla figura 3.1:

- Il primo strato rappresenta tutti gli oggetti e soggetti di una filiera, dove ogni elemento viene classificato da una chiave numerica di identificazione, come ad esempio il Global Trade Item Number - GTIN, che identifica un prodotto oppure il Global Location Number - GLN, che identifica un luogo;
- Nel secondo strato abbiamo le diverse tecnologie di codici a barre e RFID che possono essere utilizzati per trasportare in modo automatico le informazioni viste nello strato precedente, dati che possono essere conformi allo standard GS1 oppure definiti dal cliente;
- L'ultimo strato contiene gli standard legati alla condivisione dei dati tra più aziende, tra cui EPCIS, quindi il loro obiettivo è quello di definire come le informazioni devono essere diffuse in modo tale da renderle comprensibili a tutti.

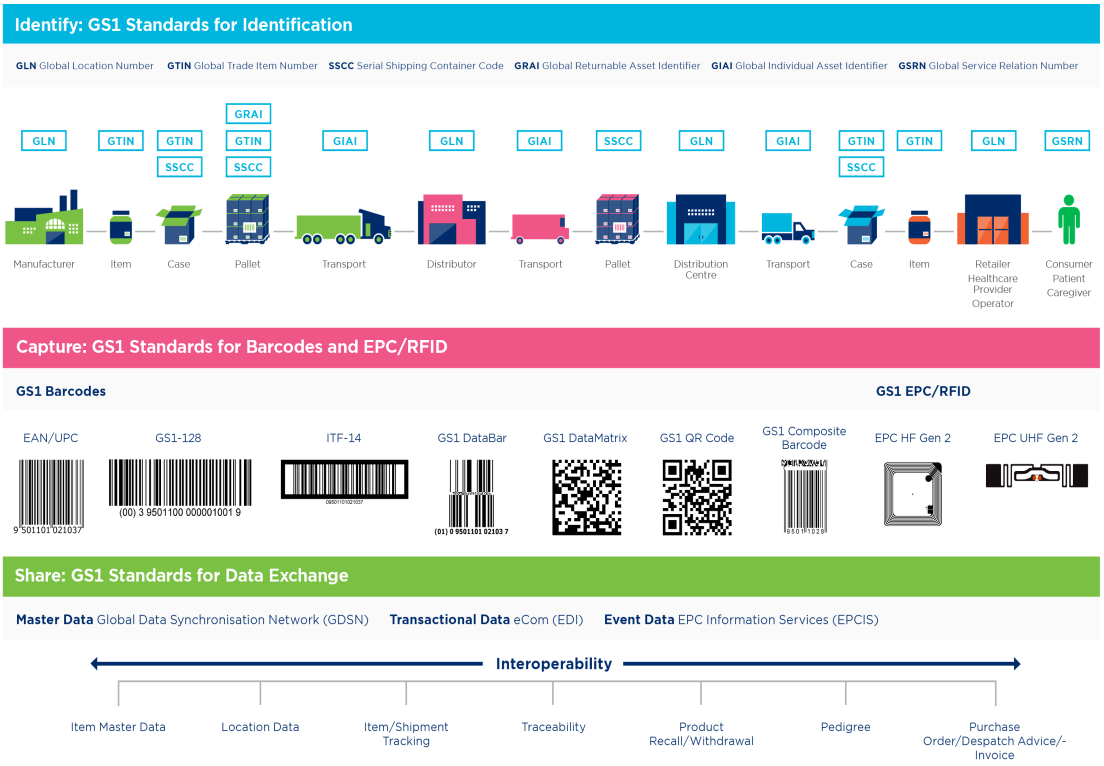


Figura 3.1: Architettura GS1

3.1.1 EPCIS

EPCIS (Electronic Product Code Information Services) è uno standard open GS1 ma anche approvato come ISO/IEC 19987 e consente di condividere tutte le informazioni di un oggetto, dal produttore al consumatore finale. Nel contesto di EPCIS un oggetto è un'entità fisica o digitale gestita all'interno di un processo commerciale che coinvolge una o più aziende. Alcuni esempi di oggetti fisici possono essere un prodotto, un'unità logistica o un documento fisico, mentre esempi di oggetti digitali sono i libri elettronici o i coupons elettronici. EPCIS è uno standard di condivisione dei dati che acquisisce le informazioni a partire da Tag EPC/RFID o Codici a barre, che si basano sull'uso dell'Application Identifier, cioè dei marker che definiscono il significato delle cifre del Codice a barre stesso.

Nella figura 3.2 è presente una rappresentazione di una filiera generica dal Supplier al Retail Store. Durante la sua fase di realizzazione, un oggetto attraversa tutti i nodi dello schema (Supplier, Warehouse, Manufacturer, Distribution Centre e Retail Store) compiendo diversi passaggi. Il dovere di ognuno degli attori della catena è quello di raccogliere le informazioni riguardanti il prodotto sia per utilizzarle internamente che per condividerle con gli altri partner. Quindi possiamo dire che EPCIS ha lo scopo di definire la struttura di queste informazioni e come devono essere distribuite tra le varie aziende di una Supply Chain.

L'unità base in EPCIS è una struttura che contiene tutte le informazioni di un prodotto in un particolare step della Supply Chain; questa struttura prende il nome di Evento EPCIS. Tutti gli eventi hanno in comune il fatto di essere organizzati secondo 5 dimensioni: What, Where, Why, When e How, l'ultima delle quali introdotta con la versione 2.0 di EPCIS, che come si vedrà all'interno di questo documento introduce la possibilità di gestione di dispositivi IoT.

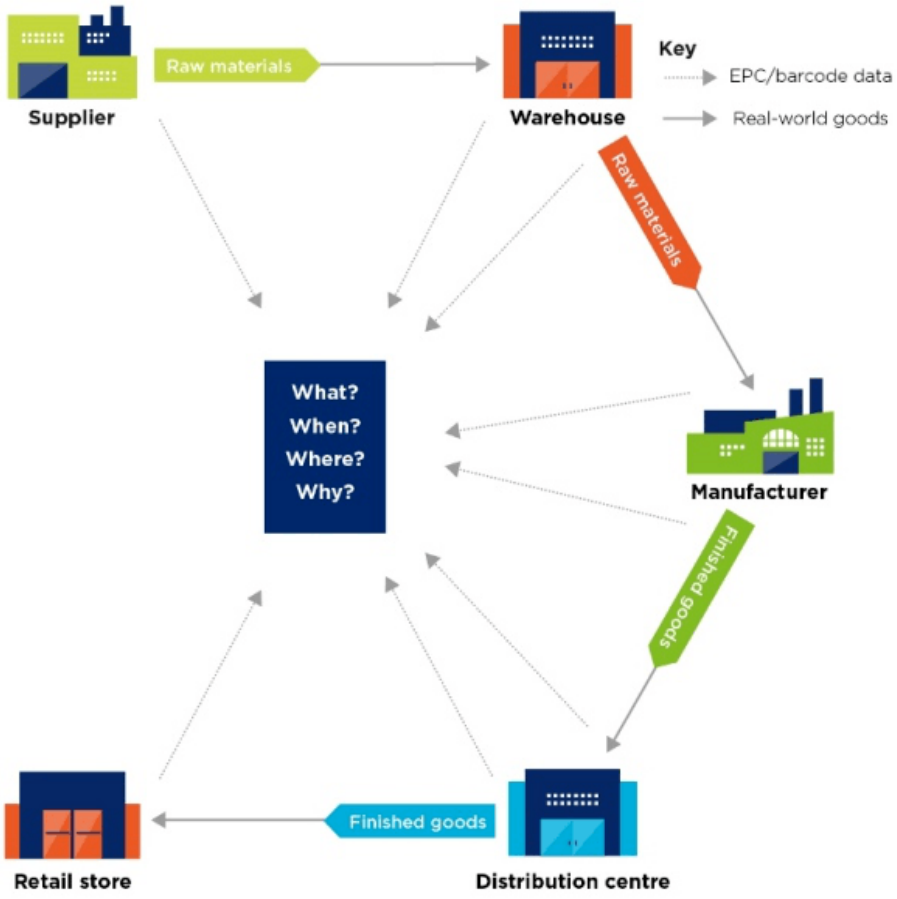


Figura 3.2: EPCIS

Gli eventi EPCIS hanno un campo 'tipo' che definisce ciò che rappresenta un evento. Esistono cinque tipi di eventi, uno generico e 4 sottoclassi:

- *EPCIS Event* è una classe generica per tutti i tipi di evento;
- *Object Event* è un evento che rappresenta ciò che è accaduto ad uno o più oggetti fisici o digitali;
- *Aggregation Event* è un evento che rappresenta ciò che è successo ad uno o più oggetti che sono insieme fisicamente, un esempio pratico è un pallet;
- *Transaction Event* è un evento che rappresenta ciò che è successo ad uno o più oggetti che sono associati tramite un identificativo di una transazione;
- *Trasformation Event* è un evento che rappresenta un oggetto in input che viene totalmente o parzialmente consumato per generare un output. Molto simile a quello che succede con l'Aggregation Event, ma la differenza è che in questo caso gli oggetti non sono più suddivisibili.

EPCIS definisce alcune Interfacce per la richiesta e la cattura di informazioni relative ad un prodotto che si sposta lungo una filiera e un Modello di Dati comune utilizzando XML per garantire la visibilità. Per meglio capire il concetto di Interfaccia possiamo

guardare la figura 3.3. Come è possibile notare esistono due tipi di interfacce: le Capture Interface e le Query Interface. Le prime fanno da ponte tra i livelli Cattura e Condivisione, mentre le seconde hanno lo scopo di dare visibilità dei dati sia alle applicazioni interne che ai partner esterni. Si è deciso di creare più livelli e di utilizzare le interfacce per fornire un isolamento tra i diversi livelli di astrazione, infatti le interfacce di EPCIS sono indipendenti dal modo in cui i dati sono catturati e quindi i livelli superiori non hanno idea di come la cattura venga effettuata.

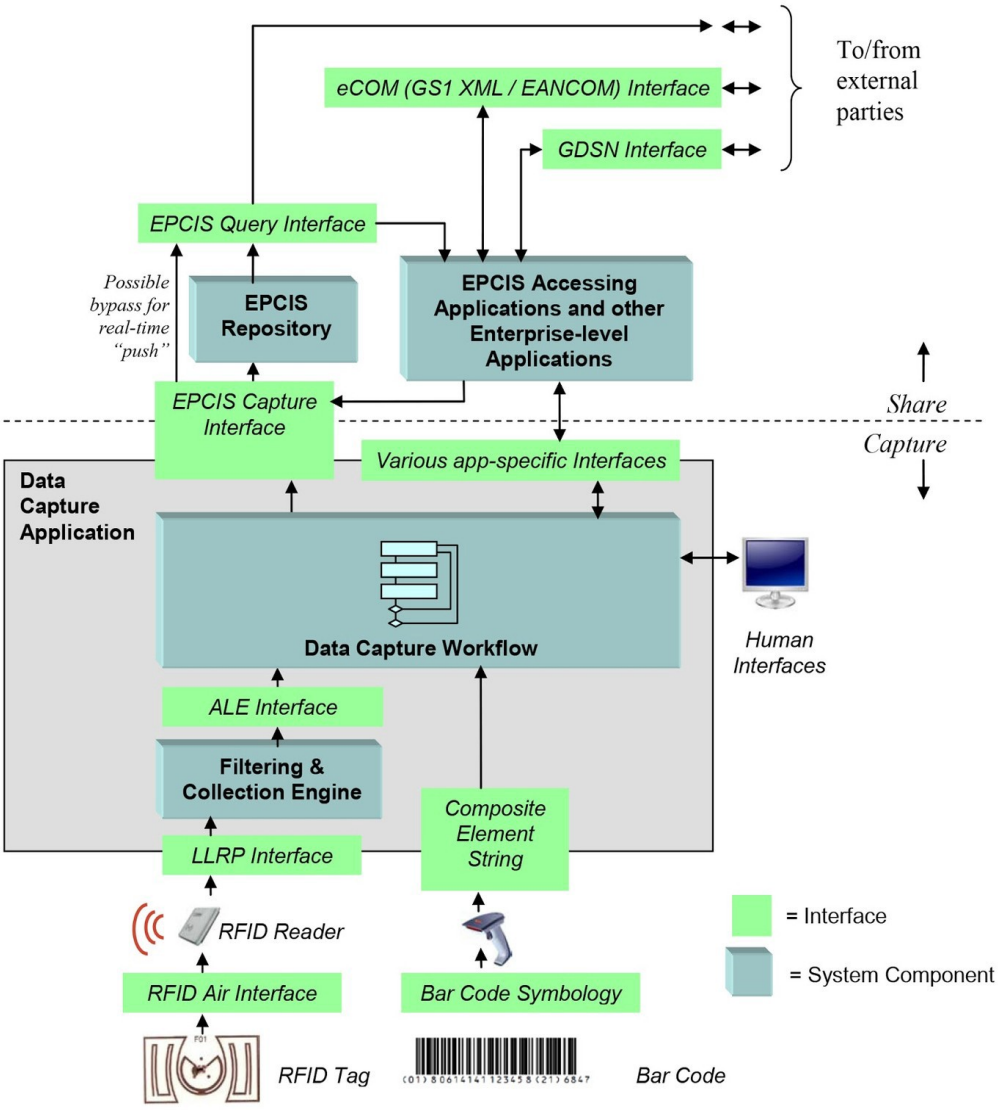


Figura 3.3: Relazione tra EPCIS e gli altri standard GS1

Come accennato in precedenza, tutte le informazioni di un evento sono raggruppate all'interno delle 5 dimensioni, l'ultima delle quali è stata aggiunta con EPCIS2.0, che verrà introdotto alla fine di questo capitolo. Adesso, analizziamo le dimensioni una alla volta per meglio capire il loro scopo e la loro struttura.

Il **What** definisce quale oggetto partecipa all'evento e possiamo classificarlo in due modi differenti: a livello di istanza, dove ogni oggetto ha un identificativo unico, ad esempio l'SGTIN, cioè un GTIN accompagnato da un seriale per renderlo unico. Oppure parliamo di classificazione a livello di classe, dove più oggetti possono avere lo

stesso identificativo, come ad esempio l'LGTIN. Gli attributi della dimensione What sono:

- *EPC* è un attributo a livello di istanza. All'interno di un evento EPCIS il suo valore è sempre un URI che identifica in modo univoco un particolare oggetto;
- *Quantity List* è un attributo a livello di classe ed è composto da tre campi: epcClass, definisce la classe in questione e il suo valore è preso da un dizionario che è definito dall'utente. Il secondo campo è quantity, un numero che definisce la quantità di oggetti all'interno del Quantity List, ed infine uom, cioè l'unità di misura della quantità specificata.

Il **Where** definisce quattro tipi che ci identificano il luogo in cui le informazioni sono catturate. In particolar modo due di questi non vengono considerati come tipi all'interno di EPCIS e si riferiscono al lettore, cioè il device che effettua la rilevazione. I due tipi in questione sono *PhysicalReaderID* che si riferisce alla sorgente fisica che legge i dati e li inserisce all'interno di un evento EPCIS, mentre il secondo tipo è il *LogicalReaderID* che si riferisce ad una sorgente logica che non corrisponde con quella fisica. E' conveniente creare una sorgente logica per diversi motivi, tra cui la possibilità di cambiare il lettore fisico senza modificare nulla all'interno delle applicazioni di cattura di EPCIS, unire più sorgenti fisiche al di sotto di un nome quando rilevano informazioni relative ad un unico luogo oppure dividere logicamente un lettore che ha la possibilità di leggere informazioni da diversi luoghi. Invece, i due tipi di Where considerati all'interno di EPCIS sono: *ReadPointID* e *BusinessLocationID*. Il primo indica la posizione specifica in cui si trova un oggetto in un determinato evento EPCIS, mentre il secondo indica la posizione dell'oggetto fino a quando un altro evento non comunica qualcosa di diverso. Nello specifico possiamo dire che il ReadPointID è una misura più precisa della posizione di un oggetto rispetto al BusinessLocationID.

La dimensione **Why** definisce il tipo di operazione che ha scatenato la creazione dell'evento EPCIS. Gli attributi più importanti della dimensione Why sono:

- *Business Step*, definisce lo step del processo commerciale all'interno del quale l'oggetto si trova ed è l'unico attributo obbligatorio della dimensione Why. Il Business Step è definito con dei verbi che sono elencati all'interno del CBV - Core Business Vocabulary, come ad esempio commissioning, shipping, packing, ecc. Il CVB è un dizionario che contiene tutti i dati che possono essere utilizzati per popolare le strutture definite in EPCIS;
- *Disposition* e *Persistent Disposition*, il primo indica la condizione commerciale di un oggetto, ad esempio in_progress, recalled, damaged, ecc. Questo campo rimane valido fino a quando un nuovo evento non ne indica il suo cambiamento, quindi gli eventi che non specificano nessun Disposition non hanno alcun effetto sul suo vecchio valore. Il secondo, Persistent Disposition, indica una o più condizioni commerciali di un oggetto. Questi valori possono essere impostati (set) o disinseriti (unset) e sono indipendenti l'uno con l'altro;
- *Business Transaction*, indica una particolare transazione commerciale. Questo attributo è composto da due campi: il tipo è un identificatore che indica il tipo di transazione effettuata e i possibili valori sono specificati ancora una volta all'interno del CBV, mentre il bizTransaction è un ID che definisce la particolare transazione;

- *Source* e *Destination*, questi due campi sono specificati quando l'evento si trova all'interno di un trasferimento commerciale. Anche in questo caso l'attributo è definito da un tipo e da un ID che determina la specifica sorgente e la destinazione.

La dimensione **When** contiene le informazioni relative al tempo di un evento. Questa dimensione è composta da tre attributi:

- *EventTime*, la data e l'ora in cui l'applicazione di cattura rileva un evento;
- *RecordTime*, la data e l'ora in cui l'evento è stato registrato all'interno della Repository EPCIS, quindi non descrive nulla rispetto all'evento reale. Questo attributo è opzionale;
- *EventTimeZoneOffset*, cioè l'offset del fuso orario presente nel luogo in cui è stata effettuata la misurazione, relativamente a UTC. Il tipo dell'attributo è una stringa nel formato '+HH:MM'.

3.1.2 EPCIS2.0

L'ultima dimensione da analizzare è How, ma prima di farlo è necessario introdurre la nuova versione di EPCIS, la 2.0. Il cambiamento più importante da notare rispetto alla versione precedente è la possibilità di gestire i sensori necessari per il monitoraggio degli oggetti, non a caso la dimensione How ci mette a disposizione una struttura per poter definire in maniera precisa quale sensore ha effettuato una specifica rilevazione. Le funzionalità introdotte per migliore EPCIS sono:

- Supporto per i dati catturati dai sensori;
- Aggiunta delle informazioni sulle certificazioni dei partner;
- Ottimizzazione del CBV;
- Aggiunta della sintassi JSON e JSON-LD;
- Aggiunta di REST;
- Altre piccole modifiche relative a EPCIS e CBV.

A questo punto siamo pronti per introdurre l'ultima dimensione, **How**. L'attributo essenziale di questa dimensione è il *SensorElement*, una struttura con all'interno due elementi: SensorMetaData e SensorReport. Il primo è un elemento opzionale contenente degli attributi che sono comuni a tutti gli elementi SensorReport di uno stesso SensorElement, il secondo contiene gli attributi che sono specifici di un'osservazione di un sensore. Alcuni attributi del SensorMetaData sono time, start time, end time, device ID, ecc. Mentre esempi di attributi del SensorReport sono type, rawdata, stringValue, booleanValue, hexBinaryValue, ecc.

Entrando nello specifico del presente lavoro di tesi, l'obiettivo è quello di introdurre all'interno di XTAP, già conforme ad EPCIS, la gestione dei sensori necessari per la raccolta dei dati. Quindi, lo scopo è quello di rendere XTAP conforme ad EPCIS2.0. Il mio lavoro di tesi è stato portato avanti insieme a due colleghi che si sono occupati principalmente della gestione dei sensori e del back-end necessario a raccogliere i dati catturati dai dispositivi. Invece, per quanto riguarda la mia attività, è specialmente incentrata sul front-end, dove l'obiettivo principale è quello di dare all'utente finale la possibilità di visualizzare ed analizzare i dati raccolti secondo le proprie esigenze.

3.2 XTAP

3.2.1 Architettura

Lo scopo di XTAP è quello di creare una piattaforma che dia a tutti gli attori della Supply Chain la possibilità di comunicare tra loro. Infatti, per un'azienda non è sufficiente raccogliere i dati di un prodotto, ma è necessario anche poter avere una visione parziale o completa dei dati raccolti dalle altre aziende. Tutto questo non è possibile se le informazioni vengono salvate e gestite localmente dalle aziende, quindi è importante avere uno strumento che aiuti a condividere questi dati. XTAP si propone di risolvere questo problema raccogliendo tutti i dati dei partner all'interno del Cloud, inoltre, ogni azienda può decidere con chi e con quale livello di granularità condividere le proprie informazioni. XTAP ha tra i suoi obiettivi quello di rendere invisibile la complessità tecnologica e degli eventi standardizzati da GS1, in modo tale da permettere l'utilizzo della piattaforma anche a consulenti o figure aziendali non esperte in materia.

L'architettura di XTAP è composta da una serie di repository private, una per ogni partner, all'interno delle quali è possibile salvare gli eventi prodotti dall'azienda stessa oppure quelli ai quali ha il diritto di accesso. Inoltre, è presente una repository di filiera gestita direttamente da XTAP e una repository destinata alle applicazioni dei consumatori finali. I dati sono raccolti dalle aziende tramite diversi strumenti (dispositivi IoT, lettori RFID, form nelle interfacce web, ecc) e vengono inviati alla Capture Interface EPCIS della piattaforma, e a questo punto in base alle regole configurate dai partner i dati sono inoltrati nelle opportune repository. Infatti, ogni attore della filiera può configurare delle regole in base alle quali gli eventi vengono distribuiti e resi visibili agli altri partner della filiera. Per questa operazione si è deciso di utilizzare Faust, una libreria per Python con la quale è possibile processare uno stream di dati, a loro volta gestiti da Kafka.

3.2.2 Tecnologie

Apache Kafka è una piattaforma distribuita open source per il data streaming sviluppata dalla Apache Software Foundation e scritta in Scala e Java. Il progetto ha lo scopo di permettere la pubblicazione, sottoscrizione, archiviazione ed elaborazione di flussi di dati in tempo reale provenienti da più fonti e distribuiti a più consumatori. Kafka si basa sul protocollo TCP per migliorarne l'efficienza e l'affidabilità e permette agli utenti di pubblicare i dati su una coda o di sottoscriversi per ricevere un particolare dato in real time. Inoltre, essendo un sistema distribuito, è composto da più client e server:

- *Server*, Kafka è eseguito come un cluster di uno o più server, alcuni dei quali hanno lo scopo di salvataggio e sono chiamati *Brokers*. Altri eseguono *Kafka Connect* che ha lo scopo di importare ed esportare dati da sorgenti esterne come i database relazionali. Questa architettura permettere a Kafka di essere altamente scalabile e resistente ai guasti, infatti se uno dei server non ha la possibilità di erogare il servizio, la piattaforma continua a funzionare grazie alla presenza degli altri server che offrono il servizio necessario ai client;

- *Client*, permette lo sviluppo di un'applicazione distribuita per la lettura, scrittura e il processing di stream di eventi. Inoltre, mette a disposizione una libreria di alto livello per Python, C/C++ e molti altri linguaggi che prende il nome di Kafka Stream.

I concetti chiave di Kafka sono:

- *Evento*, l'unità base di Kafka, anche chiamato record o messaggio, e ha lo scopo di indicare qualcosa che è successo. Generalmente è composto da una chiave, un valore, un timestamp ed, opzionalmente, da un'intestazione contenente dei metadati;

- *Producers* e *Consumers*. I Producers sono coloro che pubblicano un evento in Kafka, mentre i Consumers si sottoscrivono per leggere gli eventi. Uno dei punti di forza di Kafka è che Producers e Consumers non si conoscono a vicenda, infatti i due attori non sono dipendenti l'uno dall'altro e ciò porta ad un'alta scalabilità del sistema. Inoltre, un altro vantaggio di Kafka è l'utilizzo della politica exactly-once, cioè si è certi che un messaggio viene processato una e una sola volta;

- *Topics*. Gli eventi sono organizzati e salvati nei Topics, che possono essere visti come delle directory di un file system all'interno delle quali ci sono i file, che per Kafka sono gli eventi. Ogni Topic in Kafka può avere zero o più Producers e Consumers che pubblicano o ricevono gli eventi. Inoltre, a differenza dei tradizionali sistemi di messaggistica, è l'utente a decidere per quanto tempo gli eventi rimarranno salvati all'interno del Topic, e fino a quel momento ha la possibilità di rileggerli tutte le volte necessarie;

- *Partizionamento*. I Topic sono Partizionati, cioè un Topic è diviso in più parti ognuna delle quali è salvata su un Broker differente. Come possiamo vedere in figura 3.4, quando un evento è pubblicato, viene inserito in una delle partizioni e ogni volta che un evento con una stessa chiave è creato, questo verrà inserito sempre all'interno della stessa partizione. Kafka garantisce che i messaggi che appartengono ad una partizione verranno letti sempre nello stesso ordine con cui sono stati inseriti;

- *Connectors*, permettono a Kafka di collegarsi con sistemi esterni come database, raccolte chiave-valore, file system, ecc. I Connectors di Kafka sono dei componenti che aiutano ad eseguire l'import dei dati da un sistema esterno ai Topic Kafka e viceversa. Esistono due tipi di Connector: Sink, che invia i dati dai Topic Kafka ad un altro sistema e Source, che invece rileva un cambiamento all'interno di una raccolta di dati e lo invia ai Topic Kafka.

Per rendere i dati sempre disponibili e al riparo da eventuali guasti, Kafka da la possibilità di replicare i Topic anche all'interno di datacenter differenti. Un uso comune è quello di utilizzare un fattore di replicazione pari a 3, in modo tale da avere sempre tre copie degli stessi dati.

Ultimo aspetto da analizzare riguardo a Kafka è il suo ecosistema. In figura 3.5 è possibile notare come all'interno di un cluster ci siano più Broker che sono gestiti da Zookeeper, che ha lo scopo di mantenere lo stato del cluster ed eleggere il leader. Inoltre, un altro obiettivo di Zookeeper è quello di notificare ai Producers e Consumers la presenza di un nuovo Broker o eventuali errori su di essi. Quindi, riassumendo le operazioni che vengono eseguite all'interno di Kafka, è possibile dire che i Producers comunicano con Zookeeper per recuperare gli id dei Brokers verso i quali devono inviare i messaggi. Invece, i Consumers leggono i dati utilizzando un offset che viene messo a disposizione da Zookeeper, necessario per iniziare a leggere direttamente dalla posizione desiderata.

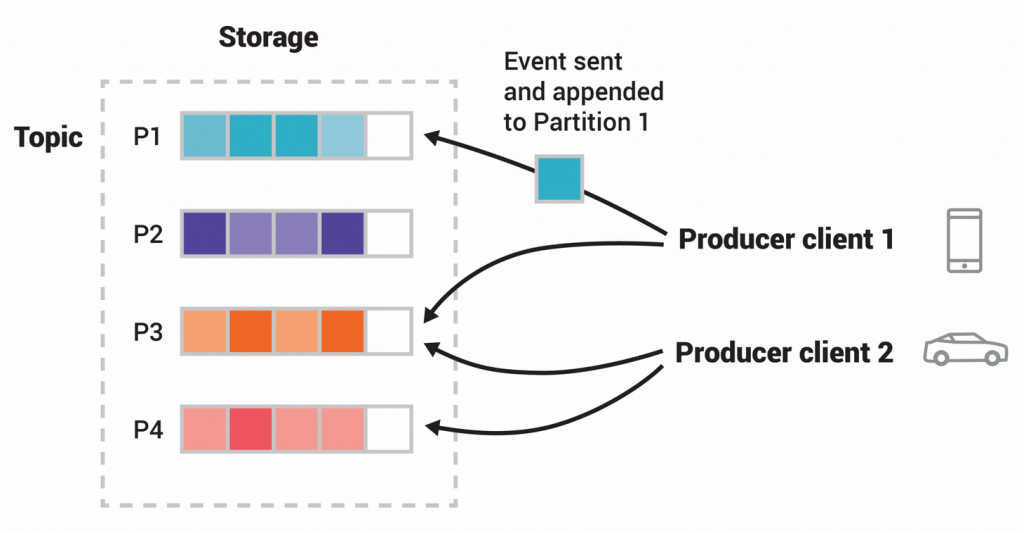


Figura 3.4: Partitioned Topics

Per il salvataggio dei dati si è scelto di utilizzare *MongoDB*, un database non relazionale e quindi lontano dalla struttura tradizionale basata su tabelle tipica dei database relazionali. La scelta è ricaduta su MongoDB soprattutto per la caratteristica di memorizzare i dati in un formato JSON-like, stesso formato utilizzato da EPCIS per la rappresentazione dei dati nella versione 2.0. Inoltre, un altro vantaggio importante per XTAP è quello relativo alla scalabilità orizzontale, quindi rimane facile da usare anche in caso di crescita della piattaforma. Sono molte le caratteristiche che hanno portato MongoDB ad essere uno dei database non relazionali più utilizzati ad oggi, tra le più importanti abbiamo:

- *Query ad hoc*, infatti MongoDB supporta la ricerca per campi, intervalli e regular expression. Le query possono restituire campi specifici del documento e anche includere funzioni definite dall'utente in JavaScript;
- *Indicizzazione*, così come nei database relazionali, c'è la possibilità di creare indici su ogni campo. Inoltre, sono disponibili anche indici secondari, indici unici, indici sparsi, indizi geospaziali e indici full text;
- *Alta Affidabilità* grazie ai replica set, che sono due o più copie dei dati. Ogni replica può avere un ruolo primario o secondario, il primo permettere di fare scritture e letture, mentre il secondo deve mantenere una copia della replica primaria e permette solo la lettura. Inoltre, se la copia primaria fallisce, parte un processo di elezione che ha lo scopo di individuare la sua sostituta;
- *Sharding e Bilanciamento dei dati.* MongoDB permette la scalabilità orizzontale; l'utente sceglie una chiave di Sharding, che determina come i dati sono condivisi tra i vari nodi. Inoltre, nel caso in cui uno shard (replica set) sia troppo carico, MongoDB ha la capacità di bilanciare i dati verso uno shard meno carico;
- *File Storage.* MongoDB può anche essere usato come un file system andando a sfruttare le funzionalità precedentemente descritte (replicazioni, bilanciamento dei dati, ecc). Questa funzione, chiamata GridFS, è inclusa nei driver di MongoDB e disponibile facilmente per tantissimi linguaggi di sviluppo. MongoDB espone delle funzioni per la manipolazione dei file;

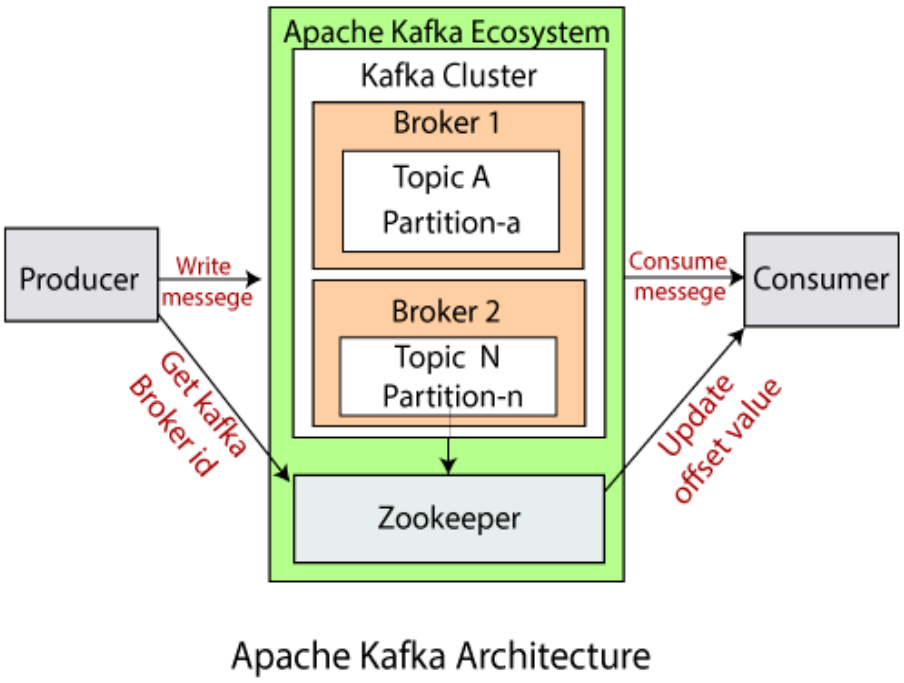


Figura 3.5: Ecosistema Kafka

- *Aggregazione*, che in MongoDB può essere sviluppata in due modi: MapReduce e Aggregration Framework, con la seconda che permette di ottenere risultati più rapidamente;
- *Capped Collection*, che sono delle collezioni a dimensione fissa. Mantiene l'ordine di inserimento e una volta raggiunta la dimensione massima si comporta come una coda circolare.

Come già accennato in precedenza, i dati di XTAP sono salvati in repository differenti per permettere la loro visualizzazione solo ai partner autorizzati. Questa separazione non è gestita fisicamente, infatti le informazioni di un certo tipo sono salvate tutte in una stessa collezione e sono caratterizzate dalla presenza di due attributi fondamentali: il primo ne specifica il possessore, mentre il secondo specifica chi ha il diritto di lettura del dato specifico. Così facendo è possibile restituire ad un partner solo quelle informazioni per le quali ha il diritto di lettura.

Per il processing degli stream si è deciso di utilizzare *Faust*, che come detto in precedenza è una libraria Python in grado di prendere i dati dai Topic di Kafka e processarli in real time. Il decorator Agent permette di definire degli stream processor che consumano i dati dai Topic e fanno qualcosa per ogni evento che ricevono. Inoltre, possono essere definiti in modo tale da eseguire le operazioni in modo asincrono. Una funzionalità importante di Faust è quella che permette di salvare i dati in modo persistente grazie alla creazione di una tabella contenente delle coppie chiavi/valori, che possono essere usate come semplici dizionari di Python. Le tabelle sono salvate localmente su ogni macchina grazie ad un database embedded molto veloce chiamato RocksDB. Così come in Kafka, Faust supporta il concetto di finestra che può essere saltata, si può scorrere e può scadere. Per l'affidabilità si scrivono i log all'interno dei Topics, quindi

Faust viene evocato non appena si nota un cambiamento e mantiene una replica dei dati. Per questo motivo può anche essere usato per recuperare dai guasti pubblicando nuovamente all'interno dei Topic di Kafka i messaggi persi.

Come già accennato in precedenza, il ruolo di Faust all'interno di XTAP è quello di processare gli eventi non appena arrivati. In particolar modo si fa un match con delle regole definite dai vari attori e in base a questo si decide a chi rendere visibili le informazioni contenute all'interno dell'evento.

Tutti i componenti dell'architettura di XTAP sono gestiti con *Docker*, che permette di creare degli ambienti ad hoc per ogni strumento e facilità la loro condivisione. Questo ambiente prende il nome di container e a differenza di una macchina virtuale non include un sistema operativo ma sfrutta l'isolamento delle risorse e i namespace separati per isolare ciò che l'applicazione può vedere del sistema operativo. Questo porta ad un grande vantaggio dal punto di vista delle risorse utilizzate. Ogni container è creato a partire da un'immagine, cioè un modello che definisce la struttura del container stesso. All'interno di XTAP tutte le componenti fondamentali sono state inserite in container di Docker e le immagini create sono:

- Back-end;
- Zookeeper;
- Faust;
- Kafka;
- Kafdrop, un tool di amministrazione per il monitoraggio e la gestione dei Topic Kafka;
- MongoDB;
- Kafka-REST.

Nella figura 3.6 è possibile vedere l'architettura di XTAP. Il primo blocco contiene il back-end che tramite delle API REST riceve dati e eventi EPCIS. Al suo interno è implementata la logica dell'applicazione e tutte le API che possono essere utilizzate dal front-end per richiedere le informazioni di cui ha bisogno l'utente. Anche quest'ultimo è contenuto all'interno del primo blocco è contiene l'interfaccia usata dall'utente per interagire con il sistema. I dati ricevuti da questo primo blocco sono salvati all'interno di MongoDB che li salva in una delle sue replica e li gestisce utilizzando tutte le proprietà descritte in precedenza. MongoDB è collegato a Kafka tramite i Connectors e grazie a questo collegamento ogni nuovo evento salvato nelle collections di MongoDB è anche pubblicato in un Topic Kafka per essere processato. Kafka è gestito da Zookeeper, che si trova all'interno di un altro container di Docker e, come detto in precedenza, ha lo scopo di gestire i Broker. Inoltre, Kafka è collegato a due stream processors gestiti con Faust: il system processor è relativo alle regole di forwarding degli eventi, mentre il plugin processor serve per la gestione di alcune estensioni ideate per l'analisi sui dati. Gli ultimi due blocchi, in basso a sinistra, definiscono i container per Kafdrop e Kafka-REST Proxy, utilizzato ad esempio dal Source Connector e utile per un amministratore della piattaforma.

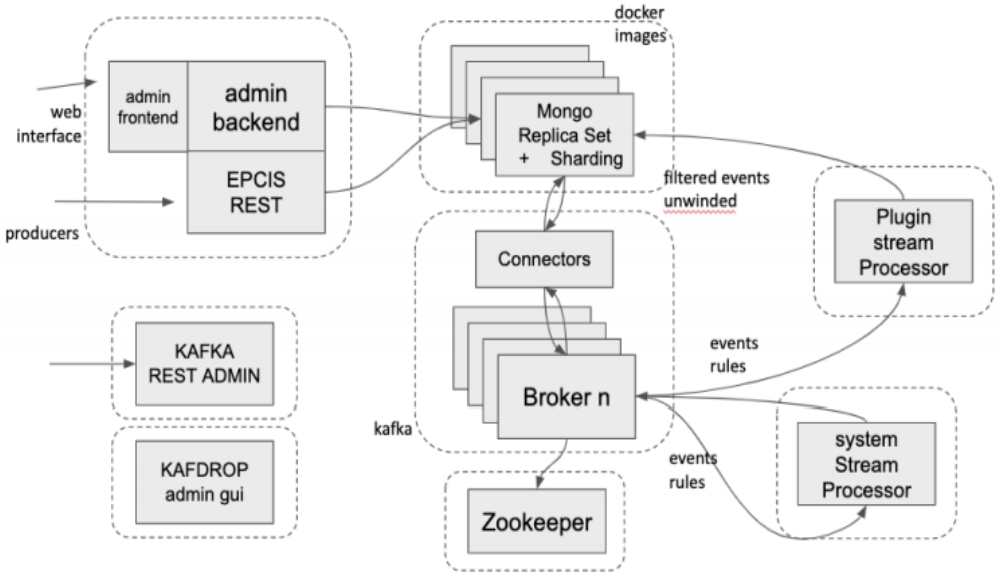


Figura 3.6: Architettura di XTAP

Capitolo 4

Monitoraggio IoT

Come già accennato all'interno di questo documento, l'obiettivo del presente lavoro di tesi è quello di raccogliere le informazioni relative ad un prodotto o un luogo tramite dei dispositivi IoT. Questi dati si vanno ad aggiungere a quelli raccolti durante il passaggio di un prodotto lungo i nodi di una filiera produttiva. In questo modo si ha la possibilità di avere una descrizione del ciclo di vita di un oggetto più dettagliata, avendo a disposizione anche dei dati relativi all'ambiente in cui si trova. Per raggiungere questo obiettivo è necessario introdurre all'interno della piattaforma XTAP delle nuove tecnologie ed è fondamentale trovare una giusta sinergia tra ciò che è già presente e quello che verrà introdotto al fine di raggiungere l'obiettivo finale.

Il Monitoring è diviso in tre fasi: Raccolta dei dati, Stoccaggio e Visualizzazione, ognuna delle quali è un'operazione complessa e che merita uno studio specifico. Le tre fasi si riflettono anche nel lavoro che è stato condotto insieme a due colleghi, il primo dei quali si è occupato della gestione dei dispositivi necessari per la raccolta dei dati, in particolar modo la loro configurazione e il modo con cui comunicano tra loro e con il server. Il secondo si è occupato in particolar modo del back-end, quindi della gestione del server e del database necessari per lo stoccaggio delle informazioni, ed infine io mi sono occupato del front-end, cioè la creazione di un'applicazione in grado di dare all'utente l'opportunità di visualizzare secondo le proprie esigenze i dati raccolti attraverso dei grafici che possono essere salvati ed inseriti all'interno di una dashboard.

4.1 Architettura

Per creare un sistema ottimale c'è stata una prima fase di analisi rivolta a comprendere l'architettura migliore per questo sistema e quale tecnologie utilizzare. L'idea di base è quella di avere dei dispositivi IoT in grado di catturare diverse informazioni come temperatura, umidità, ecc, e di inoltrarle verso un server, il quale si occupa del salvataggio all'interno di un database. Inoltre il server implementa la logica del sistema esponendo delle API, che possono essere contattate dal front-end per effettuare delle richieste per ricevere i dati e visualizzarli in uno o più grafici.

In figura 4.1 è possibile notare l'architettura del sistema di Monitoring. Entrando un pò più nel dettaglio si è deciso di utilizzare dei sensori X-NUCLEO-IKS01A3 per il monitoraggio dei valori di temperatura e pressione e dei sensori P-NUCLEO-IKA02A1 per la qualità dell'aria. Questi dispositivi sono collegati tramite un rete mesh utilizzando BLE e comunicano i dati ad un gateway, dal quale verranno inoltrati verso il server. Infatti, i dati raccolti dai sensori sono inviati verso il back-end di Monitoring e salvati all'interno di TimescaleDB. Inoltre, questi non sono gli unici dati relativi ad

un oggetto, ma ci sono anche quelli proveniente dagli eventi di XTAP. Quindi, a questo punto è necessario gestire non solo i dati raw provenienti dai sensori ma anche quelli di XTAP, che sono inoltrati al back-end di Monitoring grazie ai Sink Connectors di Kafka. Una volta salvati all'interno del back-end, il front-end potrà effettuare delle richieste per recuperare dal server i dati necessari utilizzando RestAPI o GraphQL.

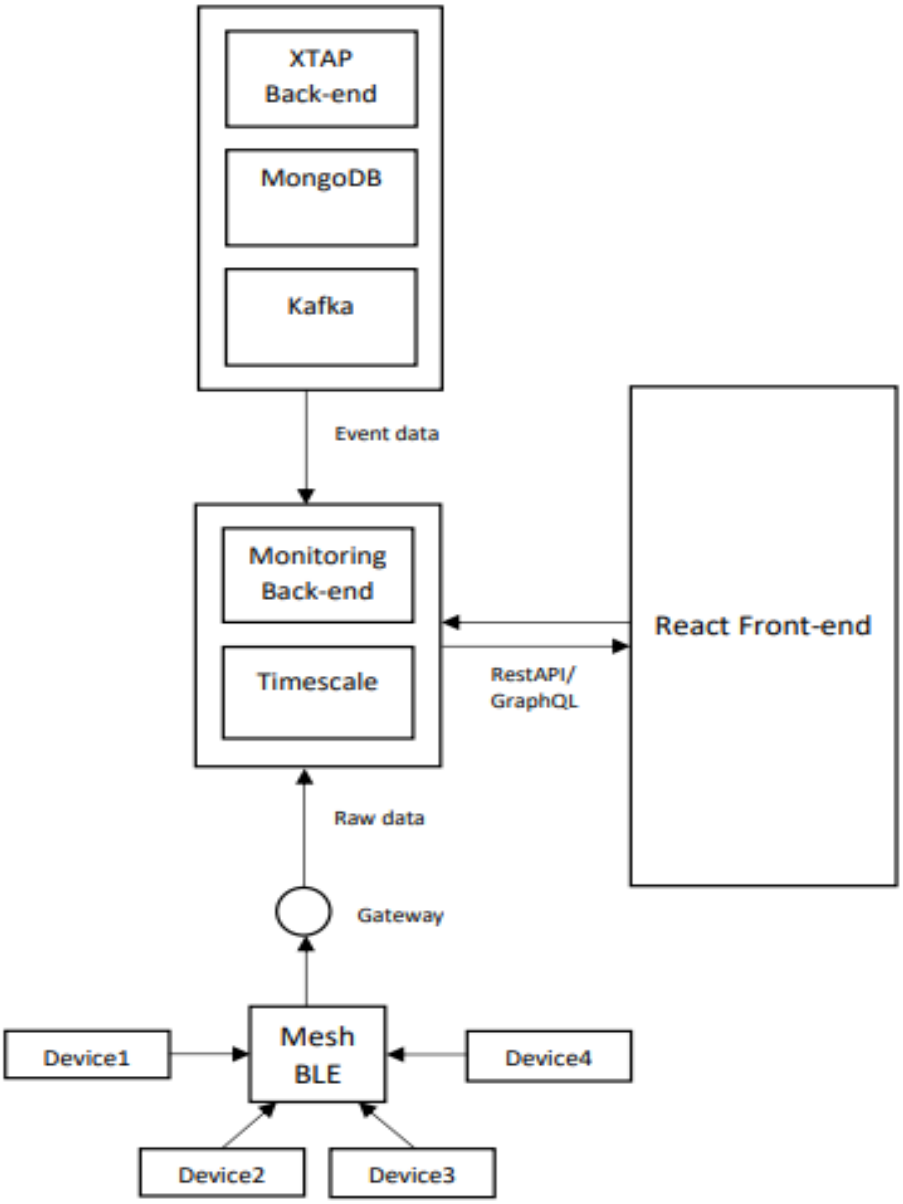


Figura 4.1: Architettura Monitoring

Come descritto in precedenza, per raggiungere l'obiettivo finale ho collaborato con due colleghi e in questo documento mi concentrerò soprattutto sulle scelte fatte riguardo al back-end e al front-end dell'applicazione, poiché l'interesse del mio lavoro è principalmente su questi due aspetti.

4.2 Front-end

Lato front-end l'obiettivo è quello di implementare uno strumento di facile utilizzo per la visualizzazione ed analisi dei dati raccolti dai sensori. Per farlo è sempre necessario pensare dal punto di vista dell'utente in modo tale da creare una User Interface chiara e che aiuti l'utente ad eseguire le operazioni necessarie commettendo il minor numero di errori possibili. Tenendo d'occhio l'obiettivo iniziale, si è pensato che il miglior modo per aiutare gli utenti a visualizzare i dati raccolti sia quello di creare dei grafici specificando tutte le informazioni di cui si ha bisogno, come ad esempio il tipo di dato da visualizzare o eventuali filtri per meglio specificare quali dati devono essere visualizzati. A questa funzionalità si è deciso di aggiungere la possibilità di creare delle dashboard per mettere insieme e salvare dei grafici di particolare importanza.

Le soluzioni che sono state analizzate per il raggiungimento dell'obiettivo sono: la possibilità di utilizzare una piattaforma già esistente per la visualizzazione dei dati, in particolar modo Apache Superset, e la realizzazione ad hoc di alcune pagine all'interno dell'applicazione di XTAP per la gestione dei grafici e delle dashboard. La prima soluzione ha il vantaggio di poter avere in breve tempo una piattaforma funzionante e già testata da molti altri utenti, ma lo svantaggio è che potrebbe non essere completamente conforme a quelle che sono funzionalità che si vogliono introdurre nell'applicazione. Al contrario, lo sviluppo di una soluzione ad hoc ha un costo più elevato dal punto di vista del tempo e del costo ma permette di implementare delle funzionalità che ricoprono totalmente le esigenze del progetto. Analizziamo le due soluzioni.

Apache Superset è un'applicazione web moderna di business intelligence pronta per l'utilizzo in azienda. Superset è veloce, leggera, intuitiva e piena di opzioni che rendono il suo utilizzo semplice per gli utenti di tutti i livelli, i quali possono esplorare ed analizzare i loro dati creando dei grafici che vanno dai più semplici ai più complessi. Superset si appoggia al cloud per il salvataggio dei dati ed è progettata per essere sempre disponibile, per scalare quando la dimensione dei dati aumenta e per essere facilmente utilizzata all'interno di un container Docker.

Le caratteristiche che contraddistinguono Superset sono:

- Un'interfaccia intuitiva per la visualizzazione dei dataset e per la creazione di dashboard interattive;
- Un'ampia gamma di grafici che possono essere utilizzati per visualizzare i dati necessari;
- La possibilità di definire i dati da estrarre e visualizzare senza far riferimento ad alcun codice;
- Un IDE SQL per la preparazione dei dati da visualizzare;
- Un livello semantico leggero che permette agli analisti di definire rapidamente dimensioni e metriche personalizzate;
- Supporto immediato per la maggior parte dei database SQL;
- Cache e query asincrone in memoria senza soluzione di continuità;
- Un modello di sicurezza estensibile che consente la configurazione di regole molto complesse su chi può accedere a quali funzionalità e set di dati del prodotto;
- Integrazione con i principali backend di autenticazione (database, OpenID, LDAP, OAuth, REMOTE_USER, ecc.);

- La possibilità di aggiungere dei plugin di visualizzazione personalizzabili;
- Un'API per la personalizzazione programmatica;
- Un'architettura cloud-native progettata da zero per la scalabilità.

Dopo questa breve descrizione è possibile notare come lo strumento in questione abbia molte caratteristiche per le quali potrebbe sembrare la scelta migliore per il nostro caso ma facendo un'attenta analisi si è notato che non si adatta perfettamente ai dati in questione e soprattutto a ciò che si vorrebbe implementare, cioè una selezione dei dati che faccia riferimento allo standard EPCIS. Infatti, utilizzando Apache Superset, non è possibile specificare i dati da visualizzare facendo riferimento alle cinque dimensioni che descrivono un evento EPCIS. A causa di questa mancanza, che risulta fondamentale per il presente lavoro di tesi, si è deciso di adottare una soluzione più dispendiosa dal punto di vista del lavoro e del tempo necessario ma sicuramente più adatta a quello che è l'obiettivo finale.

Quindi, per l'implementazione lato front-end si è deciso di utilizzare *React*, una libreria JavaScript per lo sviluppo d'interfacce utente, mantenuta da Facebook e da una comunità di singoli sviluppatori e aziende. In questo caso non è stato necessario effettuare un'analisi per determinare la tecnologia da utilizzare poiché React è una delle librerie maggiormente usate ad oggi. Inoltre, React è già utilizzato all'interno di XTAP e ciò permette una facile integrazione con il resto del sistema. I concetti chiave di React sono:

- *Componente*, che permette di suddividere la User Interface in parti indipendenti, riutilizzabili e di pensare ad ognuna di esse in modo isolato. Esistono due tipi di componenti: funzionale e di classe. Il primo consiste nel definire il componente come una funzione JavaScript, mentre il secondo permette di definire il componente come una classe;
- *DOM virtuale*, React crea una cache della struttura dati in-memory, calcola le differenze risultanti e aggiorna in modo efficiente il DOM visualizzato dal browser. Questa è una funzionalità fondamentale in quanto ogni volta che si verifica un cambiamento di stato non si fa il rendering dell'intera pagina ma solo dei sottocomponenti che cambiano effettivamente, aumentando notevolmente la velocità di reazione dell'applicazione;
- *JSX*, un'estensione di JavaScript simile al linguaggio HTML che permette di descrivere l'aspetto della User Interface. In particolar modo ogni componente contiene una serie di tag necessari per descrivere la struttura della pagina;
- *Hooks*, funzioni che permettono di gestire lo stato dell'applicazione all'interno dei componenti funzionali. Tra le funzioni Hook più importanti ci sono: useState, useEffect e useReducer. Lo stato è uno degli aspetti più importanti di React in quanto permette di mantenere le caratteristiche di un componente nel tempo. Inoltre, ogni volta che uno stato viene modificato si avvia in automatico il re-rendering della pagina come descritto in precedenza.

Oltre ai motivi spiegati in precedenza, ci sono tanti altri vantaggi nell'utilizzo di React: dal punto di vista della sviluppatore permette di avere una buona esperienza grazie allo sviluppo rapido e alla semplicità delle API. Un altro vantaggio è il grande supporto esistente, infatti sono tanti i pacchetti già esistenti oppure i documenti che

aiutano a risolvere i problemi che uno sviluppatore può incontrare. Inoltre, anche dal punto di vista delle performance è una delle soluzioni migliori, infatti la presenza del DOM virtuale permette di effettuare il re-rendering della User Interface efficacemente. Per tutti questi motivi si è scelto di utilizzare React per l'implementazione del sistema sviluppato nel progetto di tesi in questione.

4.3 Back-end

Lo scopo del back-end è quello di implementare la logica del sistema andando a mantenere lo stato dell'applicazione ed esponendo le API necessarie per effettuare tutte le operazioni richieste. A differenza di quanto successo con il front-end, si è deciso di implementare una soluzione separata rispetto ad XTAP. Ciò significa che è stato sviluppato un server esterno a quello già esistente di XTAP ed è stato utilizzato un nuovo database.

Per l'implementazione del server si è scelto di utilizzare Python, ed in particolare *FastAPI*, un web framework moderno e veloce che permette di creare delle API con Python. Si è deciso di utilizzare questa tecnologia perché già utilizzata nel server di XTAP, ma anche e soprattutto perché adatto alle nostre esigenze di creare un server con delle buone performance e di utilizzare una tecnologia semplice, veloce e che permetta di ridurre il numero di bug introdotti in fase di sviluppo dai programmatori.

Sono tante le caratteristiche che rendono FastAPI un'ottima soluzione quando si vuole sviluppare un server, vediamole insieme:

- *Veloce*, relativamente alle alte performance che ti permette di avere;
- *Veloce da Programmare*, cioè la velocità di implementazione di un sistema è incrementata di un fattore che va dal 200% al 300%;
- *Pochi bug*, infatti grazie alla sua semplicità riduce la possibilità di errore umano del 40%;
- *Intuitivo*, grazie alla presenza di un buon supporto per gli editor, il completamento automatico è sempre presente e ciò porta ad un minor consumo di tempo per effettuare il debugging;
- *Facile*, infatti FastAPI è stato progettato per essere facile da imparare e da usare. Questo è un grande vantaggio perché riduce il tempo necessario per l'apprendimento iniziale.
- *Sintetico*, FastAPI permette di scrivere in modo sintetico andando a minimizzare la duplicazione del codice e ciò porta ad una minore possibilità di introdurre bug;
- *Robusto*, potendo realizzare un codice pronto per la produzione;
- *Basato su Standard*, infatti si basa su due standard essenziali per la produzione di API: OpenAPI e JSON Schema. Il primo è uno stardard che permette sia agli umani che ai computer di scoprire quali sono le funzionalità offerte da un servizio senza avere accesso diretto al codice, mentre il secondo è un vocabolario che permette di descrivere quella che dovrebbe essere la struttura di un documento JSON in modo tale da poterlo validare automaticamente.

FastAPI può essere utilizzato con *Pydantic*, che permette di validare i dati in ingresso dal client semplicemente descrivendo quale deve essere la struttura dei dati stessi. Il suo funzionamento è molto semplice: è sufficiente creare una classe che derivi da BaseModel, contenuta all'interno di Pydantic, e definire i campi che la classe deve contenere, andando a specificare anche il relativo tipo. Se un oggetto in ingresso al server non rispetta esattamente il formato descritto nella classe Pydantic, la richiesta è bloccata e viene restituito un errore al client.

Il vantaggio di usare Pydantic è che non necessita di ulteriori conoscenze rispetto a Python, infatti la classe di validazione è descritta con la stessa sintassi utilizzata da Python. Inoltre, Pydantic permette anche di configurare alcune caratteristiche prendendo delle informazioni dalle variabili di ambiente, come ad esempio la porta da utilizzare per effettuare delle richieste al database. Altre caratteristiche fondamentali di Pydantic sono la possibilità di creare dei modelli ricorsivi, quindi classi che contengono al loro interno ulteriori oggetti composti, e la possibilità di creare dei tipi personalizzati.

Facendo riferimento allora storage dei dati, si è deciso di utilizzare *TimescaleDB*, un database SQL implementato come estensione di PostgreSQL, ciò significa che è eseguito all'interno di un server PostgreSQL come parte dello stesso processo. Ogni operazione eseguita su un database PostgreSQL che include TimescaleDB viene prima processata da quest'ultimo per determinare come deve essere pianificata ed eseguita. Il vantaggio di appoggiarsi su PostgreSQL è che si ha la possibilità di sfruttare tutte le sue funzionalità, come i più di 40 tipi disponibili, il grande numero di indici utilizzabili e un avanzato query planner. A tutto ciò TimescaleDB aggiunge la possibilità di gestire in maniera ottimale una serie di dati temporali, che, come spesso accade, conduce allo storage di una grande quantità di dati.

Grazie a questa soluzione si ha la possibilità di avere un'analisi veloce e soprattutto una buona scalabilità. Anche in questo caso è stata fatta un'analisi per determinare la migliore soluzione per il caso in questione e la scelta è ricaduta su TimescaleDB poiché la sua caratteristica principale, cioè la gestione delle serie di dati, è uno degli aspetti più importanti da considerare nella creazione di un sistema rivolto alla gestione del Monitoring e alla gestione di una grande quantità di dati generati.

Le nuove funzionalità introdotte da TimescaleDB sono rivolte in particolar modo ai dati time-series, tra le più importanti abbiamo:

- *Partizionamento dei dati automatico e trasparente*, le tabelle sono automaticamente e continuamente divise in piccoli pezzi chiamati chunk per migliorare le performance e per sbloccare alcune funzionalità per la gestione dei dati. Inoltre, i dati e gli indici dell'ultimo chunk rimangono in memoria per permettere un veloce inserimento di nuovi record e per effettuare delle query sui dati recenti più velocemente. Le tabelle gestite tramite l'utilizzo dei chunk prendono il nome di hypertable e una volta create il loro funzionamento diventa trasparente per l'utente. Infatti, è possibile effettuare qualsiasi operazione utilizzando gli stessi comandi usati in PostgreSQL. E' possibile visualizzare la struttura delle tabelle in figura 4.2;

- *Compressione colonnare native* con una compressione avanzata specifica del tipo di dato. In genere porta ad una riduzione dello spazio occupato che va dal 94% al 97% e query più veloci ai dati compressi;

- *Continuous e Real-time Aggregation*, con le quali il database mantiene continuamente e in modo incrementale una vista materializzata di dati aggregati per migliorare le performance delle query. Inoltre, gestisce in modo automatico i

dati più recenti e non ancora presenti all'interno delle viste materializzate per restituire delle informazioni sempre aggiornate;

- *Caratteristiche di gestione dei dati time-series automatiche*, come policy per la conservazione dei dati esplicite, policy di riordino dei dati, policy di aggregazione e compressione policy di downsampling e altro ancora;
- *Framework di pianificazione del lavoro* per supportare sia le policy native che quelle definite dall'utente, incluse quelle scritte in SQL o PL/pgSQL;
- *Operazione multi-nodo scalabile orizzontalmente* per scalare automaticamente i dati time-series su più database TimescaleDB, continuando a dare l'impressione all'untente che ci sia un'unica tabella.

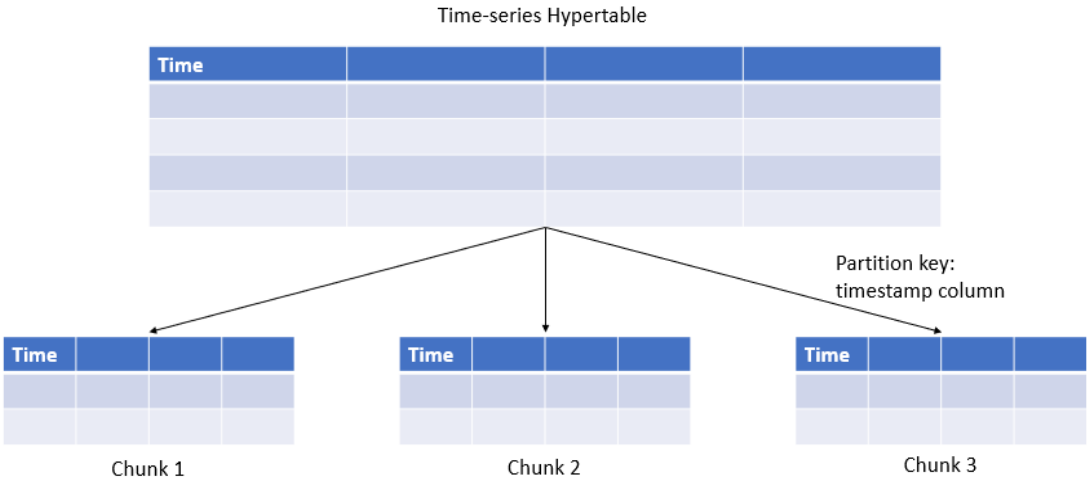


Figura 4.2: Architettura di TimescaleDB

Per tutti i motivi appena descritti TimescaleDB risulta la soluzione migliore per il salvataggio dei dati all'interno dell'applicazione. In particolar modo, uno degli aspetti da tenere in considerazione quando si parla di dati time-series è l'enorme quantità di dati da gestire e la necessità di effettuare delle query velocemente, due aspetti che sono in contrasto. Proprio per questo motivo è necessario utilizzare un database creato appositamente per la gestione di questo tipo di dati.

Così come per XTAP, anche il back-end è gestito con dei container di Docker. In particolar modo, si è deciso di utilizzare due container separati: il primo per il database, che è costruito utilizzando l'immagine 'timescale/timescaleDB-postgis', che permette di avere TimescaleDB e PostGIS già funzionanti su PostgreSQL. Così facendo, all'avvio dell'applicazione è sufficiente popolare il database con i dati degli utenti. Oltre a questo primo container si è deciso di crearne un altro per la gestione del server. Inoltre, i due container sono collegati grazie all'utilizzo di Docker-compose in modo tale da permettere la comunicazione tra server e database. Quest'ultimo è un tool utilizzato per definire ed avviare molteplici container contemporaneamente.

Riassumendo, le tecnologie utilizzate per il back-end sono FastAPI per lo sviluppo del server e TimescaleDB per il salvataggio dei dati, mentre per il front-end si utilizza React. Inoltre, la comunicazione tra back-end e front-end viene gestita con RestAPI e GraphQL, due modi differenti con i quali il server espone le API e il client effettua le richieste. In tutti i casi è possibile notare che la scelta delle tecnologie da utilizzare è

rivolta all'utilizzo di strumenti che permettano di implementare la soluzione richiesta riducendo il numero di bug ed aumentando le performance del sistema. Per ognuna di queste tecnologie si è fatto uno studio approfondito per meglio identificare le proprietà e i vantaggi che possono essere sfruttati all'interno dell'applicazione e nei capitoli successivi si vedrà nel dettaglio come possono essere utilizzate e in che modo queste tecnologie hanno aiutato a raggiungere l'obiettivo finale.

Capitolo 5

Front-end

Durante il presente lavoro di tesi la mia attenzione si è rivolta soprattutto verso lo sviluppo del front-end. Lo scopo principale di quest'ultimo è quello di interagire con l'utente per permettergli di effettuare tutte le operazione che sono offerte dal sistema. Nel dettaglio del presente lavoro di tesi, lo scopo è quello di offrire all'utente la possibilità di visualizzare, nel modo più adatto alle proprie esigenze, i dati raccolti dai dispositivi IoT, in modo tale da poter effettuare delle analisi o per comparare diversi valori. Quindi siamo nell'ultimo dei tre step del Monitoring, la Visualizzazione.

Come accennato nel capitolo precedente, in una prima fase è stato necessario fare un'analisi per comprendere al meglio le possibili soluzioni e, per i motivi già spiegati, si è scelto di sviluppare le funzionalità necessarie utilizzando React ed introducendo delle nuove pagine all'interno di XTAP.

Nel primo periodo di implementazione si è scelto di utilizzare Storybook, un tool che permette di sviluppare i componenti React al di fuori di un'applicazione, e che in questo contesto prendono il nome di 'storia'. Questa scelta ha permesso di effettuare alcuni esperimenti velocemente, senza avere la necessità di impostare uno stack di sviluppo complesso, cosa che avrebbe portato ad un evitabile spreco di tempo. In questa prima fase, infatti, sono state effettuate alcune prove per individuare la migliore libreria da usare per la visualizzazione dei grafici utilizzando dei dataset fittizi per simulare la presenza di dati ambientali. Inoltre, come spesso accade quando si parla di User Interface, sono state effettuate numerose prove per determinare il miglior layout da attribuire alle pagine da sviluppare, questo per implementare un'interfaccia di facile utilizzo per l'utente. Storybook mette a disposizione un menù attraverso il quale è possibile spostarsi tra le varie storie, inoltre permette la modifica di alcune variabili attraverso un'interfaccia grafica di semplice utilizzo. Un altro vantaggio fondamentale è che, sviluppando un componente con Storybook, c'è la possibilità di includerlo in un'applicazione semplicemente trasferendo il file di sviluppo all'interno del progetto.

L'idea iniziale era quella di sviluppare una pagina che permettesse all'utente di creare dei grafici specificandone diverse caratteristiche, come il nome, la finestra temporale desiderata, il tipo di grafico da utilizzare e quali dati devono essere inclusi nella visualizzazione. Il tutto rimanendo conforme allo standard EPCIS, quindi è stato necessario creare una finestra di selezione dei dati facendo riferimento alle dimensioni dello standard: What, Where, When, Why e How. Oltre a questa prima pagina di creazione di un grafico è necessario introdurre una seconda sezione attraverso la quale è possibile gestire la presenza di più dashboard all'interno del sistema. Per questo motivo si è pensato allo sviluppo di una seconda pagina in grado di gestire la lista di tutte le dashboard di un determinato utente e un'altra pagina per la gestione dei grafici

di una singola dashboard, attraverso la quale è possibile modificarne il layout oppure aggiungere, modificare o eliminare un grafico.

In figura 3.6 è possibile notare i componenti che sono stati realizzati per l'implementazione delle pagine descritte in precedenza. Come già specificato nel capitolo precedente, con la presenza dei componenti di React è possibile suddividere la User Interface in parti indipendenti, e quasi tutti i file utilizzati per la realizzazione del front-end corrispondono ad un componente. Nella cartella create_chart ci sono i componenti per la creazione del grafico ed il file utils.js contiene la funzione utilizzata per la generazione del codice in formato JSON che descrive il grafico da visualizzare. Nella cartella dashboard_layout ci sono i due componenti utilizzati per la gestione di una singola dashboard, mentre il file Dashboards.js fa il rendering di una tabella contenente tutte le dashboard del sistema.

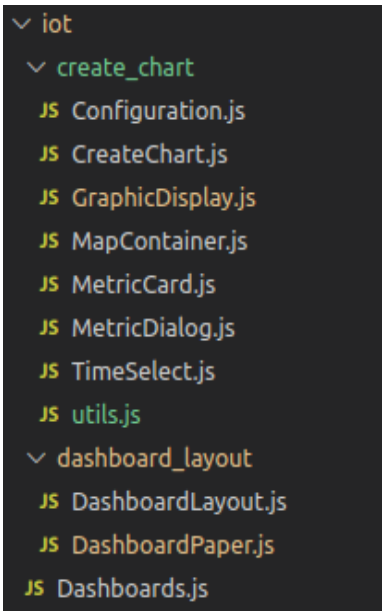


Figura 5.1: Struttura Front-end

5.1 Struttura dell'applicazione

I file appena descritti corrispondo a componenti che messi insieme vanno a definire l'interfaccia che può essere utilizzata dall'utente. Questa suddivisione permette di mantenere i componenti semplici e ognuno di essi gestisce il proprio stato. Inoltre, i componenti definiti in questo modo possono essere utilizzati all'interno di un altro componente come un semplice tag JSX e tutti gli attributi definiti vengono passati al componente figlio in un singolo oggetto chiamato props. Analizziamo nel dettaglio le funzionalità implementate all'interno del sistema e le possibilità che sono offerte all'utente.

5.1.1 Creazione di un grafico

Partendo dalla creazione di un grafico è possibile vedere in figura 5.2 la struttura della pagina: sulla sinistra è presente una colonna (implementata all'interno del file Configurator.js) dentro la quale l'utente deve inserire tutte le informazioni necessarie

per la realizzazione del grafico, ad esempio il nome, la finestra temporale, il tipo di grafico, il dato da visualizzare e gli eventuali filtri. Sulla destra, invece, è presente un componente inizialmente vuoto all'interno del quale è disegnato il grafico (implementato nel file GraphicDisplay.js) non appena sono specificate tutte le informazioni necessarie.

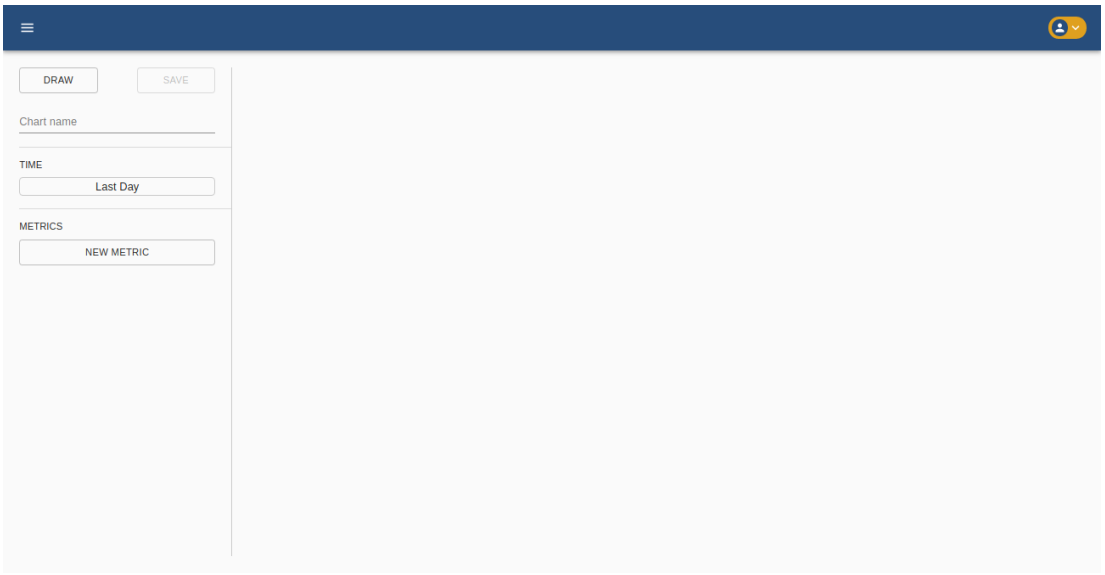


Figura 5.2: Creazione di un grafico

Entrando nel dettaglio del Configurator, in figura 5.3 è possibile notare la presenza di due bottoni nella parte alta: Draw ha lo scopo di fare il rendering del grafico, mentre Save permette di salvare il grafico all'interno di una dashboard. Per il primo è necessario dire che la creazione del grafico è avviata in automatico non appena tutti i campi necessari sono specificati, ma è anche possibile utilizzare il bottone in quei casi in cui vengono effettuate piccole modifiche come il nome o la finestra temporale. Per il bottone Save, invece, è visualizzato un dialog nel quale viene chiesto di selezionare la dashboard in cui il grafico deve essere salvato. Inoltre, se all'interno del sistema non è presente alcuna dashboard, è possibile crearne una nuova specificando il nome desiderato.

Al di sotto dei due bottoni è presente un form necessario per inserire il nome da attribuire al grafico, attributo obbligatorio per poter procedere con la definizione delle informazioni successive. Più in basso è presente un bottone che permette di selezionare la finestra temporale desiderata (sviluppato nel componente TimeSelect.js). Anche in questo caso si utilizza un dialog che verrà descritto a breve.

Infine, c'è la possibilità di creare una o più card (sviluppate nel componente MetricCard.js) per specificare le diverse metriche che si vogliono inserire all'interno del grafico. Con il bottone New Metric è possibile aggiungere una nuova card, mentre con i due bottoni all'interno della card è possibile aprire il dialog per la selezione dei parametri oppure è possibile eliminare la card. E' stata prevista la presenza di più card in quanto l'utente può decidere di visualizzare in un stesso grafico dati differenti su grafici differenti. Ad esempio è possibile visualizzare un line chart e un bar chart nello stesso grafico andando a selezionare tipi di dato differenti.

Premendo il bottone per la selezione della finestra temporale avremo il risultato della figura 5.4. Si è deciso di permettere all'utente di specificare la finestra temporale secondo due modalità differenti in modo tale da adattare al meglio le caratteristiche del grafico alle esigenze dell'utente:

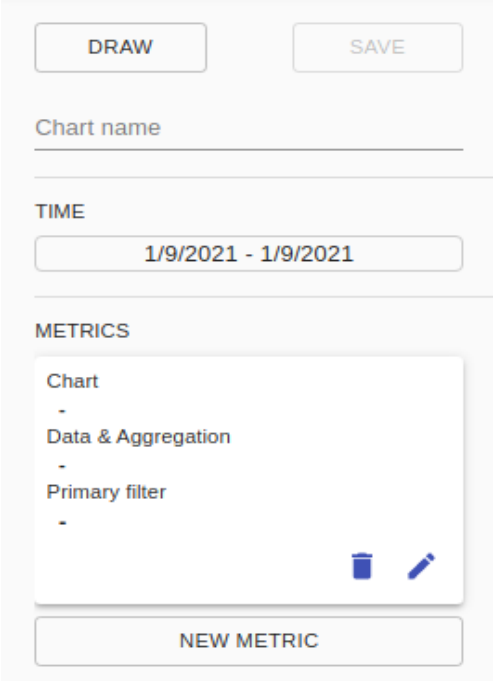
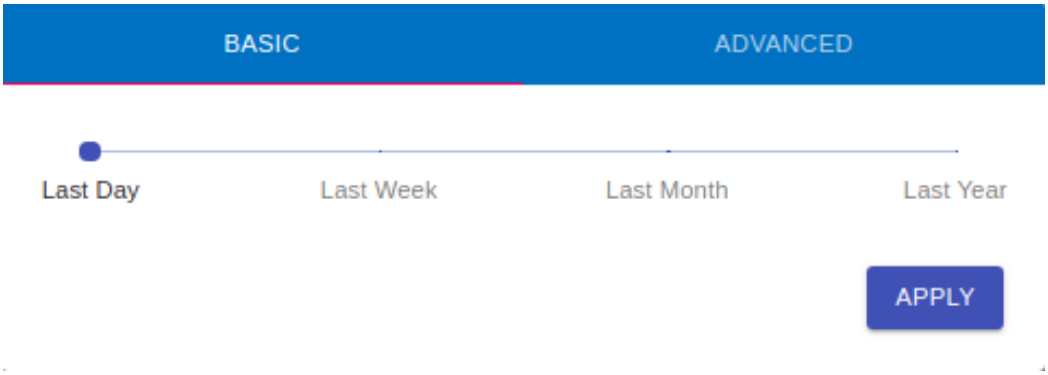


Figura 5.3: Configurator

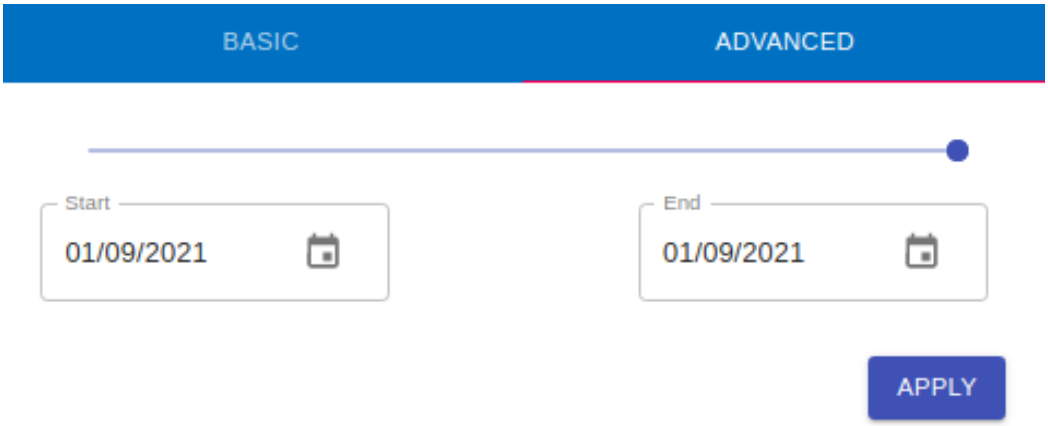
- *Basic*: in questo caso l'utente può solo decidere se visualizzare i dati dell'ultimo giorno, ultima settimana, ultimo mese o anno. Utilizzando questa modalità i dati sono continuamente aggiornati per rispettare sempre la selezione dell'utente;
- *Advanced*: nella modalità avanzata l'utente ha la possibilità di specificare la data iniziale e finale utilizzando lo slider o i due date picker messi a disposizione. In questo caso la finestra temporale è sempre la stessa, a differenza di quanto succede nella modalità Basic.

Dopo aver specificato il nome e la finestra temporale, è necessario individuare i dati da visualizzare nel grafico e per farlo l'utente deve aggiungere una nuova metrica e selezionare una serie di parametri: il tipo di dato, la sua aggregazione, alcuni filtri e dei tag. Aprendo il dialog (implementato nel file MetricDialog.js) ci si trova in una situazione come quella in figura 5.5. A questo punto, ci sono tre step che permettono all'utente di specificare tutto il necessario partendo dalle informazioni generali per arrivare a quelle più specifiche:

- Nel primo step viene chiesto di selezionare il tipo di grafico da utilizzare e al momento le opzioni disponibili sono Line Chart, Bar Chart ed Area Chart. In un'implementazione futura sarà possibile aggiungere nuovi tipi di grafici, come ad esempio l'Heatmap;
- Nel secondo step l'utente deve inserire il tipo di dato da visualizzare (Temperatura, Umidità, Qualità dell'area, ecc) e il tipo di aggregazione che deve essere calcolata (media, minimo e/o massimo). Inoltre, l'utente ha la possibilità di decidere se visualizzare all'interno del grafico anche la deviazione standard o la varianza;
- Infine, nell'ultimo step vengono richiesti i filtri da applicare in modo tale da determinare i dati da visualizzare. I filtri disponibili sono: Product (What),



(a) *Basic*



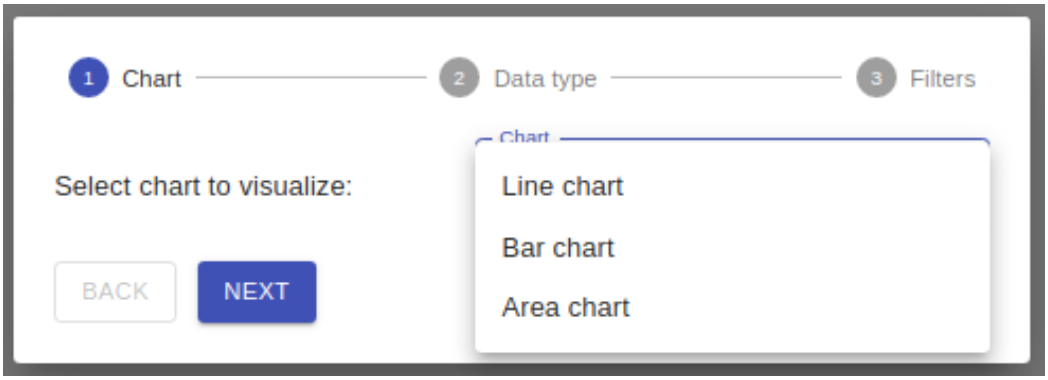
(b) *Advanced*

Figura 5.4: TimeSelect

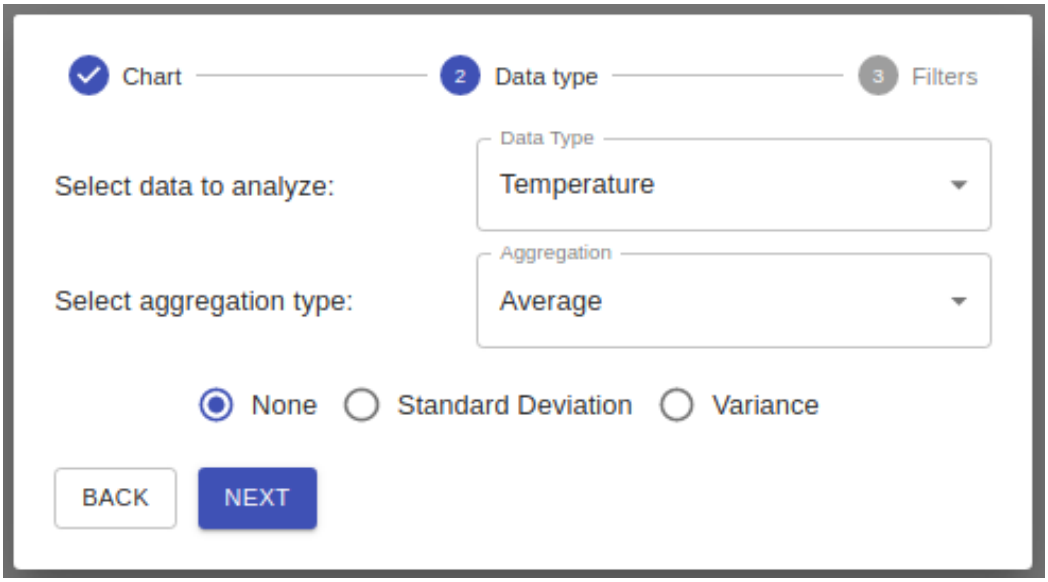
Location (Where), Business Action (Why) e Device (How), e come possiamo notare sono conformi allo standard EPCIS. Per ogni filtro si deve selezionare il tipo di dato richiesto, ad esempio per il Where le due opzioni disponibili sono Business Location e Read Point, ed infine è necessario specificare uno o più valori per determinare quali dati devono essere selezionati. Un'ulteriore selezione che può essere specificata nel terzo step è quella dei tag, che sono dei valori che possono essere assegnati dall'utente per raggruppare gli oggetti che hanno un significato simile.

Tra tutte le possibili combinazioni di filtri, la più interessante è quella che permette di selezionare i dati dei device da visualizzare all'interno del grafico tramite una mappa. In particolar modo, selezionando il filtro 'Device' ed il tipo 'Coordinates', come fatto in figura 5.6, verrà visualizzato un bottone che permette di aprire una mappa interattiva. Questa scelta è stata fatta pensando di dare all'utente più modi per selezionare i dispositivi desiderati, infatti gli stessi device possono essere selezionati tramite il loro nome, il codice GIAI (Global Individual Asset Identifier) oppure tramite la mappa, che ha il vantaggio di aiutare gli utenti ad individuare i dispositivi disponibili e a selezionare quelli necessari in maniera grafica.

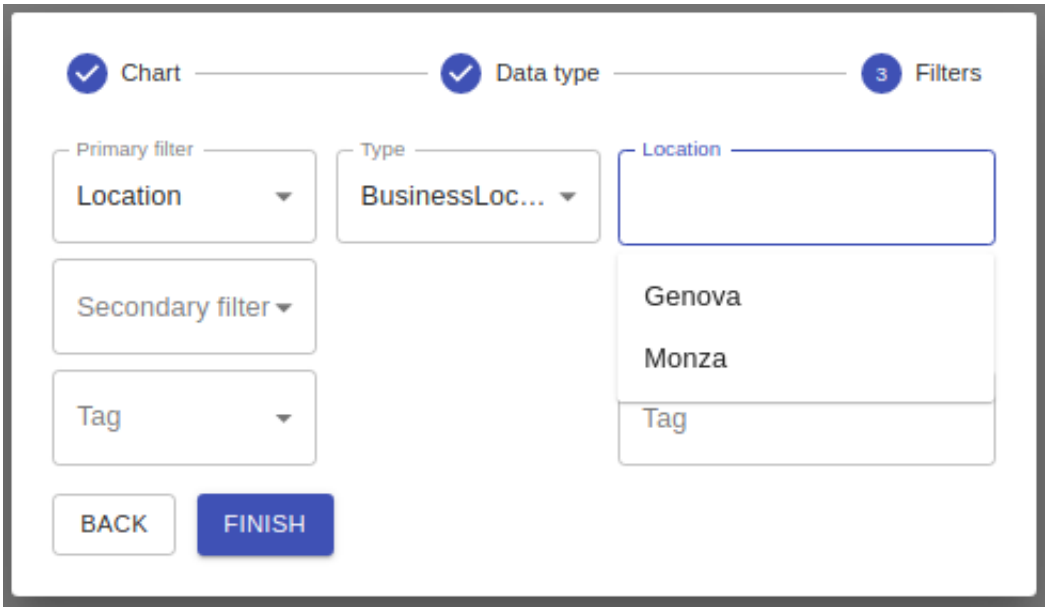
Una volta aperta la mappa, il risultato è quello in figura 5.7. Da questa pagina è possibile visualizzare tutti i device disponibili (marker rosso), ed inserendo dei marker blu è possibile disegnare dei poligoni. Effettuando una selezione utilizzando questa



(a) *Chart*



(b) *Data Type*



(c) *Filters*

Figura 5.5: MetricDialog

modalità, vengono visualizzati nel grafico tutti i dati dei dispositivi che si trovano all'interno dell'area selezionata. Inoltre, c'è la possibilità di cercare una specifica area per visualizzare la sua mappa e selezionare più dettagliatamente i device di interesse. All'interno della mappa sono presenti tutte le funzionalità di base come ad esempio zoom-in e zoom-out oppure la possibilità di muoversi per esplorare zone differenti.

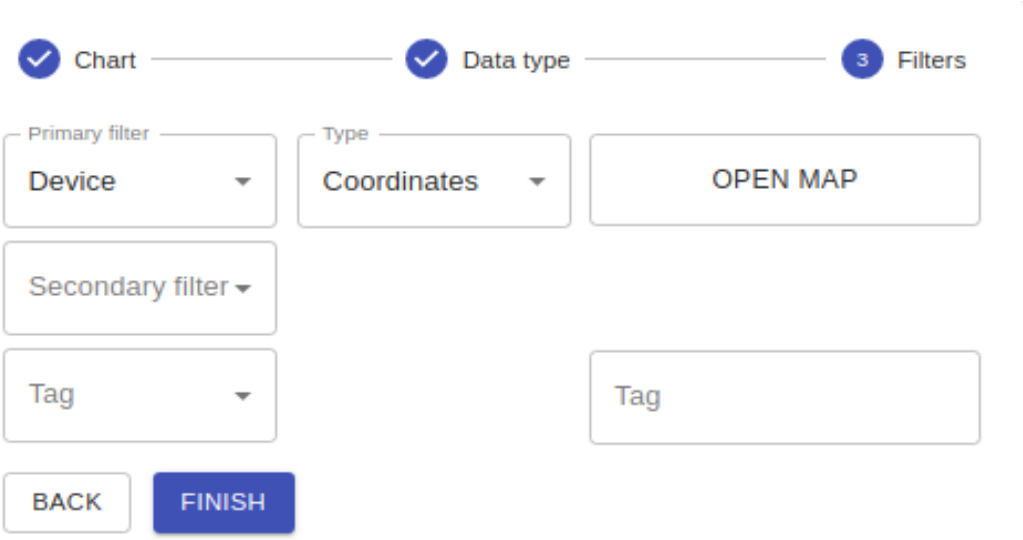


Figura 5.6: Apertura di una mappa

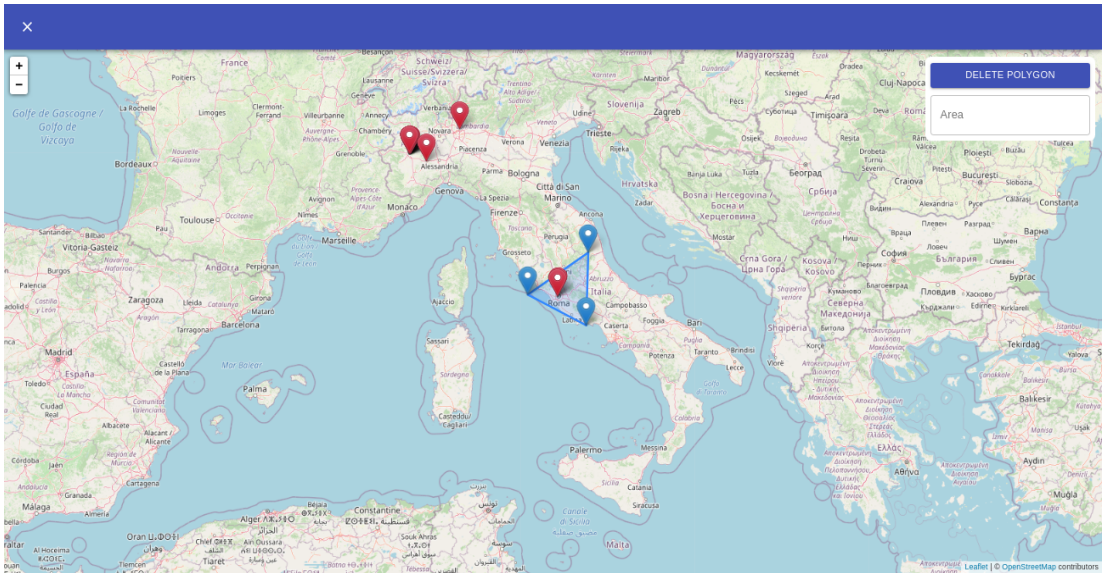


Figura 5.7: Selezione dell'area di interesse

Una volta inseriti tutti i dati descritti in precedenza è possibile "disegnare" il grafico come in figura 5.8. Come è possibile notare, sulla destra viene visualizzato il grafico e in alto c'è una piccola legenda per aiutare la lettura del grafico stesso. Inoltre, se necessario, è possibile salvare il grafico all'interno di una dashboard utilizzando il bottone Save.

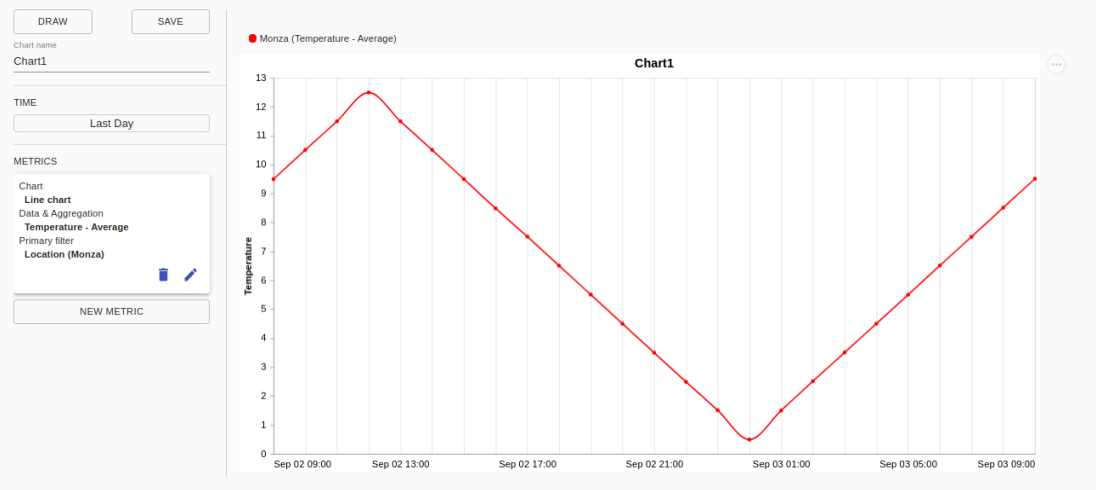


Figura 5.8: Creazione del grafico

5.1.2 Gestione delle dashboard

Per la gestione delle dashboard l'obiettivo era quello di creare un'applicazione in grado di salvare uno o più grafici all'interno di una dashboard, ma anche di permettere all'utente di modificarne il layout alterando le dimensioni e la forma di ogni singolo grafico. Per fare questo si è deciso di utilizzare due pagine differenti: una prima pagina riassuntiva all'interno della quale vengono visualizzate tutte le dashboard del sistema appartenenti al dominio dell'utente, inoltre da qui è possibile effettuare operazione come creazione, modifica o eliminazione di una dashboard. La seconda pagina invece, ha come scopo quello di visualizzare il layout della dashboard, dando la possibilità all'utente di adattarlo alle proprie esigenze e di aggiungere, modificare o eliminare un grafico dalla dashboard in questione.

Entrando nel dettaglio della pagina contenente la lista delle dashboard del proprio dominio, possiamo notare la sua struttura nella figura 5.9. In alto a destra è presente un bottone per la creazione di una nuova dashboard, premendolo si apre un dialog con all'interno un form per l'inserimento del nome, è importante inserire un nome non ancora utilizzato altrimenti viene visualizzato un messaggio di errore. Nella tabella sono visualizzati due dati differenti, il nome della dashboard e la data di creazione, inoltre sono presenti due azioni: la modifica del nome di una dashboard e la sua cancellazione. Per la gestione della tabella sono presenti due funzionalità fondamentali, la prima è l'ordinamento delle righe in base al nome o alla data di creazione e la seconda è la paginazione, che permette all'utente di scegliere se visualizzare 5, 10 o 20 dashboard per volta e di muoversi tra le pagine della tabella.

Spiegate le funzionalità messe a disposizione dalla pagina contenente tutte le dashboard, passiamo ad analizzare la sezione che visualizza il layout di una specifica dashboard. La pagina in questione si presenta come in figura 5.10 e per accedervi è sufficiente clickare sulla riga corrispondente della tabella. Nella parte alta della pagina è presente il nome della dashboard e allineato ad esso, sulla destra, è presente il bottone per editare la dashboard. Quest'ultimo permette di modificare il layout della dashboard, di aggiungere nuovi grafici, ma anche di modificare o eliminare uno o più grafici. Più in basso sono disponibili cinque bottoni con i quali è possibile visualizzare i dati dei grafici specificando temporaneamente delle finestre temporali differenti da quelle scelte in fase di costruzione del grafico. In particolar modo, il primo bottone

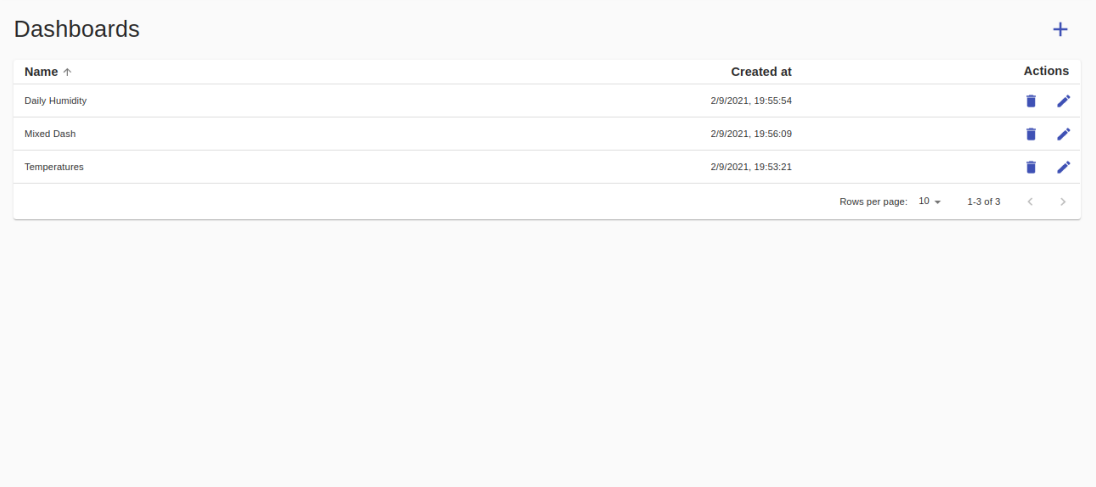


Figura 5.9: Lista delle dashboard

permette di mantenere le specifiche di default dei grafici, mentre utilizzando i restanti bottoni è possibile selezionare una finestra temporale di un giorno, una settimana, un mese oppure un anno. Per spiegare meglio questa funzionalità occorre dire che tutte le specifiche scelte per la composizione del grafico rimangono uguali, l'unica caratteristica a cambiare è il tempo. Questo permette di comparare diverse misure relativamente ad uno stesso periodo di tempo. Infine, nella restante parte della pagina è presente una card con all'interno tutti i grafici presenti nella dashboard.

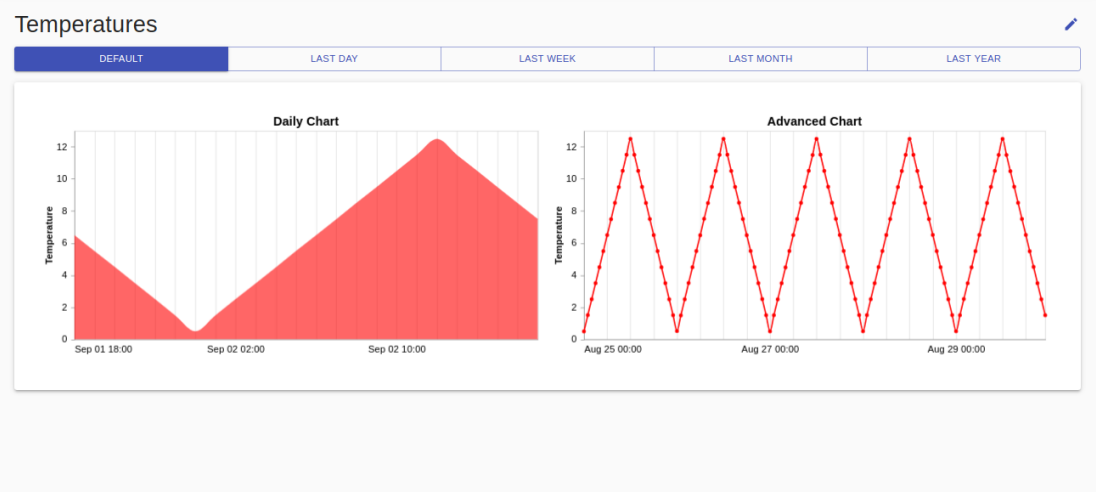


Figura 5.10: Layout di una dashboard

Modificando una dashboard, la struttura della pagina cambia leggermente e si presenta come in figura 5.11. Infatti, al centro della pagina è ancora presente la dashboard ma, a differenza di prima, adesso è possibile modificare la dimensione di un grafico, spostarlo o selezionarlo. Inoltre, non sono più presenti i bottoni per la modifica temporanea delle finestre temporali e il bottone edit è sostituito con quattro bottoni che vanno ad introdurre delle nuove funzionalità necessarie per la gestione della dashboard:

- *Add Chart* permette di aggiungere all'interno della dashboard un nuovo grafico. In particolar modo viene aperto un dialog contenente la stessa pagina di creazione del

grafico spiegata in precedenza, quindi sarà possibile specificare le caratteristiche del grafico e salvarlo nella dashboard;

- *Edit Chart* permette di modificare il grafico selezionato all'interno della dashboard. Anche in questo caso la pagina di creazione di un grafico viene visualizzata in un dialog ma, a differenza della situazione precedente, si presenta con tutti i campi già specificati. Questa operazione è disponibile solo quando è presente un unico grafico selezionato, infatti se due o più grafici sono selezionati, il bottone Edit Chart non è più disponibile;
- *Delete Chart* permette l'eliminazione di un grafico dalla dashboard. A differenza della modifica, l'operazione di eliminazione è disponibile anche con più grafici selezionati;
- *Save* permette di salvare il nuovo layout della dashboard. Una volta salvato l'applicazione indirizza nuovamente l'utente verso la visualizzazione della dashboard non in modalità editing.

Figura 5.11: Modifica di una dashboard

5.2 Librerie

Uno dei motivi per il quale React è una delle soluzioni maggiormente utilizzate è la presenza di un gran numero di librerie che permettono di effettuare alcune operazioni ad un livello di astrazione più alto. La presenza di queste librerie e la loro compatibilità con React permettono di sviluppare applicazioni complesse senza dover pensare ad alcuni particolari che porterebbero solo all'introduzione di bug all'interno del software. Inoltre, essendo utilizzate da tanti altri sviluppatori e quindi essendo testate continuamente ogni giorno, si è certi di utilizzare una porzione di codice che contiene mediamente meno errori rispetto al codice che può essere implementato da un singolo sviluppatore.

Anche nel presente lavoro di tesi sono state utilizzate delle librerie, ed in particolare alcune di esse hanno permesso di implementare dei punti fondamentali del progetto. Non si parla solo di librerie che aggiungono una componente visuale ma anche di libreria che hanno lo scopo di semplificare la logica dell'applicazione. Nel presente documento

verranno presentate tre libreria che, per ragioni differenti, sono state fondamentali nello sviluppo del software.

La prima libreria è quella di *Axios*, un client HTTP per Node.js basato sul concetto della promessa. Una promessa è un oggetto che rappresenta l'eventuale completamento o fallimento di una richiesta asincrona, quindi al suo interno ci sarà il risultato della richiesta oppure un errore che descrive il motivo per il quale non è possibile ottenere una risposta. Il motivo principale per il quale si è deciso di utilizzare Axios all'interno dell'applicazione è che dà la possibilità di gestire l'autenticazione del client tramite un file di configurazione. Entrando nel dettaglio della questione, ogni volta che il client effettua una richiesta al server è necessario trasportare delle informazioni per dimostrare di essere un client legittimo. Per evitare di inserire queste informazioni in ogni richiesta, è possibile specificarle in un file ed Axios inserisce automaticamente queste informazioni nell'header di ogni richiesta effettuata.

Un'altra libreria è *axios-hooks*, che si basa su Axios ma aggiunge delle funzionalità molto utili. Una di queste è la possibilità di gestire in automatico l'attesa per la ricezione di una risposta. In alcuni casi è utile visualizzare all'utente un loader, soprattutto in quei casi dove è necessario prendere dal server una grande quantità di dati. Per questo motivo, axios-hooks mette a disposizione una funzione hook simile a quella rappresentata di seguito.

```
[{ data, loading, error, response }, execute, manualCancel] =
        useAxios(url|config, options);
```

Come possiamo notare, la funzione useAxios accetta due parametri: il primo può essere un'url, cioè l'endpoint del server che deve essere contattato, oppure un oggetto che ha lo scopo di definire diverse configurazioni tra cui l'url, il metodo (get, post, delete, ecc), l'header, il body ed altre ancora. Il secondo parametro, invece, è un altro oggetto in cui possono essere settate alcune opzioni, come ad esempio manual, cioè se la richiesta deve essere eseguita alla creazione del componente oppure se deve essere invocata manualmente, useCache, cioè se la cache deve essere attivata o meno, ssr, per attivare o disattivare SSR ed autoCancel per attivare o disattivare la cancellazione automatica delle richieste pending.

Inoltre, la funzione useAxios ritorna un array composto da diversi parametri, vediamoli insieme:

- *data* contiene il risultato della richiesta;
- *loading* vale True se la richiesta è in corso, altrimenti è False;
- *error* contiene l'eventuale errore della richiesta;
- *response* contiene l'intero oggetto della risposta;
- *execute* permette di eseguire la richiesta manualmente. Infatti, nelle options è possibile specificare se la richiesta deve essere fatta al rendering della pagina o in un secondo momento, in quest'ultimo caso è possibile utilizzare la funzione execute;
- *manualCancel* permette di cancellare manualmente una richiesta in corso.

Avendo questi parametri a disposizione, è facile utilizzare il valore di loading per capire se la richiesta è terminata o meno, ed eventualmente fare il rendering di un loader o del risultato della richiesta.

La seconda libreria utilizzata è quella di *Vega - Lite*, una versione più ad alto livello di Vega, un linguaggio dichiarativo per creare, salvare e condividere grafici interattivi. Con Vega, la struttura e il comportamento interattivo del grafico vengono descritti in un formato JSON, dal quale viene generata la visualizzazione. Tutti i grafici all'interno dell'applicazione sono gestiti con la libreria react-vega, scelta soprattutto per una grande flessibilità che permette di descrivere a propria scelta tutte le caratteristiche che contraddistinguono un grafico, come ad esempio assi, scale, trasformazioni, legende, ecc.

Le specifiche di Vega-Lite descrivono un grafico partendo dai dati disponibili per arrivare alle proprietà dei segni grafici, come ad esempio punti o barre. Il compilatore Vega-Lite produce in automatico tutti i componenti necessari per la visualizzazione come assi, legende e scale. Il vantaggio di Vega-Lite rispetto a Vega è che, essendo più ad alto livello, permette di essere più coincisi per una creazione rapida del grafico. Una caratteristica importante di Vega-Lite è la possibilità di analisi dei dati, infatti supporta sia la trasformazione dei dati che le trasformazioni visive. Infine, l'ultima funzionalità altamente sfruttata all'interno del lavoro di tesi è la possibilità di creare un grafico multi-livello, cioè la capacità di integrare all'interno di un unico grafico più dati anche con visualizzazioni differenti.

React-vega mette a disposizione un tag JSX per la rappresentazione del grafico, al quale è necessario passare la descrizione in formato JSON del grafico, i dati da visualizzare ed altri attributi che possono essere utilizzati per specificare lo stile del grafico, come ad esempio altezza e larghezza.

L'ultima libreria che vorrei introdurre all'interno di questo documento è *Leaflet*, necessaria per lo sviluppo di mappe geografiche interattive. Infatti, questa libreria è stata utilizzata per introdurre la possibilità di selezionare i device di interesse tramite una mappa, come spiegato in precedenza. Nonostante le molte alternative presenti per la gestione di una mappa, la scelta è ricaduta su Leaflet poiché open-source e soprattutto facile da utilizzare. Inoltre, mette a disposizione tutte le caratteristiche necessarie nel nostro caso, come la possibilità di visualizzare dei luoghi di interesse (device) e di disegnare linee ed aree.

Le caratteristiche che hanno reso Leaflet una delle migliori alternative per la visualizzazione di una mappa sono:

- Possibilità di aggiungere più livelli per la raffigurazione di marker o aree, caratteristica molto utile nel caso d'uso del progetto in questione;
- Funzioni di personalizzazione con le quali è possibile modificare tutti i componenti della mappa, ad esempio modificando i colori dei Marker o delle aree disegnate;
- Ottimo supporto da parte dei browser;
- Caratteristiche di interazione come zoom, eventi, trascinamento dei marker, ecc;
- Performance ottimali;
- Caratteristiche visuali;
- Possibilità di aggiungere controlli sulla mappa, ad esempio bottoni o form;
- La sua estrema leggerezza non avendo dipendenze esterne.

Leaflet React mette a disposizione diversi strumenti per rendere il suo utilizzo semplice e veloce. Primo su tutti è il componente MapContainer che accetta diversi attributi, come il centro utilizzato per rappresentare la mappa o lo zoom iniziale. Inoltre,

altri componenti importanti sono il Marker, Polygon, che serve per raffigurare un poligono all'interno della mappa e Tooltip, necessario per la visualizzazione di tooltip personalizzati.

Capitolo 6

Back-end

Lo scopo principale del back-end è quello di preservare lo stato dell'applicazione e di implementare la logica necessaria per svolgere tutte le funzionalità richieste. Di solito vengono identificati due componenti chiave all'interno del back-end: il server, che comunica con il front-end esponendo delle API raggiungibili dal client ed implementa la logica applicativa e il database, dove vengono salvate tutte le risorse dell'applicazione. Questi due componenti interagiscono continuamente per fornire all'utente le risorse di cui ha bisogno e per aggiornare lo stato dell'applicazione.

Entrando nel dettaglio del presente lavoro di tesi, si è deciso di non includere le nuove funzionalità all'interno del back-end di XTAP ma di creare un nuovo server e di utilizzare un database diverso da MongoDB. Lo scopo è quello di fornire ai sensori un'API per inviare verso il server tutti i dati raccolti, inoltre è necessario recuperare i dati degli eventi EPCIS da XTAP utilizzando i Connectors di Kafka. Questi dati, sia quelli raw che quelli proveniente dagli eventi, devono essere salvati all'interno del database ed è necessario esporre delle API per permette al front-end di richiedere tutte le risorse di cui ha bisogno l'utente.

Come già accennato all'interno di questo documento, per l'implementazione del back-end si è deciso di utilizzare FastAPI, un web framework moderno e veloce che permette di creare delle API con Python. Questa tecnologia si adatta perfettamente a quelle che sono le esigenze del progetto, cioè creare un server che possa servire velocemente tutte le richieste che arrivano dai client per evitare situazioni che possano portare ad un rallentamento dell'applicazione. Nonostante la scelta della tecnologia da utilizzare per l'implementazione del server sia stata abbastanza semplice, è stato necessario effettuare delle analisi per individuare la migliore soluzione per il database. Analizziamo gli studi effettuati nel paragrafo successivo.

6.1 Database

Di particolare interesse è stata l'analisi effettuata per la scelta del database da utilizzare. XTAP si basa su MongoDB, un database non relazione che ha molti vantaggi ma non si adatta bene con i dati time-series, cioè una serie di dati che rappresentano come un asset o un processo varia nel corso del tempo. Per questo motivo si è deciso di non utilizzare MongoDB e, dopo un'attenta analisi, la scelta è ricaduta su TimescaleDB, un database SQL implementato come estensione di PostgreSQL. Le caratteristiche di TimescaleDB sono già state descritte nel capitolo Monitoraggio IoT, ma per arrivare a quella che è la struttura finale del database sono stati effettuati numerosi test nel corso dei mesi per poter prendere alcune decisioni fondamentali. Queste scelte, che verranno

descritte nella continuazione di questo capitolo, hanno permesso di offrire all'utente alcune nuove funzionalità, inoltre hanno migliorato le performance del back-end.

6.1.1 Campi JSON

La prima questione esaminata per decidere la struttura del database è quella relativa ai campi JSON. Infatti, TimescaleDB permette di creare delle colonne JSON o JSONB, in modo tale da supportare dati semi-strutturati. Questo permette da un lato di essere più flessibili sulla struttura dei dati, per non dover modificare la struttura di una tabella ogni volta che un nuovo dato deve essere salvato, ma dall'altro potrebbe portare ad un peggioramento delle prestazione oppure potrebbe introdurre problemi con strutture dati come gli indici.

L'esigenza di utilizzare dei campi JSON all'interno del presente lavoro di tesi deriva dalla possibilità di ricevere nuovi tipi di dato, cosa che porterebbe alla necessità di eseguire un'operazione per alterare la struttura della tabella. Inoltre, come già visto nel presente documento, c'è la possibilità di assegnare dei tag ad prodotto e quindi anche ad una misurazione. Questo è un problema quando non si sa a prescindere il numero di tag che possono essere utilizzati.

Per valutare la possibilità di utilizzare dei campi JSON sono stati effettuati dei test per esaminare il comportamento dal punto di vista delle performance. Per questo scopo sono state create due tabelle contenenti 20 colonne, nella prima utilizzando dei campi numerici, mentre nel secondo caso utilizzando campi JSONB. In particolar modo si vuole valutare la velocità di lettura e scrittura, quindi sono eseguite delle operazioni di Insert e Select. Inoltre, si è valutato la quantità di spazio occupata dalle tabelle e dai relativi indici. Nel seguito è possibile esaminare i risultati del test appena descritto.

#Data	**Insert (s)**	**Select (s)**	**Index(MB)**	**Table(MB)**
1M	284.393	0.180	591	232
2M	278.438	0.182	1182	465
3M	280.438	0.186	1773	698
4M	282.468	0.188	2363	930

Tabella 6.1: Campi JSON

#Data	**Insert (s)**	**Select (s)**	**Index(MB)**	**Table(MB)**
1M	351.181	0.062	132	63
2M	338.982	0.056	264	126
3M	350.816	0.077	396	188
4M	327.966	0.051	528	251

Tabella 6.2: Campi SQL

Come possiamo notare dai risultati ottenuti, i tempi di inserito nella tabella con i campi numerici sono leggermente superiori rispetto ai campi JSONB. Però c'è un notevole vantaggio durante le esecuzioni delle Select, circa un terzo del tempo necessario per i campi JSONB. Inoltre, anche dal punto di vista dello spazio occupato i

campi numerici sono notevolmente migliori. Questi risultati hanno dimostrato che relativamente alle performance i tipici campi di un database relazionale sono migliori rispetto ai campi JSONB. Ciò ha scoraggiato l'utilizzo di questa funzionalità all'interno di TimescaleDB e per questo motivo si è deciso di risolvere i due problemi descritti in precedenza trovando un'altra soluzione.

In particolar modo, per il problema relativo alla possibilità di ricevere nuovi tipi di dato si è deciso di creare un record per ogni dato ricevuto, questo comporta la creazione di record con lo stesso timestamp ma con tipi di dato differenti. Invece, per il problema relativo ai tag che possono essere associati ad una misurazione, si è deciso di utilizzare dei campi di tipo Array. Infatti, PostgreSQL permette anche di inserire un insieme di valori all'interno di un singolo campo e questo permette di non conoscere a priori il numero di tag utilizzati.

6.1.2 Indici

Una delle prime questioni da esaminare quando si parla di database è quella relativa alle prestazioni, infatti, a causa delle grandi quantità di dati gestite da un database, spesso si arriva ad avere una situazione in cui la lettura o la scrittura dei dati risulta troppo lenta. Uno dei modi per migliorare le performance di un database in fase di lettura è l'utilizzo degli indici. Un indice è una struttura contenente i dati di una tabella ma ordinati per una o più colonne permettendo quindi una ricerca dei dati più veloce. Però l'utilizzo di un indice ha anche delle controindicazioni, in particolare è necessario effettuare più scritture per mantenere la struttura dati aggiornata e il loro salvataggio porta ad occupare un maggiore spazio.

L'analisi rispetto alla necessità o meno di avere un indice è dovuta al fatto che i dati necessari per la visualizzazione di un grafico arrivano ad avere delle cardinalità molto alte, si parla addirittura di milioni di record, quindi è necessario trovare un meccanismo adeguato che aiuti ad aumentare le prestazioni in fase di lettura.

Come già accennato, un indice può essere creato utilizzando una o più colonne della tabella. Scegliendo di utilizzare un'unica colonna si aumentano le prestazioni in fase di lettura ma diminuisce il numero di query che possono utilizzare quell'indice, se invece si utilizza un indice multidimensionale le prestazioni peggiorano ma sarà utilizzato più volte. Per meglio capire questo concetto si espone un esempio: si supponga di avere una tabella con quattro colonne: nome, cognome, indirizzo e città. Inoltre, si supponga di avere due query, la prima con una condizione sul nome e la seconda con due condizioni su nome e città. Partiamo col creare un indice con la sola colonna nome, a questo punto eseguendo la prima query otteniamo una risposta in un tempo ottimale ma per la seconda query non è possibile utilizzare l'indice poiché al suo interno non sono presenti i dati relativi alla città. Supponendo invece di avere un indice con le colonne nome e città, sia con la prima query che con la seconda sarà possibile utilizzare l'indice, però in entrambi i casi avremo un leggero peggioramento a causa di una maggiore quantità di dati da leggere.

Un altro aspetto da prendere in considerazione quando si parla di indici multidimensionali è l'ordine con cui le colonne sono specificate. Infatti, supponendo ancora una volta di avere un indice sulle colonne nome e città (specificate in quest'ordine), quando viene effettuata una query si fa prima la selezione sulla colonna nome e poi su città. In questo modo conviene sempre utilizzare come prima colonna quella che viene maggiormente utilizzata all'interno delle query del sistema. Questo concetto è fondamentale ed è stato preso in considerazione nei test effettuati e che verranno analizzati a breve.

Prima di introdurre i test effettuati è necessario parlare di un altro aspetto relativo agli indici, cioè i diversi tipi di indice esistenti. In PostgreSQL ci sono solo quattro tipi che supportano gli indici multidimensionali:

- *B-tree*, un indice maggiormente efficiente quando ci sono dei vincoli sulla prima colonna dell'indice. Nonostante ciò, l'indice può essere utilizzato anche per quelle query che non hanno vincoli sulla prima colonna ma, nella maggior parte dei casi, il query planner deciderà di utilizzare la tabella originale;
- *GiST*, anche in questo caso la prima colonna è la più importante per determinare la quantità di indice da esaminare. Un indice GiST risulta inefficiente se la prima colonna ha pochi valori, anche se ci sono molti valori distinti nelle colonne successive;
- *GIN*, a differenza dei primi due, l'efficacia di ricerca è la stessa indipendentemente dalle colonne dell'indice utilizzate dalle condizioni della query;
- *BRIN*, anche in questo caso non è importante quale colonna viene utilizzata all'interno della condizione della query. L'unica ragione per la quale conviene utilizzare un indice BRIN, a discapito dell'indice GIN, è per avere un parametro di archiviazione pages_per_range diverso.

Nei test effettuati sono state prese in considerazione diverse variabili: il tipo di indice utilizzato, la quantità di dati letti e l'ordine con cui i dati sono inseriti nel database. Relativamente alla prima variabile si è deciso di eseguire un test per ogni tipo di indice in modo tale da poter comparare il tempo necessario per inserire i dati, lo spazio occupato dall'indice e il tempo necessario per effettuare una query. Per la quantità di dati letti sono state aggiunte due colonne nelle tabelle dei test per esplicitare tale misura, mentre per l'ordine dei dati nella tabella si è deciso di effettuare diverse misurazioni andando pian piano ad aumentare il disordine dei record. Inoltre, la tabella creata per effettuare i test è composta da 20 colonne (tempo, tag1, tag2 , ..., tag19) e per l'indice sono state utilizzate le colonne tempo e tag1.

Le query utilizzate per verificare il tempo necessario per l'inserimento e la lettura dei dati sono:

```
SELECT * FROM data WHERE time > '...' AND tag1 = 1;
SELECT * FROM data WHERE time > '...' AND tag1 = 1 AND tag2 = 1;
```

Analizzando le query notiamo che la condizione WHERE della prima query contiene le stesse colonne dell'indice, mentre non è lo stesso per la seconda query dove è presente anche una terza colonna (tag2). Per quanto spiegato prima, ci si aspetterebbe un tempo migliore per la prima query poiché le condizioni sono esattamente sulle stesse colonne dell'indice. Mentre nel secondo caso sarà necessario leggere la tabella principale per verificare la condizione sul tag2.

Nel seguito, per essere sintetici, sono inserite solo le tabelle relative ad un disordine dei dati ben preciso, questo perché si è notato che all'aumentare del disordine dei dati il tempo necessario per effettuare la lettura rimane pressoché costante. Invece, nelle tabelle riportate di seguito sono presenti tutti i tipi di indice descritti in precedenza.

Analizzando i dati a disposizione è stato possibile trarne alcune conclusioni che sono risultate molto utili per la scelta dell'architettura finale. Vediamole insieme:

- Innanzitutto si nota come il tempo necessario per l'inserimento dei dati è pressoché simile (circa 350 secondi per inserire un milione di record), a prescindere dal tipo di indice o dalla quantità di dati già presente nella tabella;

#Data	Insert (s)	Select1 (s)	Select2 (s)	Read data 1	Read data 2	Index (MB)	Table (MB)
1M	343,175	0,176	0,175	224449	56168	73	117
2M	351,084	0,361	0,345	474625	117018	145	234
3M	343,488	0,561	0,533	724919	181867	218	351
4M	364,801	0,709	0,781	974108	244284	291	468
5M	359,811	0,912	0,911	1224360	307200	364	585
10M	1739,153	1,931	1,865	2475122	619366	727	1170

Tabella 6.3: Indice B-tree sulle colonne tempo e tag1

#Data	Insert (s)	Select1 (s)	Select2 (s)	Read data 1	Read data 2	Index (MB)	Table (MB)
1M	376,769	0,301	0,212	224988	56611	85	117
2M	376,183	0,606	0,457	476094	119668	170	234
3M	375,755	0,943	0,849	725847	181904	255	351
4M	389,751	1,321	1,091	975964	244756	340	468
5M	379,136	1,601	1,407	1226302	307522	426	585
10M	1840,22	3,085	2,821	2475346	619147	851	1170

Tabella 6.4: Indice GiST sulle colonne tempo e tag1

#Data	Insert (s)	Select1 (s)	Select2 (s)	Read data 1	Read data 2	Index (MB)	Table (MB)
1M	362,76	0,295	0,212	225531	56305	92	117
2M	344,894	0,609	0,431	475325	118931	186	234
3M	349,924	0,913	0,645	724542	181482	281	351
4M	342,907	1,231	0,911	974088	244004	371	468
5M	348,634	1,551	1,127	1224327	306689	463	585
10M	1737,604	3,264	0,762	2472801	618948	925	1170

Tabella 6.5: Indice GIN sulle colonne tempo e tag1

#Data	Insert (s)	Select1 (s)	Select2 (s)	Read data 1	Read data 2	Index (MB)	Table (MB)
1M	358,119	0,233	0,163	224981	56265	31	117
2M	330,738	0,495	0,319	474329	118599	62	234
3M	348,305	0,739	0,483	723964	180993	94	351
4M	349,338	1,018	0,666	973774	243482	126	468
5M	357,635	1,252	0,841	1223004	305754	158	585
10M	1718,622	2,526	0,891	2473746	618456	316	1170

Tabella 6.6: Indice BRIN sulle colonne tempo e tag1

- Un'altra osservazione molto utile è che le query effettuate su un indice B-tree sono più veloci rispetto agli altri, mentre il secondo indice in ordine di velocità è il BRIN;

- Altro dato fondamentale è quello relativo alla dimensione degli indici. In questo caso l'indice BRIN permette di utilizzare una quantità di memoria abbastanza contenuta, mentre con gli altri tre indici lo spazio occupato aumenta considerevolmente;

- Come già valutato in precedenza, la prima query permette di effettuare delle letture più velocemente. Infatti, nonostante il tempo delle due SELECT possa sembrare in un primo momento simile, c'è da considerare la quantità di dati letti, che nella seconda query sono circa il 25% rispetto alla prima;

Durante gli studi effettuati sono stati eseguiti altri test andando a considerare indici su tre colonne (time, tag1 e tag2) oppure utilizzando delle query dove la condizione sul tempo era un range. Si è deciso di non riportare le tabelle all'interno di questo documento poiché i loro risultati sono in linea con quelli appena discussi.

Detto questo, è facile capire come gli indici che portano ad un maggior miglioramento sono il B-tree e il BRIN. Ciò nonostante, un dato fondamentale da tenere in considerazione è che, pur guardando l'indice più veloce in fase di lettura (B-tree), il tempo necessario per effettuare la prima query con 10 milioni di dati già inseriti nella tabella è molto elevato, infatti si aggira intorno ai due secondi. Sapendo che il numero di dati a disposizione nell'applicazione è più elevato rispetto a quello considerato, si capisce come il tempo necessario per effettuare una lettura sia troppo alto anche con l'utilizzo dell'indice migliore. Per questo motivo è stata valutata una nuova soluzione, cioè quella spiegata nel paragrafo successivo.

6.1.3 Continuous Aggregates

Per il motivo appena descritto si è deciso di valutare altre soluzioni ed in particolare è stata analizzata la possibilità di usare i *Continuous Aggregates*, una funzionalità resa disponibile da TimescaleDB. Continuous Aggregates è qualcosa di simile alle viste materializzate, ma la differenza più importante è che non necessita di essere aggiornata manualmente ogni volta che un nuovo dato viene inserito nella tabella. Infatti, la riaggregazione è eseguita in background ed è l'utente a specificare ogni quanto tempo i dati devono essere aggiornati attraverso delle semplici policy definite in fase di creazione della vista.

I due principali vantaggi nell'utilizzo dei Countinuous Aggregates sono il miglioramento delle performance, infatti adesso non è più necessario scannerizzare la tabella con i dati raw ma è sufficiente leggere questi risultati pre-calcolati. Inoltre, il secondo vantaggio è la possibilità di salvare i dati raw solo per un periodo di tempo limitato, continuando a mantenere i dati aggregati. Così facendo si hanno dei dati riassuntivi per degli eventi che sono molto indietro nel tempo, invece per gli eventi più recenti si continua ad avere tutti i dati raccolti. Ciò porta ad un grosso risparmio dal punto di vista della spazio occupato.

Per meglio capire le potenzialità dei Continuous Aggregates si espone un esempio: si supponga di raccogliere ogni secondo dei dati relativi ad una CPU e di salvarli all'interno di una tabella. Volendo visualizzare le informazioni orarie dell'ultima settimana relative alla CPU dovranno essere processate 60 secondi * 60 minuti * 24 ore * 7 giorni = 604800

righe di dati. D'altro canto, se si utilizzano i Continuous Aggregates per aggregare i dati orari, le righe da leggere saranno solo 24 ore * 7 giorni = 168.

Il problema di questa soluzione è che i dati non sono aggiornati ad ogni INSERT effettuata, ma solo ad ogni intervallo di tempo specificato dall'utente. Quindi, in fase di lettura i dati più recenti non sono presenti nelle viste materializzate. Per limitare questo problema si potrebbe diminuire l'intervallo di tempo necessario per il refresh dei dati, ma ciò porta ad un aumento del carico computazionale. Quindi c'è un compromesso tra la velocità di lettura e l'aggiornamento dei dati.

Per ovviare al problema appena descritto, TimescaleDB ha introdotto un nuovo concetto: *Real-time Aggregation*. Con le aggregazioni in tempo reale i dati presenti all'interno di una vista sono combinati con quelli della tabella che non sono ancora stati materializzati, il tutto in modo trasparente per l'utente. In questo modo è possibile avere dei dati sempre aggiornati all'ultimo istante ma anche continuare ad approfittare dei vantaggi dei Continuous Aggregates.

Ritornando all'esempio di prima, supponiamo che il refresh dei dati venga effettuato ogni 2 ore. Supponendo di essere nel caso peggiore le righe di dati da analizzare sono: 22 ore + 24 ore * 6 giorni = 166 dalla vista materializzata, alle quali si aggiungono 60 secondi * 120 minuti = 7200 della tabella con i dati raw. Quindi in totale le righe lette sono 7366, notevolmente meno rispetto alle 604800 righe lette dalla tabella raw. Quindi, con questa nuova soluzione è possibile effettuare le operazioni di lettura velocemente, ma continuando a selezionare i dati sempre aggiornati.

Anche in questo caso sono stati effettuati dei test. In particolar modo si è deciso di valutare il reale incremento di prestazioni utilizzando l'aggregazione in tempo reale, quindi sono state effettuate tre simulazioni: nella prima la query è stata eseguita sui dati raw, nella seconda è stata creata una vista materializzata semplice e nell'ultima si è utilizzata l'aggregazione in tempo reale. I risultati ottenuti dai test sono esposti nella tabella 6.5.

Tipo di Query	**Tempo**	**Freschezza dei dati**
Dati Raw	585ms	Aggiornati
Continuous Aggregates	8ms	Ritardo di 2 ore
Real-time Aggregation	211ms	Aggiornati

Tabella 6.7: Continuous Aggregates e Real-time Aggregation

Dai risultati si è notato che il tempo necessario per eseguire una query sui dati raw è stato di 585 ms, per la lettura dei dati nei Continuous Aggregates sono bastati 8 ms e per l'aggregazione in tempo reale sono necessari 211 ms. Quindi si nota come l'aggiunta dell'aggregazione in tempo reale porti ad un peggioramento delle prestazioni ma si continua ad avere un miglioramento rispetto alla query effettuata sui dati raw, inoltre ha il grosso vantaggio di permettere una lettura dei dati sempre aggiornata.

Considerando la necessità di avere dei dati aggiornati all'interno dell'applicazione sviluppata, si è deciso di utilizzare le aggregazioni in tempo reale. Questo è fondamentale nel caso d'uso del lavoro di tesi poiché non è accettabile la possibilità di visualizzare dei dati che non sono aggiornati all'ultimo istante. Questa scelta implementativa ha

anche permesso di non utilizzare gli indici e di fornire tutti i dati necessari al client in un intervallo di tempo ragionevole.

6.1.4 PostGIS

Un altro studio è stato effettuato per capire quale fosse la migliore soluzione per gestire le coordinate spaziali all'interno di un database. Infatti, come già spiegato in questo documento, l'utente ha la possibilità di selezione i dispositivi di suo interesse andando a visualizzare una mappa e disegnando un'area all'interno della quale verranno selezionati tutti i dispositivi presenti.

Per introdurre questa funzionalità è stato necessario trovare una soluzione che permettesse di gestire le coordinate all'interno del database e dopo un'attenta ricerca si è identificato come soluzione migliore *PostGIS*, un'estensione di PostgreSQL che aggiunge il supporto agli oggetti geografici. PostGIS permette di definire dei punti spaziali ma anche delle linee e dei poligoni e per ognuno di questi è possibile indicarne anche l'altitudine. In PostGIS le coordinate possono essere gestite secondo due modalità differenti:

- *Geometry*, dove le coordinate sono gestite su un piano, ciò significa che il percorso più vicino tra due punti è una linea retta;
- *Geography*, in questo caso le coordinate sono gestite su una sfera. Così facendo il percorso più breve tra due punti non sarà più una linea retta ma un arco. Utilizzando questa modalità i calcoli sono più precisi ma c'è un costo dal punto di vista computazionale, infatti è consigliato utilizzarlo solo in quei casi in cui si è certi di gestire punti molto lontani tra loro.

Tra le due possibilità si è deciso di utilizzare la prima poiché nel caso specifico del progetto implementato non è necessario calcolare la distanza tra due punti, inoltre, così facendo si ha un miglioramento dal punto di vista delle performance. Questa estensione risulta fondamentale per il funzionamento dell'applicazione poiché dà la possibilità di inserire le coordinate per ogni sensore del sistema specificandone latitudine e longitudine e, una volta selezionato il poligono di interesse, permette di effettuare delle query per ottenere tutti i dispositivi all'interno dell'area.

6.2 Comunicazione client/server

Lo scopo di un'applicazione back-end è quello di esporre delle API che possano essere raggiunte dal front-end per effettuare delle richieste. Una delle questioni più importanti quando si parla della comunicazione tra client e server è lo stile architetturale da utilizzare per mettere in contatto questi due attori. Al giorno d'oggi, lo stile maggiormente utilizzato è *RestAPI* ma in alcuni casi è conveniente utilizzare altre soluzioni, come ad esempio *GraphQL*. Prima di discutere della soluzione adottata all'interno della nostra applicazione è necessario fare una breve introduzione di queste due tecnologie, analizzandone soprattutto le differenze.

La prima differenza tra RestAPI e GraphQL è che il primo mette a disposizione molteplici endpoint, cioè più punti d'ingresso che possono essere utilizzati dal client per effettuare le richieste. In genere si crea un endpoint per ogni tipo di richiesta di cui il client può avere bisogno, ad esempio, supponendo di avere un'applicazione che gestisce dei corsi, ci saranno tanti endpoint quante sono le operazioni che possono essere

effettuate sui corsi. Invece, GraphQL espone un singolo endpoint e quindi è il client a specificare il motivo per il quale sta effettuando una richiesta. Di solito l'endpoint di GraphQL utilizza il metodo POST, mentre con RestAPI si utilizzano tutti i metodi che mette a disposizione HTTP per meglio specificare il significato di una richiesta. Uno dei vantaggi di GraphQL è che, potendo specificare le informazioni di cui si ha bisogno, si possono ottenere tutte le risorse necessarie effettuando una sola richiesta. Infatti, uno dei problemi di RestAPI è quello dell'over-fetching e dell'under-fetching, dove nel primo caso un endpoint ritorna più informazioni di quelle necessarie, mentre nel secondo caso è necessario effettuare più richieste per recuperare tutti i dati che servono al client. A causa di questa caratteristica si dice che GraphQL è client-driven, mentre RestAPI è server-driven. Infine, un'altra caratteristica fondamentale di GraphQL è che permette di realizzare tre operazioni differenti:

- *Query*, attraverso le quali è possibile chiedere ed ottenere delle risorse dal back-end, è un'operazione simile al metodo GET di HTTP usato in RestAPI. Nella Query vengono definiti i campi da leggere ed eventuali condizioni per filtrare i record. Inoltre, il server definisce uno schema necessario per specificare quali campi possono essere richiesti;
- *Mutation*, che viene utilizzata per aggiungere o modificare una risorsa del sistema, in questo caso può essere comparata alle operazioni POST, PUT e DELETE di HTTP usate in RestAPI;
- *Subscription*, cioè la possibilità del client di iscriversi ad una particolare risorsa ed ottenere tutte le modifiche che vengono effettuate su quella risorsa senza la necessità di eseguire una nuova richiesta ogni volta. In questo caso non c'è nessuna operazione simile in RestAPI.

D'altra parte, RestAPI ha i vantaggi di avere una struttura più rigida ed essere maggiormente diffuso, il che aiuta molto in fase di sviluppo.

All'interno dell'applicazione sviluppata si è deciso di implementare la comunicazione tra client e server utilizzando RestAPI. Si è arrivati a questa decisione poiché la maggior parte delle funzionalità implementate nell'applicazione non hanno nessuna caratteristica particolare che non permetti l'utilizzo di RestAPI, quindi considerando la sua maggior diffusione e facilità d'utilizzo è sembrata la scelta più ovvia. Ciò nonostante, è stato utilizzato anche GraphQL per una richiesta in particolare, cioè la lettura dei dati utilizzati per disegnare un grafico. Un motivo in particolare ha condotto a questa scelta, cioè la possibilità di implementare in futuro la stessa funzionalità ma con dei dati che si aggiornano in real time. Infatti, come detto in precedenza, con GraphQL il client può iscriversi ad una particolare risorsa e così facendo si possono creare dei grafici che aggiornano i dati visualizzati in tempo reale.

6.3 Master Data

Un'altra operazione eseguita nel corso dello sviluppo del back-end è stata quella di inserire i Master Data da XTAP all'interno di TimescaleDB. I Master Data sono dei dati condivisi da un partner verso gli altri e che forniscono una descrizione delle entità nel mondo reale che sono identificate con una chiave GS1, ad esempio oggetti, luoghi fisici e logici. Lo scopo di questa operazione è quello di creare un canale tra XTAP e il back-end del monitoring in modo tale da avere i Master Data continuamente aggiornati. Infatti,

questi dati sono inseriti dagli utenti attraverso l'interfaccia di XTAP e per renderli disponibili anche al back-end del monitoring si utilizzano i Connectors di Kafka.

In particolar modo è necessario prendere i dati relativi al What, Where e Why dal back-end di XTAP e trasferirli in TimescaleDB poiché verranno utilizzati per permettere all'utente di selezionare i dati necessari per la visualizzazione di un grafico. Le collezioni che devono essere esaminate per ricavare questi dati sono:

- api_items, dalla quale è possibile ricavare i dati del What. In questa collezione il dato più significativo è il codice che identifica un prodotto e può essere un GTIN14, Global Trade Item Number su 14 caratteri, oppure una codifica interna dell'azienda. Inoltre, è necessario recuperare anche il nome e l'object_owner, cioè i domini che hanno visibilità su quel dato;

- api_places, dalla quale si ricavano i dati Where relativi alla Business Location. In questo caso, il dato principale è un GLN (Global Location Number) oppure una codice privato, inoltre è necessario leggere il nome e l'object_owner;

- api_suplaces, dalla quale è possibile ricavare i valori di Read Point relativi al Where. Invece di un GLN contiene un SGLN, e ancora una volta è necessario recuperare il nome e l'onject_owner;

- capture_event_templates, dalla quale si ricavano dei valori che non verranno inseriti come dimensione Why, ma potranno essere utilizzati come un tag. In particolar modo, il problema della dimensione Why è che contiene delle operazioni troppo generiche, ad esempio packing, unpacking, commissioning, ecc. Quindi non permetterebbe all'utente del sistema di identificare con precisione l'evento che vuole visualizzare. Per questo motivo, tramite il filtro relativo alla dimensione Why è possibile selezionare un'operazione generica, presa dal CBV, invece tramite i tag possono essere selezionate delle operazioni più specifiche.

Entrando nel dettaglio delle operazioni necessarie per il passaggio dei dati, la prima è quella di utilizzare una serie di Source Connectors per prendere i dati da MongoDB e inserirli all'interno dei Topic Kafka. Per ogni Source Connector è necessario definire il nome del Topic e il nome della collezione dalla quale prendere i documenti. Si utilizzano più Connectors poiché ognuno di essi può prendere i dati da una sola collezione e creare un Topic differente, infatti nel nostro caso è necessario avere un Topic per ogni tipo di dato che vogliamo catturare.

Una volta eseguita questa operazione è necessario catturare i dati dai Topic ed inserirli all'interno di TimescaleDB. A questo scopo è necessario utilizzare dei Sink Connectors. Per farlo si creano diversi Agent con Faust, ancora una volta uno per ogni Topic, e all'interno di questi Agent si scrivono le operazioni necessarie per salvare i dati nel database. Il vantaggio di utilizzare i Connectors è che ogni volta che i dati in XTAP sono modificati, questi sono trasmessi direttamente nei Topic e successivamente in TimescaleDB, ciò permette di avere i Master Data sempre aggiornati con XTAP. In particolar modo, all'interno degli agents di Faust si fa un check per individuare l'operazione che ha scatenato l'inserimento di un nuovo messaggio all'interno del Topic; le possibili operazioni sono insert, update o delete. Fatto ciò, si fanno le operazioni corrispondenti per mantenere lo stato di TimescaleDB aggiornato con quello di MongoDB. Un altro particolare da discutere è come la modifica o eliminazione di un record sono effettuati: infatti, per individuare il giusto record da modificare o eliminare è necessario avere un campo univoco. Per questo motivo si è deciso di inserire all'interno delle

tabelle di TimescaleDB un nuovo campo, 'documentKey', che ha lo scopo di ospitare la chiave primaria del relativo documento in MongoDB. Così facendo, quando un record è inserito nella tabella la chiave primaria del documento è inserita. Se invece le operazioni sono update o delete, si utilizza la chiave primaria del documento per individuare il giusto record da modificare o eliminare.

Capitolo 7

Conclusioni

La funzionalità di Monitoring introdotta all'interno di XTAP ha lo scopo di fornire ai suoi utenti gli strumenti necessari per la visualizzazione dei dati relativi ad un prodotto o luogo facenti parte di una filiera produttiva. L'applicazione entra a far parte di un contesto più ampio, quello della tracciabilità, che è ad oggi uno degli aspetti più importanti quando si parla della produzione di un prodotto. I dati raccolti permettono di descrivere meglio il ciclo di vita di un oggetto, andando ad identificare i suoi problemi o punti di forza in modo tale da migliore la sua produzione. Inoltre, queste informazioni sono particolarmente importanti per i clienti, i quali si informano sempre più sui loro acquisti, soprattutto a livello alimentare.

Una delle caratteristiche del sistema sviluppato, che lo contraddistingue da quelli già esistenti, è la sua conformità con EPCIS. Infatti, uno dei requisiti necessari per essere inglobato all'interno di XTAP è proprio la possibilità di parlare un "linguaggio" comprensibile all'intera applicazione, cioè quello di EPCIS. Sviluppare un sistema in grado di riconoscere EPCIS ha permesso di interoperare con il resto dell'applicazione per meglio raggiungere lo scopo finale. Inoltre, utilizzare EPCIS è un vantaggio poiché è uno strumento molto robusto ed utilizzato a livello globale, permettendo così la comunicazione tra diverse aziende. La conformità con EPCIS ha permesso anche l'integrazione dei dati generati durante la produzione di un oggetto con quelli raccolti tramite i dispositivi IoT.

Nel contesto appena descritto, il presente lavoro di tesi si incentra soprattutto sulla visualizzazione dei dati, operazione fondamentale poiché è il punto di contatto con gli utenti della piattaforma. Anche il front-end, così come il resto dell'applicazione, è stato costruito per essere conforme ad EPCIS in quanto è uno standard già conosciuto dagli utenti anche al di fuori di XTAP. Un aspetto fondamentale nello sviluppo del front-end è quello dell'usabilità, cioè l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti. In pratica definisce il grado di facilità e soddisfazione con cui si compie l'interazione tra l'uomo e uno strumento. Obiettivo del presente lavoro di tesi era quello di mettere l'utente al centro del processo per migliorare l'usabilità del sistema. Analizzando il risultato finale si può dire che l'obiettivo è stato raggiunto poiché nel corso dei mesi c'è stata un'evoluzione continua che ha portato ad un miglioramento dal punto di vista dell'usabilità. L'obiettivo è stato raggiunto nonostante l'elevata complessità del sistema, infatti per essere conformi ad EPCIS è stato necessario introdurre tanti piccoli particolari che da un lato permettono di specificare meglio le esigenze dell'utente ma dall'altro lato aumentano la complessità del sistema.

Dal punto di vista delle funzionalità, gli obiettivi del presente lavoro erano quelli di

sviluppare un front-end in grado di introdurre la costruzione dei grafici specificandone diverse caratteristiche: la finestra temporale, il dato da visualizzare, il tipo di grafico, l'aggregazione dei dati ed infine dei filtri per meglio individuare i dati di interesse per l'utente. Ciò non era sufficiente poiché il miglior modo di comparare dei dati è quello di visualizzare più grafici contemporaneamente. Quindi, un altro obiettivo del presente lavoro di tesi era quello di permettere all'utente di gestire una o più dashboard. Anche per questo obiettivo l'idea era quella di rispettare alcuni vincoli fondamentali nella gestione di una dashboard, come ad esempio la possibilità di modificarne il layout oppure l'opportunità di aggiungere, modificare o eliminare un grafico.

In conclusione del presente lavoro di tesi si può dire che gli obiettivi prefissati nella prima fase di sviluppo sono stati raggiunti in quanto il sistema implementato è in grado di dare agli utenti un processo di creazione dei grafici contenente tutte le caratteristiche richieste. Inoltre, è possibile notare come la configurazione di un grafico sia conforme alla struttura di EPCIS, il che permette agli utenti di orientarsi meglio all'interno dell'applicazione. Relativamente al secondo obiettivo si è ottenuto un risultato simile a quello precedente, infatti è stato sviluppato un meccanismo di gestione delle dashboard con tutte le caratteristiche necessarie.

L'ultimo aspetto da analizzare sono le possibili implementazioni future che possono essere incluse all'interno del sistema. Una di queste è la possibilità di creare dei grafici che si aggiornano in real time, infatti come già descritto nel capitolo del back-end, si è deciso di utilizzare GraphQL per la richiesta dei dati. GraphQL permette ad un client di iscriversi ad una risorsa e in questo modo è possibile ricevere tutti i cambiamenti relativi a quella risorsa: aggiunta, modifica o eliminazione di un record. Sfruttando questa caratteristica di GraphQL è possibile creare dei grafici che aggiornano i dati in tempo reale senza effettuare delle nuove richieste al back-end. Ciò porterebbe ad una migliore esperienza per l'utente che utilizza l'applicazione in quanto avrà la possibilità di visualizzare come i dati cambiano col passare del tempo. Un'altra futura implementazione è quella di inserire la possibilità di creare nuovi tipi di grafici, come ad esempio le mappe o le heatmap. In questi casi, come in altri, è necessario non solo aggiungere i nuovi grafici nella selezione, ma anche modificare il tipo di informazioni che l'utente fornisce per la definizione del grafico.

Bibliografia

[1] Apache Superset, *What is Apache Superset?*, https://superset.apache.org/docs/intro, Accessed: 02/09/2021.

[2] eMoldino, *How it works*, https://www.emoldino.com/how-it-works, Accessed: 10/08/2021.

[3] FastAPI, *FastAPI*, https://fastapi.tiangolo.com/, Accessed: 01/09/2021.

[4] Faust, *Faust - Python Stream Processing*, https://faust.readthedocs.io/en/latest/, Accessed: 30/08/2021.

[5] GS1, *EPCIS and CBV 2.0. Mission-specific work group*, https://www.gs1.org/sites/default/files/docs/gsmp/epcis_2.0_cta_final.pdf, Accessed: 19/08/2021.

[6] GS1, *EPCIS Standard 2.0*, Accessed: 21/08/2021.

[7] GS1, *CBV 2.0*, Accessed: 21/08/2021.

[8] GS1it, *EPCIS - Garantire la tracciabilità in tempo reale dei prodotti*, https://gs1it.org/assistenza/standard-specifiche/epcis-tracciabilita-prodotti-tempo-reale/, Accessed: 19/08/2021.

[9] Kafka, *Documentation*, https://kafka.apache.org/documentation/#gettingStarted, Accessed: 23/08/2021.

[10] Thingsboard, *Thingsboard*, https://thingsboard.io/, Accessed: 10/08/2021.

[11] TimescaleDB, *Why use TimescaleDB*, https://docs.timescale.com/timescaledb/latest/overview/core-concepts/#why-use-timescaledb, Accessed: 01/09/2021.

[12] Trackyfood, *Tracciabilità alimentare: cos'è, come funziona e quali sono le norme*, https://www.trackyfood.com/tracciabilita-alimentare-cose-come-funziona-e-quali-sono-le-norme/, Accessed: 06/08/2021.

[13] Wikipedia, *GS1*, https://it.wikipedia.org/wiki/GS1, Accessed: 19/08/2021.

[14] Wikipedia, *EPCIS*, https://en.wikipedia.org/wiki/EPCIS, Accessed: 19/08/2021.

[15] Wikipedia, *Apache Kafka*, https://en.wikipedia.org/wiki/Apache_Kafka, Accessed: 23/08/2021.

[16] Wikipedia, *MongoDB*, https://it.wikipedia.org/wiki/MongoDB, Accessed: 26/08/2021.

[17] Wikipedia, *React (web framework)*, https://it.wikipedia.org/wiki/React_(web_framework), Accessed: 02/09/2021.

[18] Youtube, *Webinar — Academy di GS1 Italy*, https://www.youtube.com/playlist?list=PL3EThPF3sbJr6TyUOlnHch2lOGyiF2R_9, 20/03/2019.