Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Applicazione web in cloud Microsoft per il controllo di dispositivi ST

Relatore
Giovanni Malnati

Candidato
Davide Sergi

Supervisore aziendale
STMicroelectronics
Ing. Emanuele Quacchio

Luglio 2018

Alla mia famiglia,
stelle guida nella notte nera della mia vita.
A mio fratello Michele,
faro al centro della nostra tempestosa esistenza.

Indice

Sommario

Il lavoro svolto durante questo percorso di tesi sperimentale ha avuto l'obiettivo di sviluppare una soluzione cloud in grado di abilitare chi ne faccia uso a monitorare, controllare e configurare, da remoto e in tempo reale, dispositivi IoT (Internet Of Things) sviluppati da STMicroelectronics.

Le piattaforme hardware di STMicroelectronics utilizzate sono state la famiglia STM32Nucleo e STM32 IoT Discovery Board insieme al firmware package FP-CLD-AZURE1, mentre i servizi cloud utilizzati sono quelli forniti da Microsoft (Microsoft Azure).

Il sistema sviluppato consente agli utenti di crearsi un proprio account e di registrare i propri dispositivi IoT; all'accensione se appositamente configurati, questi invieranno dati di telemetria che verranno poi storicizzati, fino a un massimo di 48 ore, e resi fruibili agli utenti on-demand mediante web dashboard.

Sempre mediante dashboard, è possibile interagire in tempo reale con i dispositivi connessi, in particolare configurandoli mediante apposite strutture dati JSON, o invocando metodi direttamente su di essi, ad esempio al fine di controllare motori elettrici montati sul dispositivo oppure per forzare l'aggiornamento del firmware installato (FOTA). Il sistema infine permette agli utenti di definire delle regole, scritte in linguaggio Javascript, in grado di riconoscere particolari eventi telemetrici e notificare, di conseguenza, via email gli utenti stessi. I devices sono in grado di comunicare con la soluzione sviluppata attraverso l'utilizzo di protocolli standard (MQTT) (Figure 1).

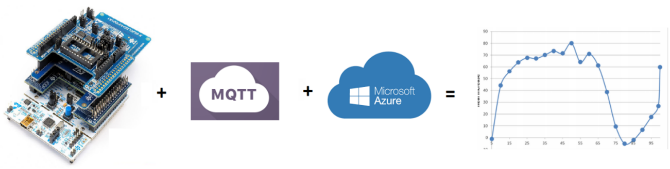

Figura 1. Soluzione end-to-end

Capitolo 1

Introduzione

1.1 Internet of things

Internet of Things, in italiano Internet delle cose, è un neologismo riferito all'estensione di Internet al mondo degli oggetti e consiste tecnicamente in una rete di dispositivi connessi, *non necessariamente aventi una "vocazione digitale"*, identificati univocamente da indirizzo IP e dotati di tecnologie che consentono loro di interagire con l'ambiente circostante raccogliendo dati utili al fine di comunicarli in rete. Un esempio applicativo, in ambito cittadino, può essere un rilevatore collocato in una strada in grado di controllare il corretto funzionamento dei lampioni, oppure raccogliere informazioni sulla qualità dell'aria o sulla presenza di persone. [1]

L'Internet of Things rappresenta uno tra i contesti tecnologici più innovativi, ed anche le stime più conservative prevedono tassi di crescita annui a doppia cifra. Si prevede infatti che nel 2020 i dispositivi connessi ad internet supereranno i 50 miliardi di unità. [1]

IoT non è solo hardware, è anche Cloud Computing. Esso infatti attraverso le sue caratteristiche più tipiche quali: elasticità, scalabilità, potenza ed economicità, sta sostanzialmente abilitando il mondo IoT. L'aspetto principale, che rende così interessante il connubio tra cloud e IoT, è sicuramente la quantità di dati che l'IoT è in grado di generare e che possono essere analizzati dal cloud al fine di trarre conclusioni rilevanti sugli utenti, tendenze di mercato e così via (si noti un ulteriore collegamento con "Big Data & Analytics"). Ed è proprio qui che l'IoT viene naturalmente sorretto dalla potenza e flessibilità del cloud. [2]

Dal punto di vista del marcato, IoT rappresenta una grossa opportunità di business per le grandi industrie produttrici di hardware. Infatti, la proiezione di crescita del mondo IoT è stimata ad affermarsi intorno ai 8,9T di dollari entro il 2020, raggiungendo un tasso di crescita composto del 19,92% (Compound Annual Growth Rate, CAGR).

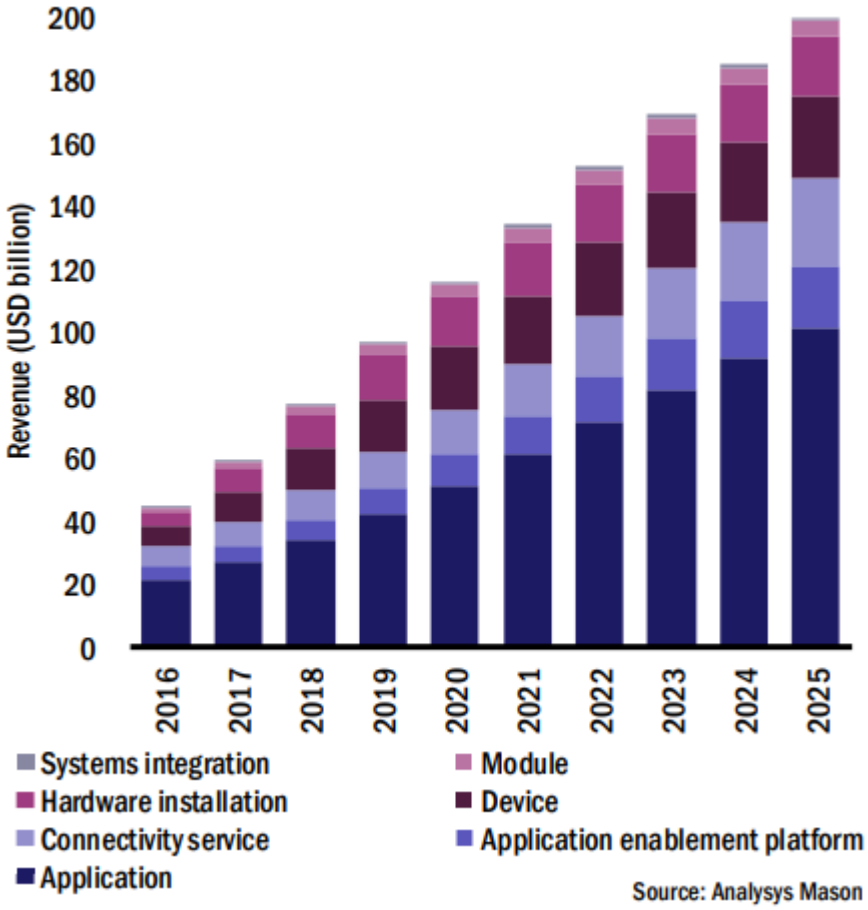

Figura 1.1. Total value chain revenue per IoT, 2016-2025 [3]

1.2 Cloud computing

Il cloud computing, in parole semplici, consiste nella messa a disposizione di servizi di calcolo, come server, risorse di archiviazione, database, rete, software, analisi e molto altro, tramite Internet. Le società che offrono queste funzionalità sono dette provider di servizi cloud, e in genere addebitano un costo in base all'utilizzo che l'utente finale ne fa, in modo analogo alle spese domestiche per acqua o elettricità.

In questo ambito, esistono tre paradigmi di cloud computing, ognuno dei quali si colloca su un differente livello di astrazione delle risorse messe a disposizione:

- *Software as a service (SaaS)*: è un modello di distribuzione del software applicativo dove un produttore software sviluppa, opera (direttamente o tramite terze parti) e gestisce una applicazione web che mette a disposizione dei propri clienti via internet previo abbonamento. Si tratta spesso, ma non sempre, di servizi di cloud computing. [4]
- *Platform as a service (PaaS)*: attività che consiste nel mettere a disposizione piattaforme di elaborazione in termini di hardware e software. Gli elementi del PaaS permettono di sviluppare, sottoporre a test, implementare e gestire le applicazioni senza i costi e la complessità che ne deriverebbero nel caso di acquisto, configurazione e gestione dell'hardware e del software necessari. [5]
- *Infrastructure as a service (IaaS)*: è un'attività economica con la quale un vendor terze parti fornisce un'infrastruttura IT altamente automatizzata e scalabile (archiviazione, hosting, elaborazione dati, networking) fatturando solo l'effettivo utilizzo della infrastruttura messa a disposizione.

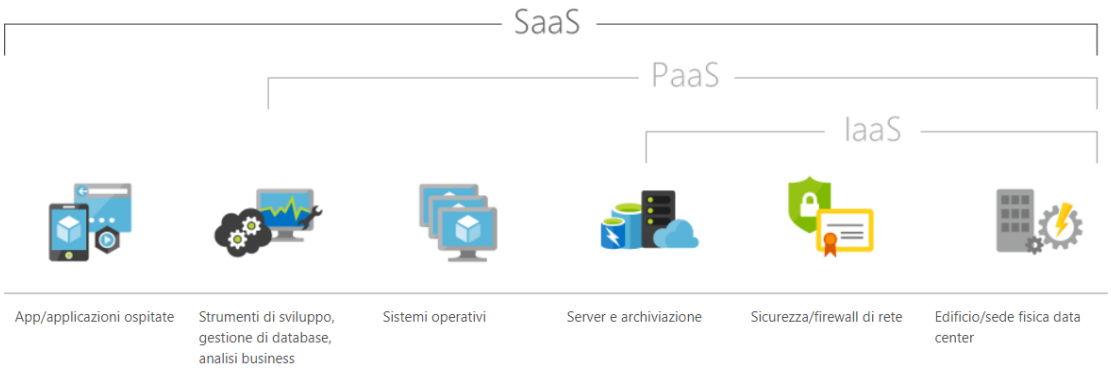

Figura 1.2. Paradigmi SaaS, PaaS e IaaS [6]

Ad oggi, i due principali Cloud Providers sono Microsoft Azure e Amazon AWS. Infatti, secondo un'indagine condotta da RightScale (azienda che opera nelle vendite di software as a service, SaaS), su 997 aziende intervistate è risultato che Azure e AWS insieme ricoprono più del 60% del mercato di cloud computing (Figure 1.3).

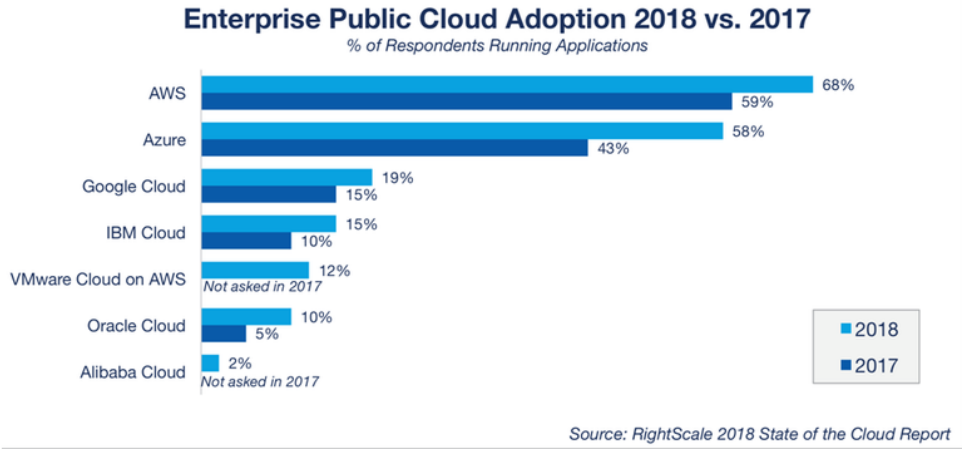

Figura 1.3. Risultati sondaggio RightScale [7]

1.3 Piattaforme di sviluppo hardware

Il mercato delle piattaforme di sviluppo hardware oggigiorno offre agli sviluppatori di applicazioni embedded una vasta gamma di soluzioni a prezzi accessibili.

A seguito del successo delle schede Arduino, i principali silicon vendor hanno lanciato sul mercato diverse piattaforme Hardware/Software per microcontrollori con l'obbiettivo abbattere le barriere di ingresso (in termini di costi e complessita' di utilizzo) per gli sviluppatori di soluzioni embedded. Con il proliferare e l'integrazione dei moduli di connettivita' (Bluetooth Smart, Wi-Fi, LoRA, Sigfox, etc.), queste schede sono diventante l'elemento portante nel design ed implementazione di applicazioni e soluzioni IoT; integrando infatti nel Firmware librerie ed sistemi operativi RealTime direttamente gestiti dai cloud providers (link a Microsoft SDK e Amazon FreeRTOS), queste schede permettono in pochissimo tempo di collegare microcontrollori, sensori ed attuatori, con le funzionalita' e servizi cloud, abilitando la prototipazione rapida id soluzioni end-to-end device/cloud.

La tabella seguente fornisce una overview delle schede di sviluppo offerte dai maggiori silicon vendor.

Vendor	Piattaforma	Ambiente di sviluppo	Conn.	MCU/MPU	Prezzo	Cloud
Intel	Galileo	Visual Studio, Arduino	Eth	Intel Quark CPU 256 MB, DDR3, SRAM 512KB, Flash 8MB	$60, $80	IBM IoT, AT&T M2X, Intel Cloud, Microsoft Azure
Arduino	WiFi Shield	Arduino	WiFi		$87	IBM IoT, AT&T M2X, Temboo, etc.
Qualcomm	Gobi. IoE dev kit	Java Embedded ME	WiFi, GSM e 3G, GPS	QSC6270 chipset, 128MB RAM, lights and enviromental sensors	$500	Etherios
TI	Beaglebone		Eth	1Ghz Stara ARM Cortex-A8, uHDMI, uSD, USB, built-in 4Gb storage	$49	Same as Launch-Pad
Renesas	RX62N		Eth	100Mhz 32bit RX MCU with FPU, USB, CAN, Flash 512KB	$90, $100	Exosite, Free-board.io, Dweet.io

1.4 Lavoro di tesi

1.4.1 Overview

STMicroelectronics, azienda leader nel mercato della produzione di componenti elettronici a semiconduttore, si è riposizionata negli ultimi anni su due mercati principali di riferimento, Smart Driving e IoT:

- *Smart Driving*: rientra in questo ambito tutta l'elettronica (sensori, attuatori, microcontrollori) dedicata ai clienti automotive: controllo motore, sistemi avanzati di assistenza alla guida e guida autonoma, dispositivi di safety, dispositivi per electric charging etc.

- *Internet of things*: data la vastità e la frammentazione del mercato IoT, rientrano in questo ambito la maggior parte dei clienti ST, dai key accounts alle decine di migliaia di SME/start-ups servite attraverso i canali di distribuzione. Rientrano nel segmento IoT tutte le famiglie di microcontrollori a 32bit (STM32), sensori (accelerometri, sensori prossimità, microfoni, etc.), attuatori (controllo motore), connettività low power (SubGhz, Bluetooth Smart, Sigfox etc.).

Per sfruttare le potenzialità del mercato IoT e per incrementare l'utilizzabilità dei componenti ST (in primis microcontrollori e sensori) in particolare per piccole aziende, startups e makers, ST ha progressivamente introdotto dal 2014 delle schede general purpose di prototipazione: STM32 Nucleo, expansion boards e IoT Discovery Kit. Contestualmente, ST ha sviluppato e rilasciato pacchetti FW a corredo delle schede di prototipazione che permettono di implementare e testare in poco tempo complesse funzionalità (www.st.com/stm32ode). In particolare dal 2016 sono stati sviluppati pacchetti firmware che permettono di collegare le schede di prototipazione ST direttamente con i principali Cloud Providers (Amazon AWS, Microsoft Azure, IBM Watson, GCP). A corredo del firmware per il collegamento con Microsoft Azure, ST ha pubblicato anche una dashboard web, applicazione sviluppata sfruttando i servizi PaaS offerti da Azure, con l'obiettivo di rendere ancora più semplice monitorare, configurare e interagire con i dispositivi ST connessi

1.4.2 Motivazioni

Perché una dashboard web?

La ragione principale, per la quale ST ha iniziato a sviluppare dashboard web per i propri dispositivi, è nata dalla necessità di avere un'applicazione in cloud mass market, fortemente personalizzata per i firmware ST, e generica nelle funzionalità offerte, che permetta agli utilizzatori delle tecnologie ST di iniziare a prototipare soluzione IoT complete (dal sensore all'applicazione cloud) in pochissimo tempo. Lato cliente infatti l'obiettivo è facilitare lo sviluppo di applicazioni IoT basate su sistemi ST, al fine di arrivare a fare prototipazione anche nel caso in cui non si abbiano competenze di sviluppo in cloud, o banalmente non si abbia intenzione di creare un account Azure a pagamento. A tal proposito, la soluzione ST si propone infatti di offrire un servizio gratuito e pronto a l'uso.

Oltre a questo, si aggiunge una motivazione interna all'azienda in quanto, attualmente, ST non possiede ancora, in particolare nel settore marketing/vendite, competenze tecniche specifiche sullo sviluppo di applicazioni cloud che permettano promuovere efficacemente HW and FW dedicato alla connettività con i cloud providers. La possibilità di avere soluzioni web sul cloud che possano facilmente dimostrare le potenzialità dell'HW ST, rappresenta un ulteriore punto a favore dal punto di vista marketing.

Il codice sorgente del firmware e delle applicazioni web è disponibile in open-source; lato cloud questo offre la possibilità ai clienti di replicare su una propria sottoscrizione Azure l'intera soluzione e di personalizzarla a piacere. Questo ultimo aspetto apre inoltre interessanti potenzialità di partnership con cloud providers (Microsoft) e distributori di HW e licenze per l'utilizzo cloud (Avnet); sviluppare applicazioni web dedicate per l'harwdare ST e facilmente clonabili su account personali, permettere infatti di poter partecipare in revenue sharing dei ricavi generati dalla distribuzione di licenze Azure, nel momento in cui queste sono direttamente generate dalla soluzione end-to-end sviluppata da ST.

Figura 1.4. AVNET Silica, STMicroelectronics e Microsoft partnership

Stato dell'arte

La prima versione della dashboard è disponibile online all'indirizzo stm32ode.azurewebsites.net. Si prevede un primo rilascio della nuova versione, sviluppata durante il percorso di tesi, in Q4 del 2018 (Ottobre/Novembre 2018).

L'attuale applicazione web si compone principalmente di tre viste: accesso, telemetria e gestione del dispositvo.

Accesso

In questa pagina è presente un form di input in cui bisogna inserire l'indirizzo MAC del dispositivo che si vuole monitorare.

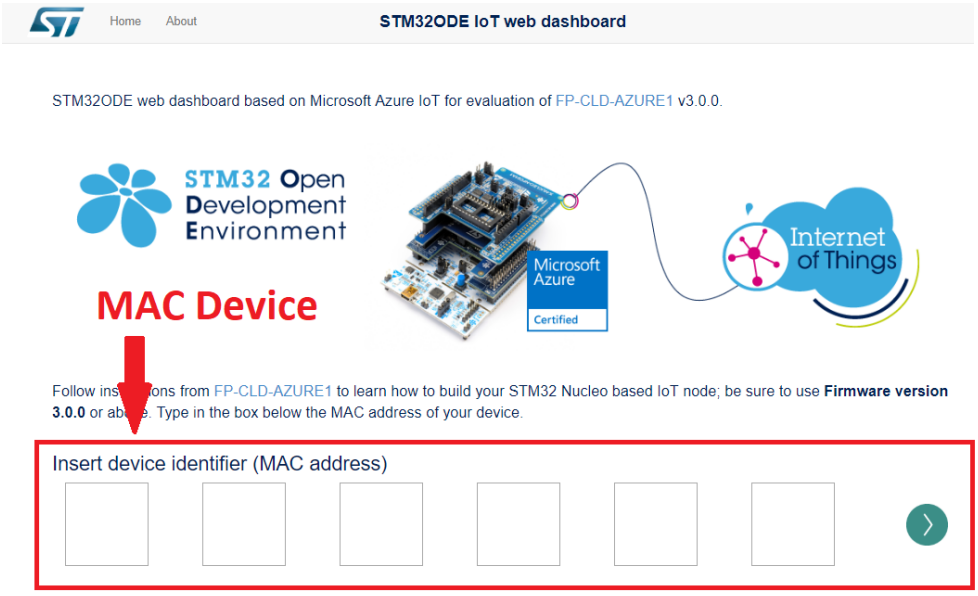

Figura 1.5. Vista input form

Telemetria

In questa pagina è possibile controllare se il dispositivo in questione è connesso o meno, e osservare in real-time i dati di telemetria che sta inviando. I tipi dato supportati sono quattro: temperatura, pressione, umidità e dati accelerometrici; questi sono generati tipicamente da dispositivi che montano expansion boards MEMS oppure basati sulle schede di sviluppo integrate IoT Discovery.

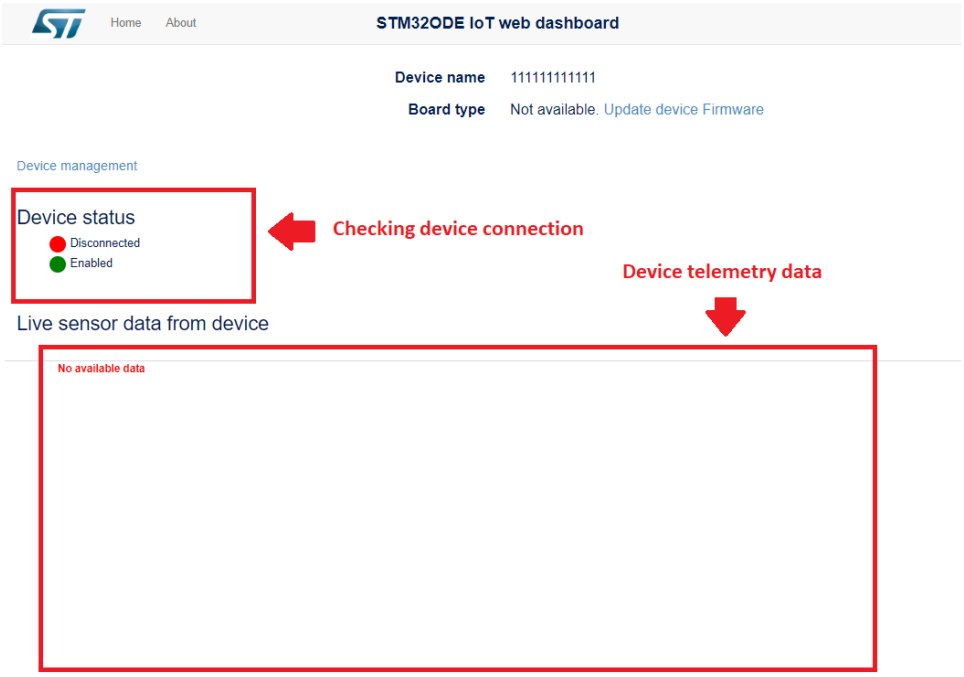

Figura 1.6. Vista telemetria dispositivo

Gestione dispositivo

In questa pagina l'utente è in grado di configurare il dispositivo editando un apposito file JSON renderizzato graficamente. Può inoltre interagire con esso: a seconda delle funzionalità supportate si è in grado di inviare comandi oppure invocare metodi direttamente sul dispositivo. Ad esempio, l'utente può selezionare (previo caricamento) un nuovo firmware da installare e invocare il metodo che realizza l'operazione di firmware update.

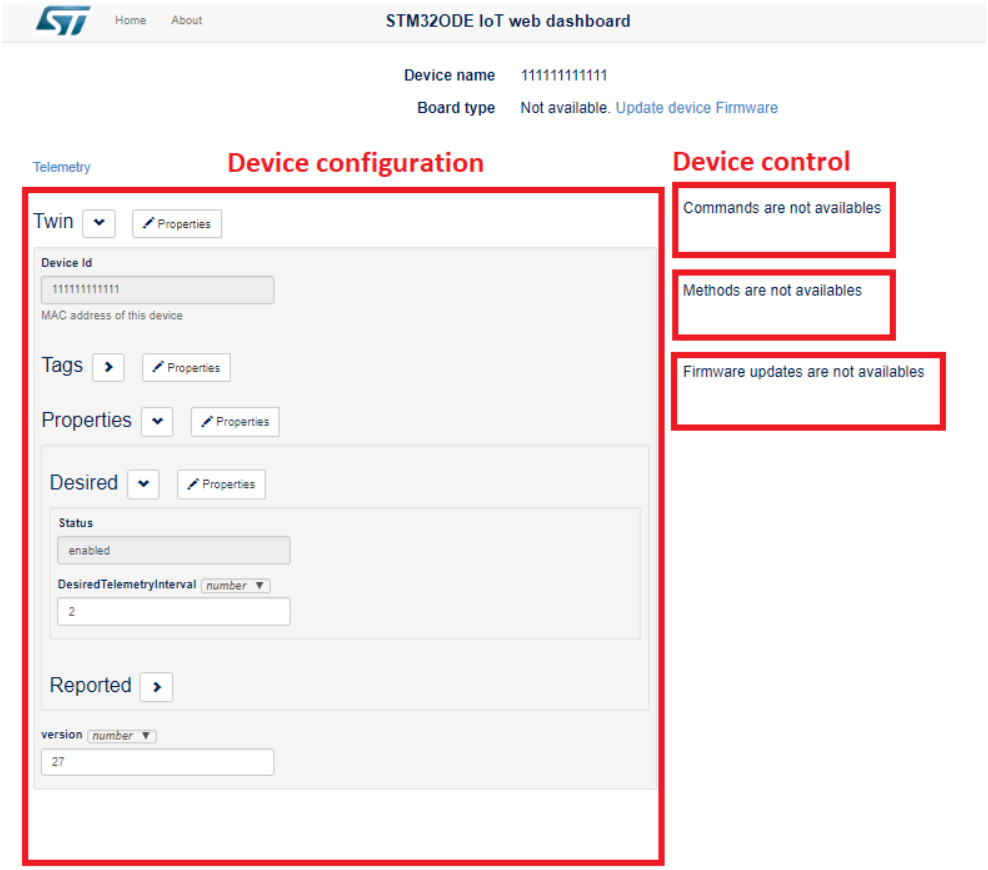

Figura 1.7. Vista gestione dispositivo

Perchè una *nuova* dashboard?

Le potenzialità offerte da questo prodotto sono enormi, per sfruttarle al massimo non resta che migliorare ciò che è stato già realizzato.

Avere una applicazione professionale in cloud con un'ampia gamma di funzionalità offerte e una migliore esperienza utente, rappresenta infatti per i clienti una ragione in più per scegliere di utilizzare soluzioni ST. Durante la fase di planning del lavoro di tesi, si è speso un po' di tempo ad individuare quelle nuove funzionalità da implementare in grado di rendere la prototipazione ancor più facile e intuitiva lato utente: una descrizione del device graficamente più chiara e immediata, dà all'utente un'indubbia sensazione di effettivo controllo del device connesso. Altre funzionalità come: visualizzare lo storico dei dati inviati dai dispositivi e poter implementare su questi meccanismi di notifica utente sono di obiettiva utilità in molti ambiti applicativi. Avere quindi una dashboard più attrattiva permette di quindi

di promuovere in modo più efficace la tecnologia ST, aumentando quindi i ricavi dirette legate alla vendita dell'HW, dando ai clienti strumenti più efficaci di prototipazione; permette inoltre di avere soluzioni IoT sul cloud in grado di aumentare le possibilità di revenue sharing con i partner cloud e distributori.

Infine, l'aggiunta di un profilo utente che mancava nella prima soluzione, permette di poter profilare gli utenti che utilizzano la dashboard; questo risulta estremamente prezioso per migliorare il marketing aziendale, in quanto permette di capire quando e quali componenti ST sono stati utilizzati da quale customer ed in che modo.

Capitolo 2

Ambiente di sviluppo

2.1 STM32 Open Development Enviroment

Open Development Environment (ODE) è un ambiente di sviluppo offerto da STMicroelectronics, per la realizzazione e prototipazione di applicazioni basate sulla famiglia STM32 di microcontrollori a 32-bit, combinati con altri componenti ST state-of-the art connessi per mezzo di schede di espansione. [8]

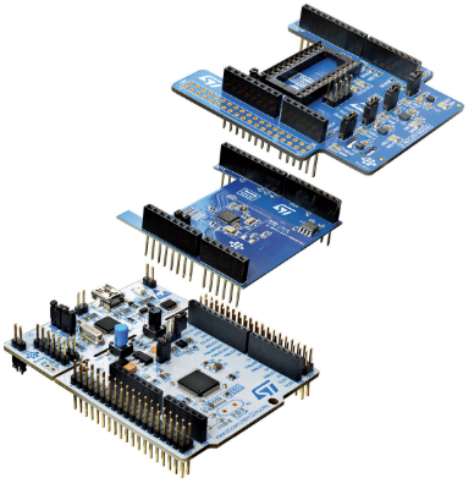

Figura 2.1. STM32 base board (Nucleo) ed expansion boards

STM32 ODE si compone principalmente di cinque moduli fondamentali: (Figure 2.2)

- *STM32 Nucleo development boards*: gamma di schede di sviluppo per la famiglia STM32, con capacità di espansione illimitata e debugger/programmatore integrato.
- *STM32 Nucleo expansion boards*: schede aventi funzionalità addizionali, come: rilevamento, controllo, connettività, alimentazione, audio processing etc. Più complesse funzionalità possono essere ottenute combinando più schede mediante i connettori integrati.
- *STM32Cube software framework*: set di tools ed embedded softwares gratuiti, che abilitano lo sviluppo di applicazioni su sistemi STM32.
- *STM32Cube expansion software*: software gratuito e compatibile con STM32Cube framework, che abilita lo sviluppo di applicazioni con schede di espansione STM32 Nucleo.
- *STM32Cube Function Packs*: pacchetti software che implementano funzionalità utili nei più comuni contesti applicativi. Sono sviluppati sfruttando la modularità e interoperabilità delle schede di sviluppo e espansioni Nucleo STM32, mediante il software STM32Cube.

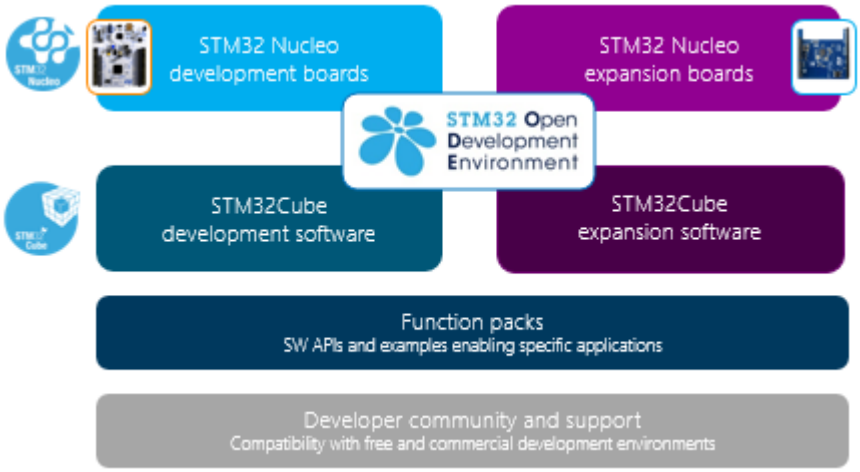

Figura 2.2. STM32 Open Development Enviroment

2.1.1 Development boards

All'interno del programma STM32 ODE, le piattaforme di sviluppo hardware, selezionate per essere utilizzate insieme alla dashboard web, sono state la STM32 Nucleo e l'IoT Discovery Kit.

STM32 Nucleo

La scheda di sviluppo STM32 Nucleo (Figure 2.3) permette di realizzare e testare applicazioni firmware per la famiglia di microcontrollori STM32. La scheda integra un ST-Link debugger, e può essere estesa con schede di espansione grazie ai connettori Arduino UNO R3 compatible e Morpho integrati. [9]

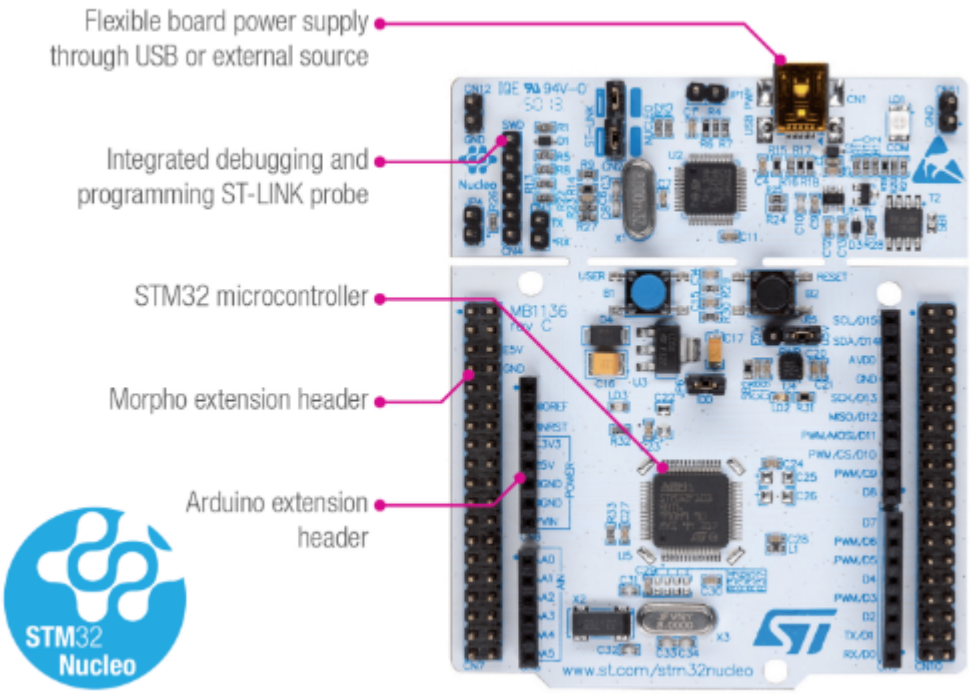

Figura 2.3. STM32 Nucleo Development Board

IoT Discovery Kit

La scheda di sviluppo STM32 IoT Discovery (Figure 2.4) permette di realizzare e testare applicazioni firmware per la famiglia di microcontrollori STM32. La scheda integra un ST-Link debugger, e può essere estesa con schede di espansione grazie ai connettori Arduino UNO R3 compatible e Morpho integrati, ma a differenza della STM32 Nucleo, offre un'ampia gamma di sensori e moduli di connettività integrati. [10]

Di seguito sono elencati i moduli e sensori integrati nella scheda:

- Bluetooth® V4.1 module (SPBTLE-RF)
- Sub-GHz (868 or 915 MHz) low-power-programmable RF module (SPSGRF-868 or SPSGRF-915)
- Wi-Fi® module Inventek ISM43362-M3G-L44 (802.11 b/g/n compliant)
- Dynamic NFC tag based on M24SR with its printed NFC antenna
- 2 digital omnidirectional microphones (MP34DT01)
- Capacitive digital sensor for relative humidity and temperature (HTS221)
- High-performance 3-axis magnetometer (LIS3MDL)
- 260-1260 hPa absolute digital output barometer (LPS22HB)

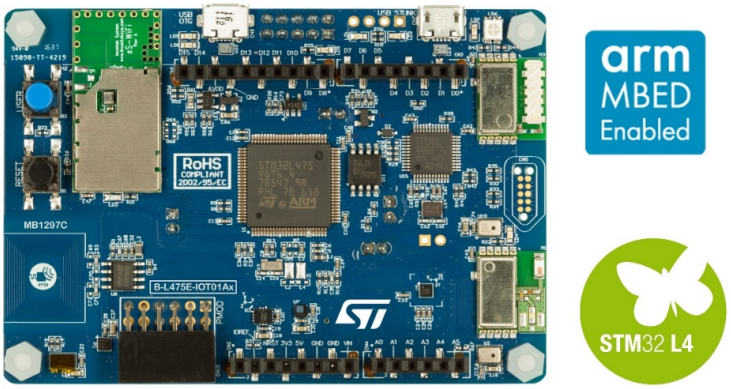

Figura 2.4. IoT Discovery Development Board

2.1.2 STM32 Expansion boards

Le schede di espansione STM32 Nucleo sono disegnate per lavorare assieme alle schede di sviluppo ST sopra citate, e permettono di realizzare applicazioni STM32 full-based. Come per le schede di sviluppo, integrano connettori Arduino UNO R3 compatible e Morpho necessari per combinare più schede tra di loro al fine di ottenere un alto livello di complessità funzionale delle applicazioni. [11]

Le schede di espansione STM32 sono organizzate per tipo:

- *STM32 ODE Connect HW*: schede di espansione che offrono connettività dal Bluetooth al classico Wi-Fi.
- *STM32 ODE Move-Actuate HW*: schede di espansione che integrano drivers per diversi tipi di controllo motore.
- *STM32 ODE Power-Drive HW*: schede di espansione che offrono funzionalità di power management monitoring e conversion per i dispositivi sui quali sono montate.
- *STM32 ODE Sense HW*: schede di espansione che integrano sensori per rilevamento di grandezze ambientali, audio etc.

Tutte le schede di espansione sono supportata dai moduli contenuti nel pacchetto STM32Cube expansion software.

Figura 2.5. STM32 product portfolio

2.1.3 Function Packs

STM32 Function Packs (FP) (Figure 2.6) sono pacchetti software contenenti drivers di basso livello, librerie di middleware e applicazioni di esempio.

Usati in combinazione con le schede di sviluppo ed espansione STM32, oppure integrate come l'IoT Discovery, i function packs offrono numerose funzioni standard utili nei più comuni ambiti applicativi:

- Cloud connectivity
- Networking
- Security

- Sensing

All'interno del dominio Cloud connectivity è stato sviluppato FP-CLD-AZURE1, function pack che ha l'obiettivo di abilitare i dispositivi ST a connettersi ed interagire con la piattaforma cloud Microsoft Azure. [12]

Attualmente FP-CLD-AZURE1 è compatibile con la prima versione della dashboard web. Una nuova versione del firmware è in fase di sviluppo per supportare le funzionalità introdotte nella seconda versione della dashboard, realizzata durante questo percorso di tesi.

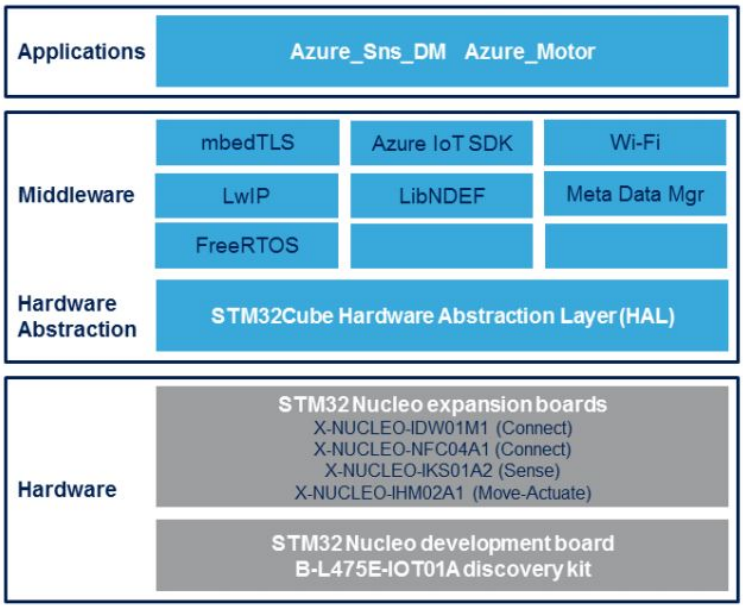

Figura 2.6. FP-CLD-AZURE1 Function Pack

2.2 Microsoft Azure

Microsoft Azure è la piattaforma di cloud computing offerta da Microsoft attraverso la rete globale di datacenters da essa stessa gestita. Azure è in grado di fornire servizi nei tre paradigmi conosciuti: SaaS, PaaS e IaaS; ed ha lo scopo di abilitare i propri clienti allo sviluppo, gestione, testing e distribuzione di applicazioni e/o servizi cloud based.

Figura 2.7. Microsoft Azure Cloud

Microsoft Azure offre più di 600 servizi raggruppati per tipo. Verranno descritti brevemente i servizi di rilievo che sono alla base del funzionamento della applicazione web sviluppata.

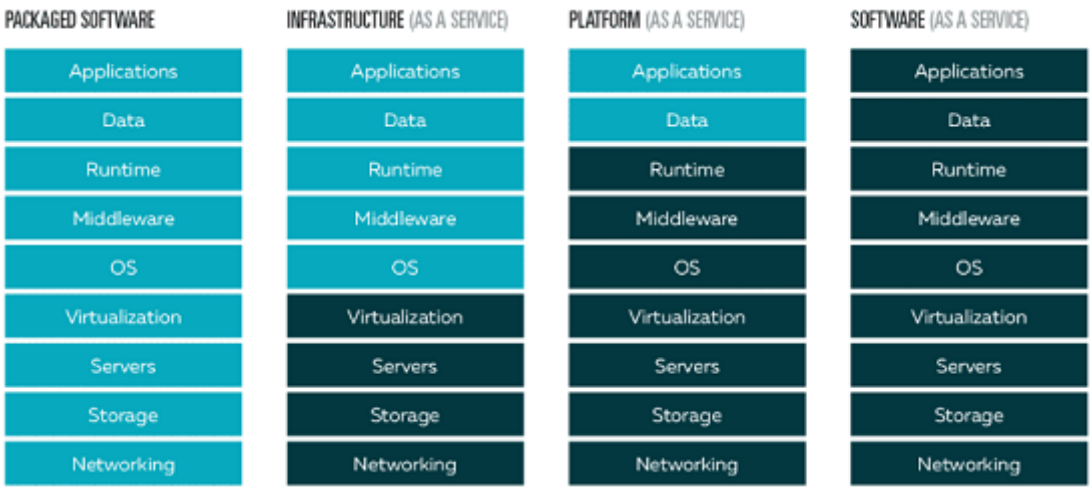

Figura 2.8. Servizi Azure

2.2.1 Azure Web Apps

Web Apps è un servizio offerto da Azure per l'hosting di applicazioni web, REST APIs e mobile back-ends, e supporta un'ampia gamma di tecnologie per lo sviluppo: .NET, .NET Core, Java, Ruby, Node.js, PHP e Python. [13]

È possibile fare load balancing automatico e/o manuale su più nodi, all'interno dell'infrastruttura globale dei datacenters Microsoft, il che rappresenta un valore aggiunto dal punto di vista della robustezza, scalabilità e disponibilità delle applicazioni distribuite.

Per quanto concerne il processo di sviluppo tecnico delle applicazioni, la funzionalità *Continuous integration and deployment* permette di sincronizzare la soluzione distribuita con il repository sorgente di riferimento (GitHub, BitBucket, Docker Hub, Visual Studio Team), automatizzando e velocizzando la realizzazione delle applicazioni stesse.

È offerto un supporto CORS per gli scenari API RESTful, come in questo caso, dove l'applicazione web realizzata è stata strutturata come Single Page Application (SPA).

2.2.2 Azure Functions

Le Azure Functions sono pensate per contesti applicativi serveless, in cui l'obiettivo è eseguire, in cloud, piccoli pezzi di codice senza necessità di eseguire deployment esplicito. Questo scarica l'utente da tutte le problematiche riguardanti l'infrastruttura, vedendosi addebitare dal provider solo il tempo d'esecuzione effettivo impiegato dalla funzione. [14]

Le Azure Functions possono essere scritte in molteplici linguaggi: C#, F#, Node.js, o PHP;[14] e possono essere sviluppate sia in locale, per poi essere distribuite, e sia direttamente sul Web Portal messo a disposizione da Azure.

Tipicamente una Azure Function è utilizzata per interagire con altri servizi cloud attivi sulla propria sottoscrizione. In particolare vi sono tre modalità di interazione: associazioni di input, output e trigger. Un trigger definisce come viene richiamata una funzione, ad esempio all'arrivo di un messaggio presso l'IoT Hub o all'aggiunta di un record in un database. I trigger hanno dei dati associati, ovvero in genere il payload che ha attivato la funzione. [15] Le associazioni di input e output forniscono una modalità dichiarativa per connettersi alle sorgenti dati dall'interno del codice (esempio ad un database). Le associazioni sono facoltative e una funzione può avere più associazioni di input e/o output.

In particolare, l'applicazione web realizzata si serve di tre Azure Functions scritte in C# e Node.js, le quali interagiscono con istanze di Azure Cosmos DB e IoT Hub.

2.2.3 Azure IoT Hub

IoT Hub è un servizio gestito, ospitato in cloud, che agisce da hub di messaggi centralizzato per comunicazioni bidirezionali tra la propria soluzione IoT e i dispositivi da essa gestiti. Permette di scalare su milioni di dispositivi connessi simultaneamente, riuscendo a processare milioni di eventi per secondo. [16]

Prima che un dispositivo possa connettersi, è necessario che la sua identità sia stata registrata nell'*Identity Register* dell'hub. La fase preliminare alla comunicazione vera e propria è detta *Device Provisioning* ed ha lo scopo di creare, per il dispositivo facente richiesta, una entry nell'Identity Register e configurare il relativo *Device Twin*. IoT Hub supporta entrambe le comunicazioni device-to-cloud e cloud-to-device, offrendo la possibilità di realizzare svariati pattern applicativi di messaggistica come: telemetria, upload di file da dispositivo e metodi richiesta-risposta per controllare i dispositivi connessi direttamente da cloud. [16]

I servizi Azure sono stati disegnati per interagire tra di loro: a questo scopo l'IoT Hub consente di instradare i messaggi in arrivo dai dispositivi verso altri servizi Azure, rendendoli così fruibili dalla propria soluzione di back-end. Ad esempio è possibile configurare il sistema in modo tale da consegnare i messaggi ricevuti dall'hub ad una Azure Function avente il compito, a sua volta, di storicizzarli in un apposito database Azure Cosmos DB.

La funzionalità di monitoring integrata, offerta da IoT Hub, permette di tracciare molteplici tipi di eventi che vedono coinvolto l'hub, ad esempio: creazione, registrazione e connessione di un dato dispositivo, oppure guasti del dispostivo durante l'interazione con esso.

Microsoft Azure offre per lo sviluppo su device un SDK dedicato (Azure IoT device SDK) utile per realizzare applicazioni interagenti con l'IoT Hub. Queste librerie supportano molteplici linguaggi di programmazione: C, C#, Java, Python e Node.js; e forniscono differenti protocolli di comunicazione interamente supportati lato cloud: HTTPS, AMQP, AMQP su WebSockets, MQTT, MQTT su WebSockets. Nel caso in cui non si voglia utilizzare i protocolli di comunicazione standard offerti dall'SDK, è possibile estendere direttamente l'IoT Hub in modo tale da renderlo compatibile con i protocolli di comunicazione custom implementati dai propri dispositivi.

Identity register

L'identity register è di fatto un database nel quale sono archiviate informazioni riguardanti i dispositivi a cui è consentito connettersi, comprese dunque le credenziali da essi utilizzate per autenticarsi durante la connessione all'hub.

L'identity register espone API REST allo scopo di: [17]

- Creare l'identità del dispositivo.
- Aggiornare l'identità del dispositivo.
- Recuperare l'identità del dispositivo tramite ID.
- Eliminare l'identità del dispositivo.
- Elencare le identità registrate (al massimo 1000).

- Esportare tutte le identità da un archivio BLOB Azure.
- Importare identità da un archivio BLOB di Azure.

È possibile disabilitare i dispositivi aggiornando la proprietà *status* associata a un'identità storicizzata nell'identity register. Questa proprietà viene generalmente usata in due scenari applicativi: [17]

- Durante un processo di orchestrazione di device provisioning.
- Se si ritiene che un dispositivo sia compromesso o non più autorizzato a comunicare.

Oltre allo stato, il registro riporta un campo denominato *connectionState*, che indica se un dispositivo è attualmente connesso o meno all'hub. Questa informazione è gestita direttamente dall'IoT Hub solo nel caso in cui il dispositivo utilizza MQTT o AMQP come protocollo di comunicazione, e non HTTPS.

L'applicazione web sviluppata fa uso del connectionState, in quanto il firmware ST lavora solo su MQTT.

Device twin

I device twin (Figure 2.9) sono documenti JSON nei quali vengono archiviate informazioni sullo stato dei dispositivi (metadati, configurazioni etc.). L'hub crea un device twin per ogni dispositivo registrato. [18]

L'uso dei device twin abilita il back-end della soluzione a specificare la configurazione voluta per i dispositivi gestiti, anziché inviare comandi specifici. Il dispositivo ha la responsabilità di aggiornare la propria configurazione nel modo migliore in base alla configurazione richiesta da cloud, segnalando continuamente lo stato corrente e le potenziali condizioni di errore del processo di aggiornamento. [18]

In particolare i device twin sono utili per: [18]

- Archiviare metadati dei dispositivi lato applicazione back-end.
- Permettere all'applicazione del dispositivo di segnalare informazioni circa lo stato corrente del dispositivo, ad esempio: il metodo di connettività usato, metodi supportati, etc.
- Sincronizzare lo stato dei flussi di lavoro a esecuzione prolungata tra l'applicazione dispositivo e l'applicazione back-end, ad esempio durante il processo di aggiornamento firmware.
- Eseguire query lato cloud su metadati, configurazione, stato dei dispositivi etc.

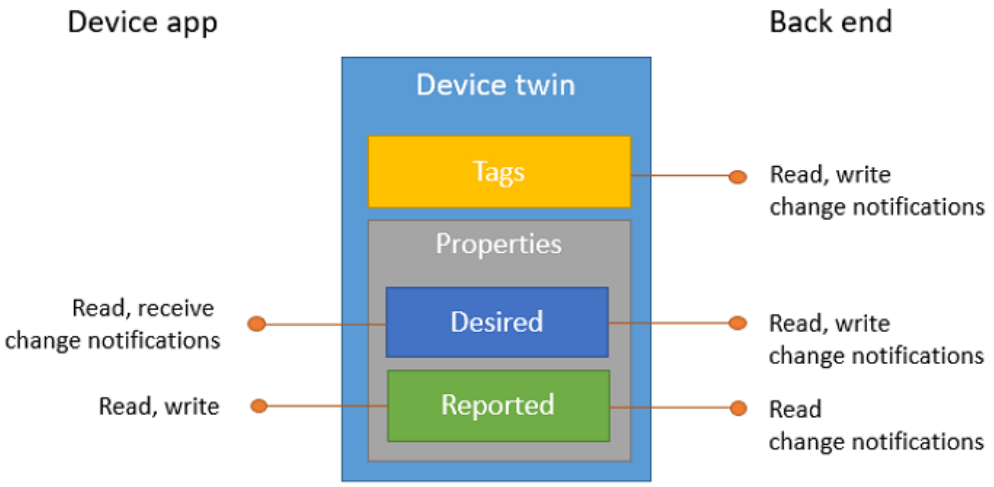

Figura 2.9. Device twin

Un device twin si compone principalmente di tre sezioni: [18]

- *Tag*: metadati dei dispositivi accessibili solo dal back-end della soluzione
- *Desired properties*: rappresenta lo stato del dispositivo desiderato, ed è un oggetto JSON editabile esclusivamente lato back-end e accessibile in sola lettura dell'applicazione del dispositivo.
- *Reported properties*: rappresenta lo stato attuale del dispositivo, ed è un oggetto JSON editabile esclusivamente dall'applicazione del dispositivo e accessibile in sola lettura lato back-end.

Metodi diretti e messaggi cloud-to-device

I *metodi diretti* sono implementati logicamente nel dispositivo, ed accettano zero o più parametri di input. Per richiamare un metodo si utilizza un URI standard sul quale il dispositivo rimane in ascolto (iot/hub/device id/methods/). L'esecuzione avviene in maniera sincrona e può avere esito positivo o negativo, mentre la durata è limitata superiormente da un timeout, il cui valore predefinito è 30 secondi, per un massimo impostabile di 3600 secondi. I metodi diretti risultano adatti per scenari interattivi in cui si vuole che il dispositivo agisca esclusivamente se online, ad esempio nel caso di accensione di un led da remoto. Qui l'esito positivo o negativo deve essere immediato, in modo che il sistema possa agire di conseguenza in base

al risultato. Il dispositivo può restituire facoltativamente dati come risultato del metodo. Il payload, contenuto nelle richieste e risposte dei metodi, è un documento JSON avente dimensione massime di 128 KB. [19]

I *messaggi cloud-to-device* rappresentano il terzo ed ultimo modo, offerto dall'IoT Hub, per interagire lato cloud con i dispositivi connessi. Nello specifico, è possibile inviare messaggi ai dispositivi i quali verranno messi in code loro dedicate, gestite direttamente dall'hub. I messaggi rimarranno in coda finché i dispositivi saranno disconnessi o in generale non saranno in grado di riceverli.

2.2.4 Device Provisioning Service

Il Device Provisioning Service (DPS) di Azure è un servizio di assistenza per l'IoT Hub che realizza il provisioning automatico Just in Time (JIT) dei dispositivi. [20] Questo processo, come già accennato in precedenza, ha lo scopo di abilitare un dispositivo a comunicare con l'IoT Hub, creando una voce nell'identity register e impostando opportunamente il contenuto del twin relativo al dispositivo in essere.

L'automatizzazione del device provisioning è utile per molteplici ragioni: [20]

- Eliminare la necessità di inserire informazioni puntuali hardcoded nell'IoT hub durante la configurazione iniziale.
- Offrire la possibilità di fare load balancing dei dispositivi tra più IoT hub.
- Permettere di connettere un dispositivo all'IoT Hub geograficamente più vicino, garantendo latenza minima durante la comunicazione.
- Permettere di connettere dispositivi a una soluzione IoT specifica a seconda del caso d'uso (isolamento della soluzione).
- Abilitare il provisioning multiplo di uno stesso dispositivo, ad esempio a seguito di una modifica nel dispositivo stesso.

Molti dei passaggi manuali tradizionalmente previsti nel processo di device provisioning diventano automatici utilizzando il DPS, riducendo così il tempo di distribuzione dei dispositivi e il rischio di errori manuali. [20]

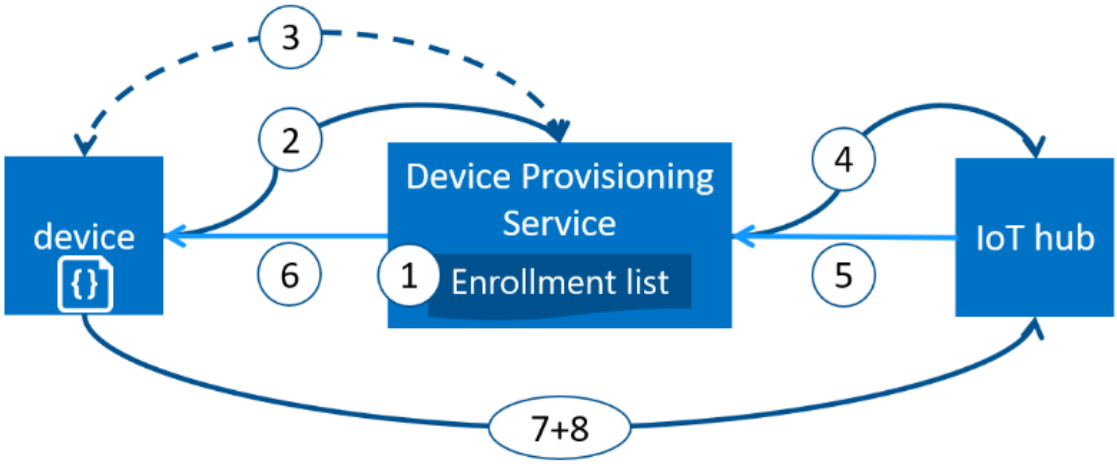

Figura 2.10. Processo di device provisioning

Le operazioni che vengono eseguite durante il provisioning, prima che un dispositivo possa essere registrato, sono: [20]

1. Il produttore aggiunge le informazioni di registrazione del dispositivo nell'enrollment list tramite il portale di Azure (ad esempio impostando un X.509 root certificate in grado di identificare un gruppo più o meno esteso di dispositivi soggetti a provisioning automatico).
2. Il dispositivo contatta l'endpoint esposto dal DPS passando al servizio di provisioning le informazioni necessarie a dimostrare la propria identità.
3. Il DPS, a questo punto, convalida l'identità del dispositivo validando la chiave e ID di registrazione, rispetto alla voce che trova nell'enrollment list. Questa operazione viene svolta tramite una richiesta di verifica nonce (nel caso di TPM) o verifica X.509 standard (nel caso di certificati X.509).
4. Il DPS registra il dispositivo presso un dato IoT Hub e popola il device twin associatogli.
5. L'IoT Hub restituisce al DPS le informazioni circa l'ID del dispositivo.
6. Il DPS, a sua volta, restituisce al dispositivo le informazioni necessarie per connettersi all'hub, presso il quale è stato appena registrato.
7. Il dispositivo si connette all'IoT Hub.
8. Il dispositivo riceve le desired properties definite nel proprio device twin e inizia a comunicare con l'IoT Hub.

Come per l'IoT Hub, il DPS è in grado di interagire con i dispositivi tramite molteplici protocolli di comunicazione: HTTPS, AMQP, AMQP su WebSockets, MQTT, MQTT su WebSockets. Il DPS supporta due tipi di device provisioning: [20]

- X.509 Certificate: basati sul flusso di autenticazione del certificato X.509 standard.
- Trusted Platform Module (TPM): basato su una richiesta di verifica nonce, il quale usa lo standard TPM relativo alle chiavi per presentare un token di firma di accesso condiviso (SAS).

2.2.5 Azure Cosmos DB

Azure Cosmos DB è un database multi-modello distribuito a livello globale da Microsoft, in oltre 50 aree geografiche. È stato progettato per offrire molteplici modelli di dato (a tabella, grafo, documento etc.) con regole di coerenza dati meno rigide rispetto a quelle tradizionali.[21]

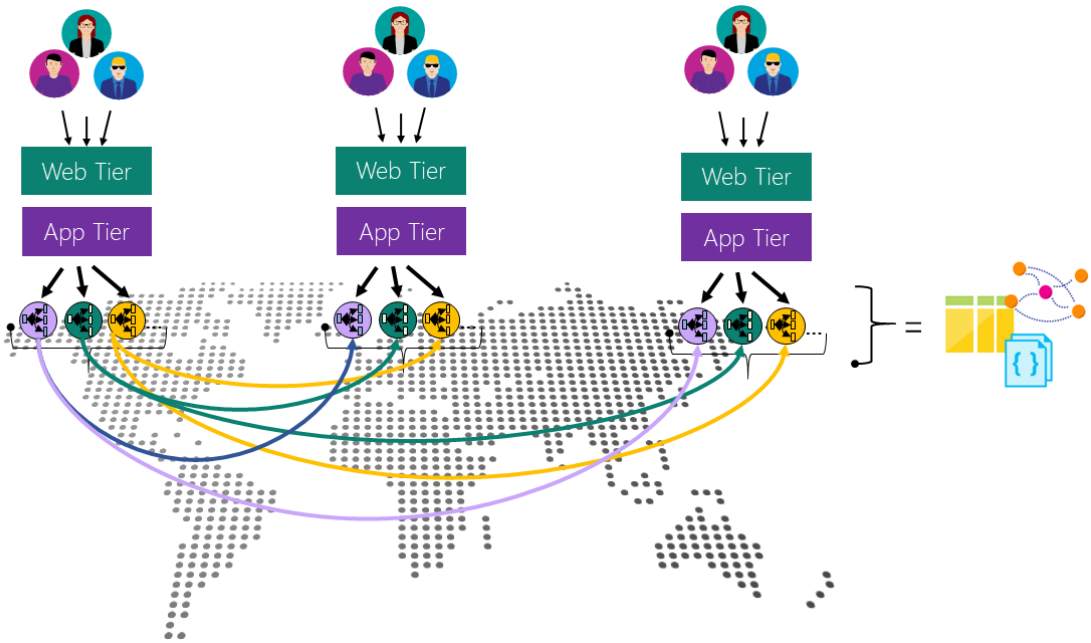

Figura 2.11. Servizio Azure Cosmos DB

Database distribuito

Il vantaggio principale di un database distribuito a livello globale è la bassa latenza di accesso ai dati storicizzati offerta. Le misure di latenza sono stimabili al 99 percentile per molte delle operazioni su database. Il protocollo di replica dei dati usato garantisce che, le operazioni su database, sia in lettura che in scrittura, vengano sempre eseguite nell'area in cui si trova il client. Azure Cosmos DB consente di aggiungere (associare) o rimuovere (dissociare) aree dall'account del database in qualsiasi momento. [21] (Figure 2.11)

Modelli di dati

Azure Cosmos DB indicizza automaticamente tutti i dati senza che sia necessario gestire manualmente indici o schemi. Il modello di dati basato sulla sequenza di record ATOM (ARS), su cui si fonda Cosmos, supporta in modo nativo molteplici modelli dato: a documenti, grafi, coppie chiave-valore, tabelle e colonne. [21]

Le API esposte per accedere ai dati secondo i differenti modelli summenzionati sono: [21]

- *API SQL*: motore di database JSON senza schema con funzionalità di esecuzione di query SQL.
- *API Mongo DB*: MongoDB come servizio a scalabilità elevata compatibile con librerie, driver, strumenti e applicazioni MongoDB esistenti.
- *API Cassandra*: Cassandra come servizio distribuito compatibile con librerie, driver, strumenti e applicazioni Apache Cassandra esistenti.
- *API Graph (Gremlin)*: servizio di database a grafo completamente gestito scalabile orizzontalmente che semplifica la compilazione e l'esecuzione delle applicazioni che usano set di dati a connessione elevata e che supportano le API Open Graph.
- *API Table*: servizio di database di coppie chiave-valore progettato per offrire funzionalità aggiuntive (ad esempio, indicizzazione automatica) alle applicazioni di archiviazione tabelle di Azure senza necessità di apportare modifiche alle applicazioni esistenti.

Coerenza dati

Azure Cosmos DB offre attualmente cinque livelli di coerenza: *assoluta*, *decadimento ristretto*, *sessione*, *prefisso coerente* e *finale*. I modelli di coerenza con obsolescenza associata, sessione, finale e con prefisso coerente vengono definiti "modelli di coerenza meno rigidi", in quanto forniscono un livello di coerenza inferiore alla coerenza assoluta, ovvero il modello di coerenza massima disponibile. [21]

2.2.6 Azure Storage

Azure Storage è un servizio di archiviazione in cloud che permette di storicizzare in modo scalabile oggetti dato, fornire servizio di file system in cloud e offrire un supporto per l'archiviazione di messaggistica in code. [22]

Azure Storage include i seguenti archivi: [22]

- *BLOB*: archivio dati per oggetti di tipo testo e/o binario.
- *File*: archivio file per la condivisione dei dati sia per soluzioni distribuite in cloud Azure e sia per servizi locali.
- *Code*: archivio di messaggistica per comunicazioni affidabili tra i componenti delle soluzioni cloud Azure.

Azure Blob

I BLOB sono ottimizzati per archiviare enormi quantità di dati di tipo testo e/o binario non strutturati. L'archivio BLOB è ideale per: [23]

- Invio di immagini o documenti direttamente in un browser.
- Archiviazione di file per accesso da remoto.
- Flussi audio e video.
- Scrittura in file di log.
- Archiviazione di dati per backup e ripristino.
- Archiviazione di dati a scopo di analisi da parte di un servizio locale o ospitato in Azure.

È possibile accedere agli oggetti storicizzati in archivi BLOB mediante i protocolli di comunicazione HTTP o HTTPS. Gli utenti o le applicazioni client possono accedere ai BLOB tramite URL, API REST, interfaccia Power Shell, console integrata in Portal Azure oppure mediante la libreria client appositamente sviluppata da Microsoft e fruibile in differenti linguaggi (.NET, Java, Node.js, Python, PHP, Ruby). L'archivio BLOB espone tre risorse: account di archiviazione, contenitori e BLOB. Il diagramma seguente mostra la relazione tra queste tre risorse.

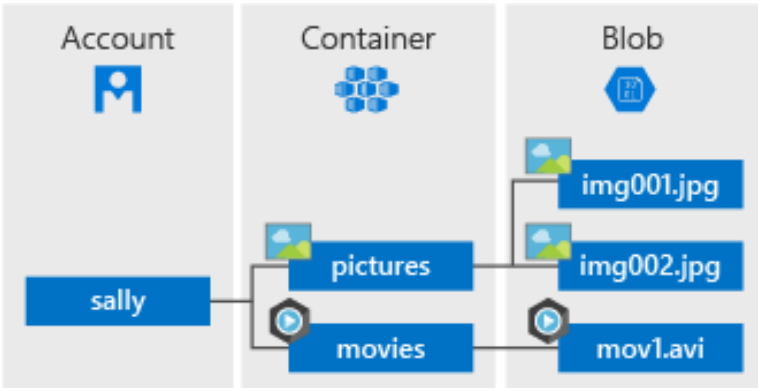

Figura 2.12. Azure BLOB storage

Un contenitore consente di organizzare un set di BLOB analogamente a una cartella in un file system. Tutti i BLOB risiedono in un contenitore. Un account di archiviazione può contenere un numero illimitato di contenitori, ciascuno dei quali può archiviare un numero illimitato di BLOB. [23]

I BLOB in Azure sono disponibili in tre versioni: in blocchi, di aggiunta, di pagine. I BLOB in blocchi archiviano testo e dati binari, fino a circa 4,7 TB e possono essere gestiti individualmente. I BLOB di aggiunta sono costituiti da blocchi, come i precedenti, e sono ottimizzati per le operazioni di aggiunta. I BLOB di pagine archiviano file VHD fino a un massimo di 8 TB ed offrono accesso casuale ai dati. [23]

Azure File

Il servizio Azure File consente di condividere file in rete mediante il protocollo standard di Server Message Block (SMB). Più VMs possono quindi condividere gli stessi file con accesso sia in lettura che in scrittura. È possibile leggere i file usando l'interfaccia REST o le librerie client per l'archiviazione offerte. [22]

Una delle differenze tra la condivisione fornita da File di Azure e quella custom, è la possibilità di accedere ai file da qualsiasi parte del mondo usando un URL. L'indirizzo infatti punta al file e include un token di firma di accesso condiviso. È possibile generare token di firma consentendo l'accesso a uno specifico asset privato per un periodo di tempo specifico. [22]

Archiviazione code

Il servizio di accodamento di Azure viene usato per archiviare e recuperare i messaggi. La dimensione massima dei messaggi nella coda può essere di 64 KB e

può arrivare a contenere milioni di messaggi. Le code vengono in genere usate per archiviare elenchi di messaggi da elaborare in modo asincrono. [22]

2.3 Frameworks e linguaggi di sviluppo in cloud

2.3.1 Back-end

Il back-end dell'attuale applicazione web, è stato sviluppato utilizzando il framework web ASP .NET C#. Durante la fase di planning del lavoro di tesi, si è deciso di abbandonare questa tecnologia a discapito del più recente e moderno framework ASP.NET Core.

Il motivo principale, per il quale si è scelta questa nuova tecnologia, è dato dal fatto che NET Core, essendo multipiattaforma, permette di sviluppare applicazioni compilabili per molteplici sistemi operativi (Windows, Linux e MacOS) facilitando così una eventuale futura migrazione dal server Windows sul quale attualmente è ospitata l'applicazione.

ASP.NET Core è una riprogettazione di ASP.NET 4.x, con modifiche a livello architetturale che lo rendono un framework più efficiente e modulare. Infatti, viene fornito esclusivamente in pacchetti NuGet, il che consente di ottimizzare l'applicazione da sviluppare includendo solo le dipendenze necessarie. È chiaro che una riduzione della superficie occupata dall'applicazione offre anche altri vantaggi, tra cui: una maggiore sicurezza, una riduzione delle esigenze di assistenza e un miglioramento delle prestazioni. Per quanto riguarda il contesto web, a differenza di ASP .NET, la versione Core permette di compilare assieme applicazioni per l'interfaccia web utente con applicazioni web API. [24]

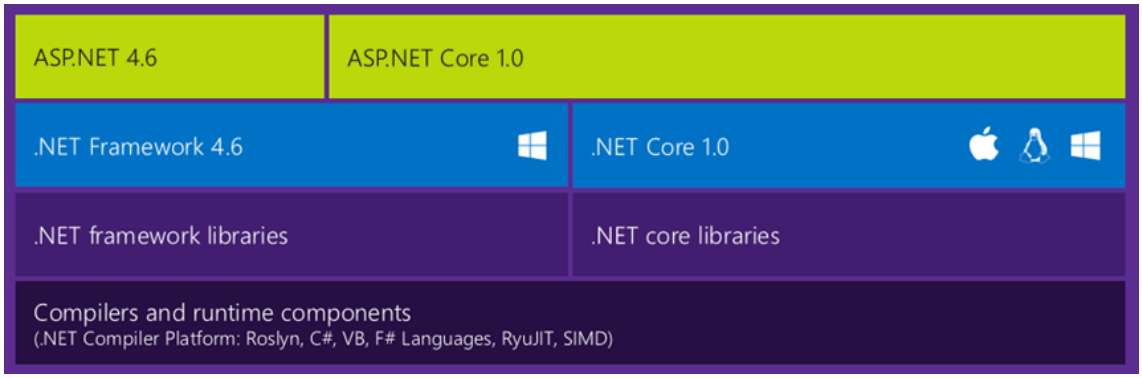

Figura 2.13. ASP .Net e ASP .Net Core

Per l'interazione tra applicazione web e database relazionale è stata utilizzato Entity Framework Core, un mapper relazionale a oggetti (O/RM) sviluppato da Microsoft per l'ambiente .NET Core. [25] Per interagire con i servizi cloud Azure sono state utilizzate le librerie sviluppate e distribuite in pacchetti da Microsoft.

ORM

Object-Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS. [26]

I principali vantaggi nell'uso di tale sistema sono: [26]

- superamento (più o meno completo) dell'incompatibilità di fondo tra il progetto orientato agli oggetti ed il modello relazionale;
- elevata portabilità rispetto alla tecnologia DBMS utilizzata: cambiando DBMS non devono essere riscritte le routine che implementano lo strato di persistenza lato applicazione;
- riduzione della quantità di codice sorgente da redigere;
- favorisce la realizzazione dell'architettura di un sistema software mediante approccio stratificato.

2.3.2 Front-end

Il front-end dell'attuale applicazione web è stato realizzato utilizzando ASP.NET MVC. Questo libreria, sviluppata da Microsoft on top di ASP .NET, rappresenta un'alternativa al vecchio ASP .NET Web Forms e permette di disegnare applicazioni web seguendo il pattern Model-View-Controller (MVC).

Nella soluzione web attuale, il front-end è interamente strutturato a pagine. Anche in questo caso, come per il back-end, si è deciso di abbandonare questo approccio a discapito di una interfaccia utente ibrida: pagine per la sezione di accesso della applicazione, e single page application (SPA) per la dashboard vera e propria. Per lo sviluppo della sezione di accesso è stato utilizzato ASP .NET Core MVC, mentre per lo sviluppo della SPA si è optato per il framework, di recente uscita, Vue.js.

Vue.js è un framework JavaScript che nasce dalle ceneri del vecchio Angular.js, ed è principalmente utilizzato per la creazione di interfacce utente. Negli ultimi anni il numero di contributori allo sviluppo del framework è cresciuto notevolmente, questo a testimonianza del fatto che Vue gode di un notevole interesse tra le comunità di sviluppatori.

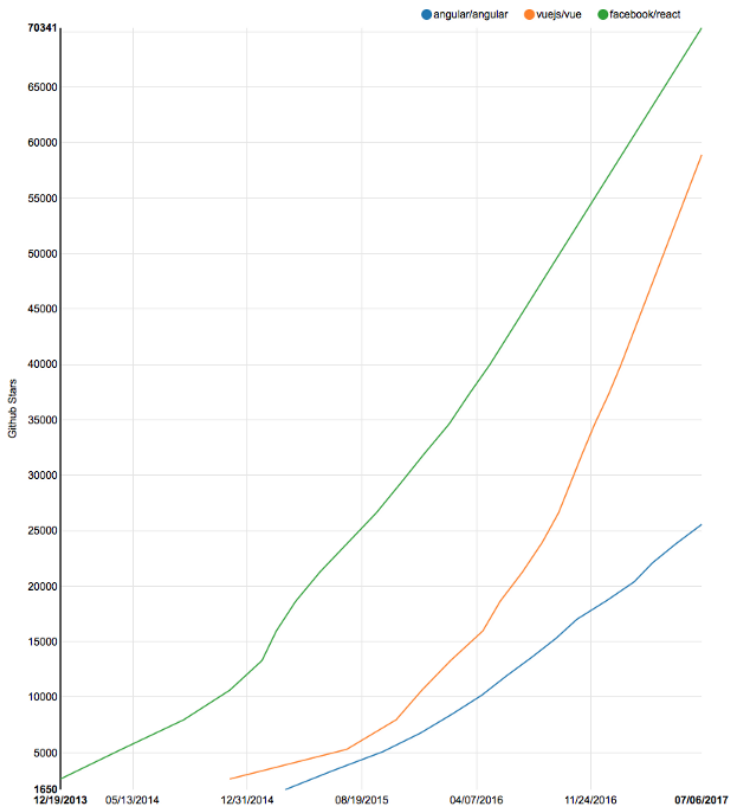

Figura 2.14. Github Stars History di Angular(blue), React(green) e Vue(orange) [27]

Nello scegliere il framework per la realizzazione della SPA, è stata condotta un'analisi comparativa preliminare tra i frameworks open source più conosciuti attualmente disponibili in rete: Angular5, React.js e Vue.js. Dallo studio condotto, si è appreso che Vue presenta una curva di apprendimento nettamente meno ripida rispetto agli altri frameworks (in linea con i tempi di sviluppo prestabiliti per il lavoro di tesi). Inoltre soluzioni sviluppate in Vue presentano una bassa complessità di progetto, rendendolo quindi più leggibile, il che rappresenta un ulteriore valore aggiunto, in ottica futura, quando nuove risorse dovranno prendere parte allo sviluppo della applicazione web. Infine, a differenza del monolitico Angular5, applicazioni sviluppate in Vue sono facilmente integrabili con progetti esistenti e librerie Javascript terze parti.

SPA - Single Page Application

Con Single page application si intende un'applicazione, o sito web, che può essere usata o consultata su singola pagina con l'obiettivo di fornire una esperienza utente

più fluida e simile alle applicazioni desktop dei sistemi operativi tradizionali. In una applicazione SPA tutto il codice necessario (HTML, CSS, Javascript) è recuperato in un singolo scaricamento della pagina, e le risorse addizionali sono caricate dinamicamente e aggiunte alla pagina quando necessario (ad esempio mediante richieste HTTP e/o comunicazioni su WebSockets). [28]

2.4 Metodologia di sviluppo

L'applicazione web realizzata durante il percorso di tesi può essere considerata un progetto software di dimensioni non trascurabili, per questo motivo è stato necessario strutturare il lavoro di sviluppo secondo una tecnica standard tipica dell'ingegneria del software: a tal proposito si è optato per la metodologia Agile, in particolare SCRUM, una tra le più conosciute sotto-pratiche agile. Durante la fase iniziale di planning del lavoro, sono state redatte un insieme di user stories con lo scopo di definire le funzionalità base dell'applicazione. Dopo di che, ogni user story è stata decritta più dettagliatamente stilando la relativa tasks list, riportando tutte le componenti necessarie alla realizzazione della macro funzionalità individuata dalla user story.

As a customer I want to create a personal account so that data generated from my devices is kept private and only visible to me. As a customer I want to be able to visualize and store data generated from my devices to analyse them As provider of the solution I want to be able to collect information about customers and users of the solution for martketing and promotional purposes. As a customer I want an easy way to register and connect my device, so that I can start quickly to visualize, store and analyze data in the web dashboard. As a customer I want to be able to register and manage more than one device so that I can prototype more complex use cases. As a solution provider I want that limit the usage of the service to ST boards, so that technical and development resources are not leaked. As a solution provider I want to better track the usage of ST boards, FW and web-dashboard to understand the reception and acceptance of the solution. As a customer I want a visualize device status and sensors data in professional way, so that I can understand how the final control panel of a custom design will look like. As a customer I want to have an overview of the registered and connected devices, showing the status of the HW, FW, connectivity and location. As a customer, I want to manage the device according to the exact combination of sensors/actuators I'm using. As a customer, I want to edit some simple rules to automate basic actions and notifications when received sensors data is over pre-define thresholds. As a solution provider I want to offer a professional like web service which can attrack customers so that they are motivated to start IoT solutions projects based on ST development boards.

A questo punto, i tasks sono stati inseriti in un backlog di attività, e gli sviluppi organizzati in sprint della durata media di due settimane. A valle di ogni sprint è stato svolto un incontro col team ST con l'obiettivo di dimostrare, mediante demo, il lavoro condotto durante la sessione di sviluppo, e individuare carenze e possibili nuove funzionalità da implementare utili a migliorare la qualità dell'applicazione web.

Metodologia Agile

Per metodologia Agile si intende un insieme di metodi di sviluppo software che prediligono tecniche di lavoro meno strutturate e focalizzate sulla consegna al cliente di software stabile e funzionante, in tempi brevi e frequentemente. Questo approccio va in contrapposizione ai modelli a cascata o in generale ai modelli di sviluppo software tradizionali.

SCRUM

SCRUM è il metodo Agile più diffuso, particolarmente indicato per progetti complessi ed innovativi. Si tratta di un framework che prevede di suddividere il progetto in intervalli temporali ben precisi, della durata di pochi giorni o settimane, detti sprint, entro i quali consegnare un sotto-insieme delle funzionalità del prodotto finale. Queste unità di lavoro fanno parte di sessioni di sviluppo più estese, della durata di settimane o mesi, dette Incrementi, alla fine delle quali avviene la vera e propria consegna della soluzione finale, o parte di essa.

2.5 Tools

Visual Studio

L'intera applicazione web, dal back-end al front-end, è stata realizzata in Visual Studio, ambiente di sviluppo integrato (IDE) realizzato da Microsoft il quale supporta attualmente molteplici tipi di linguaggio. (Figure 2.15)

Tra i tanti tool integrati, Visual Studio due strumenti fondamentali e che sono stati utilizzati per la messa a punto dei sistemi in cloud:

- *Power Shell*: console utilizzata principalmente per manipolare le tabelle contenute nel database relazionale in cloud, e per l'installazione di pacchetti di utilità;
- *Azure tool*: tool utilizzato per il deploy automatico in cloud dell'applicazione web nel servizio WebApp.

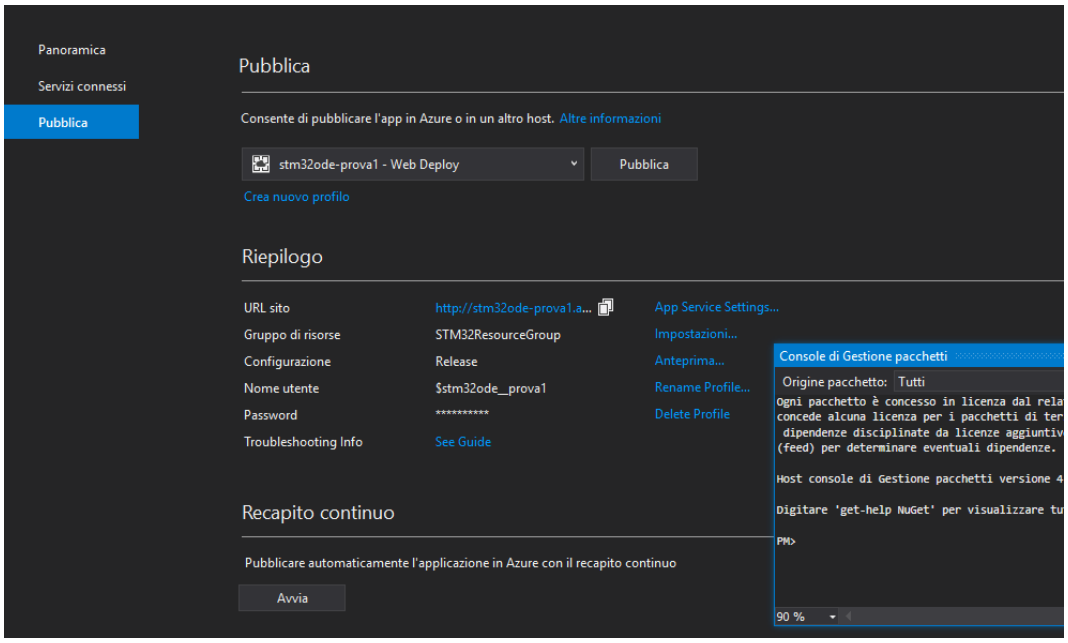

Figura 2.15. Visual Studio

Azure Portal

Applicazione web sviluppata da Microsoft che ha lo scopo di abilitare l'utente a creare, rimuovere, configurare e gestire da remoto i servizi Azure nell'ambito della propria sottoscrizione. ((Figure 2.16))

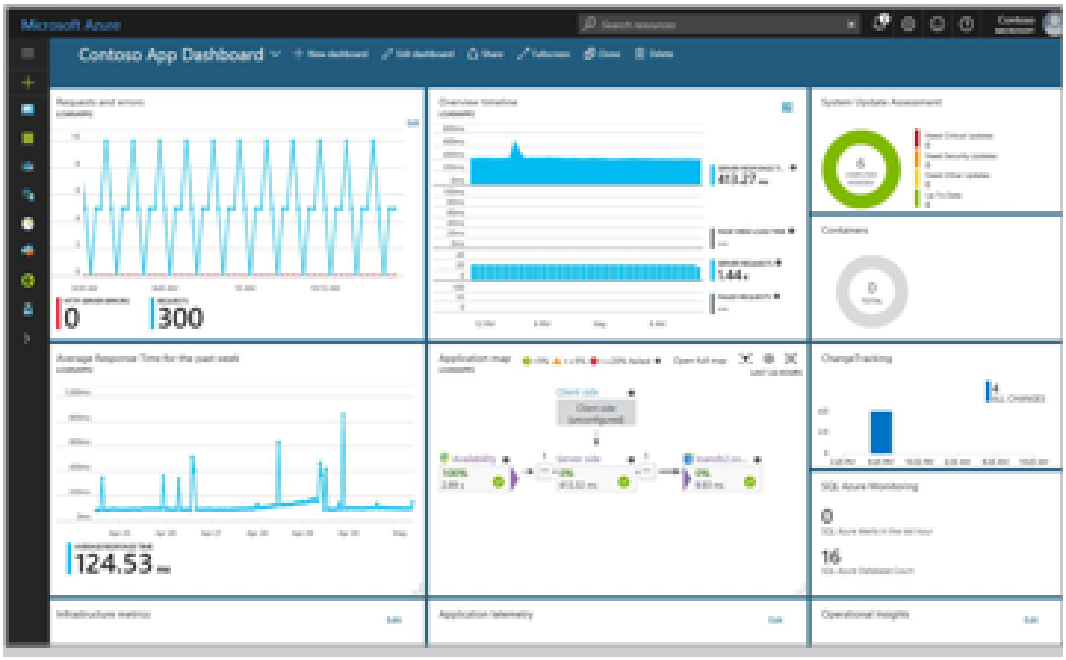

Figura 2.16. Azure portal

Capitolo 3

Architettura

3.1 Stato dell'arte

I servizi cloud utilizzati nella prima versione della dashboard web sono:

- *Azure WebApps*: utilizzato per pubblicare l'applicazione web e il servizio di registrazione device custom.
- *Azure IoT Hub*: utilizzato per comunicare con i dispositivi.
- *Stream Analytcs*: servizio dedicato all'inoltro, previa opportuna trasformazione, dei messaggi ricevuti dall'hub, e all'aggiornamento dei dati contenuti nel database.
- *Azure SQL DB*: database utilizzato per storicizzare informazioni sull'attività dei devices.
- *Azure WebJob*: utilizzato per schedulare processi di background, aventi il compito di monitorare le attività dei dispositivi.
- *Azure BLOB Storage*: servizio di archiviazione utilizzato per storicizzare i firmware caricati dall'utente.
- *Azure EventHub e Azure Service Bus*: servizi di accodamento, utilizzati per rendere più efficienti le comunicazioni tra i servizi cloud attivati.

3.1.1 Device provisioning

Nel 2016 il servizio di provisioning automatico in cloud (DPS) non era ancora disponibile, per questo motivo si è deciso di realizzare una applicazione web API REST che fungesse da servizio custom di provisioning dei dispositivi.

L'obiettivo del processo illustrato in figura, consiste nell'automatizzare la registrazione dei dispositivi presso l'IoT Hub di riferimento, al fine di consegnare loro la stringa di connessione necessaria per la comunicazione.

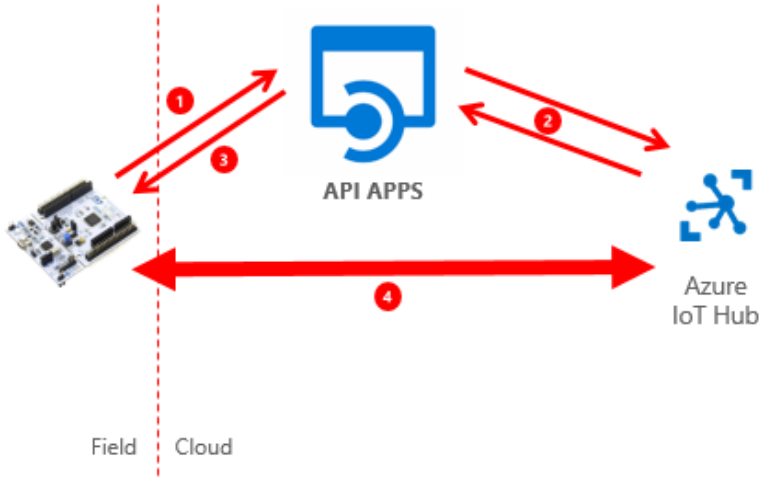

Figura 3.1. Device provisioning scheme

Il processo di provisioning si svolge nella seguente maniera:

1. Il device fa richiesta al servizio di registrazione comunicandogli il proprio identificativo, ovvero l'indirizzo MAC corrente.
2. Contestualmente alla richiesta del device, l'applicazione interagisce con l'IoT Hub:
   (a) Se il device è già registrato presso l'hub, l'applicazione si limita a richiedere la stringa di connessione.del device.
   (b) Se il device non esiste (ciò vuol dire che il dispositivo è la prima volta che si presenta con quell'identificativo MAC), l'applicazione prima di tutto richiede all'hub di creare un'entry associata al device nell'identity register, e poi recupera la stringa di connessione appena creata come fatto nel caso precedente.
3. A questo punto l'applicazione ritorna la stringa di connessione come risposta alla precedente richiesta avanzata dal device.

4. Il device contatta l'IoT Hub mediante la stringa di connessione ottenuta, svolgendo prima un'autenticazione di tipo challenge simmetrica, sfruttando l'identificativo usato per registrarsi, e in caso di successo potrà iniziare a comunicare con l'hub.

3.1.2 Device monitoring

Telemetria

I dispositivi connessi, che supportano la telemetria, inviano dati ad intervalli regolari, secondo un parametro riportato nel device twin configurabile dall'utente tramite dashboard. Il monitoring in real-time di questa attività consiste, di fatto, nella visualizzazione dei dati ricevuti su appositi grafici.

Il dispositivo, una volta registratosi e autenticatosi, inizia a comunicare con l'IoT Hub, il quale funge da punto di ingestione dati, instradando i messaggi ricevuti verso lo Stream Analytics.

Lo Stream Analytics ha il compito di ricevere i messaggi in uscita dall'hub, trasformarli in un formato standard, stabilito in fase di design, coerente con la logica applicativa, e inviarli alla Web App. Inoltre, essendo in grado di accedere al payload dei messaggi, ha il compito di aggiornare lo stato del database in base al loro contenuto.

A questo punto, alla ricezione di ogni messaggio, l'applicazione web aggiornerà opportunamente la pagina relativa al dispositivo monitorato.

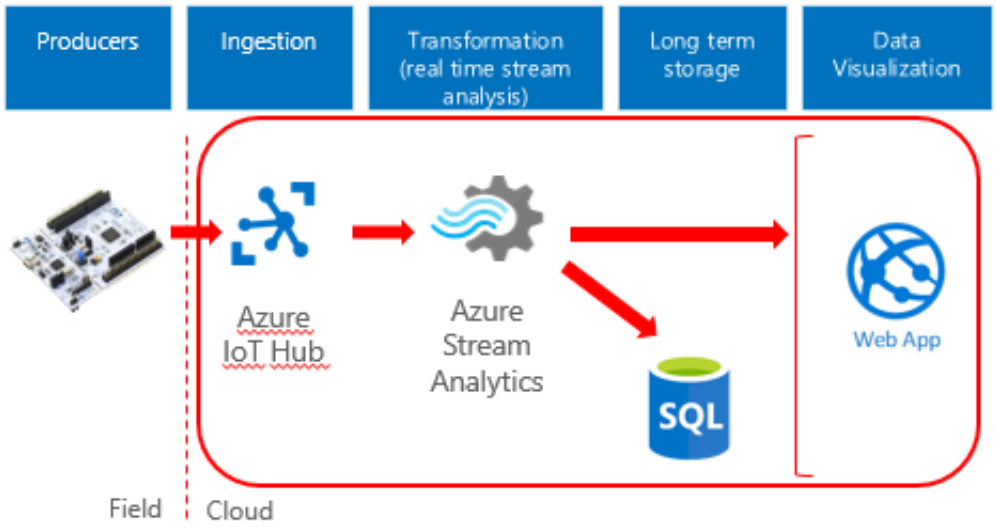

Figura 3.2. Data telemetry flow

Blocco/sblocco device

Identificando univocamente il device tramite indirizzo MAC, ci si è posti dinanzi un problema di dimensioni non trascurabili: tutelare la soluzione da possibili denial-of-service (DOS) generati da dispositivi malevoli.

Si è deciso di porre rimedio a questo scenario schedulando ogni trenta minuti un processo di background, distribuito come Web Job, il cui compito è bloccare, direttamente a livello di IoT Hub, i dispositivi che, nell'ultima mezz'ora di attività, hanno raggiunto una soglia massima prestabilita di messaggi inviati.

In particolare, il processo legge il contenuto del database, tenuto aggiornato dallo Stream Analytics, al fine di controllare lo stato di attività di ogni device. Per i dispositivi selezionati come 'da bloccare', il processo richiede all'IoT Hub di rifiutare connessioni entranti da essi e di abbattere quelle attualmente attive.

In tutto questo, l'applicazione web viene notificata circa questi eventi tramite appositi messaggi generati dall'IoT Hub. Ogni 24 ore, a mezzanotte, tutti i dispositivi vengono riabilitati sull'IoT Hub, e il contenuto del database resettato.

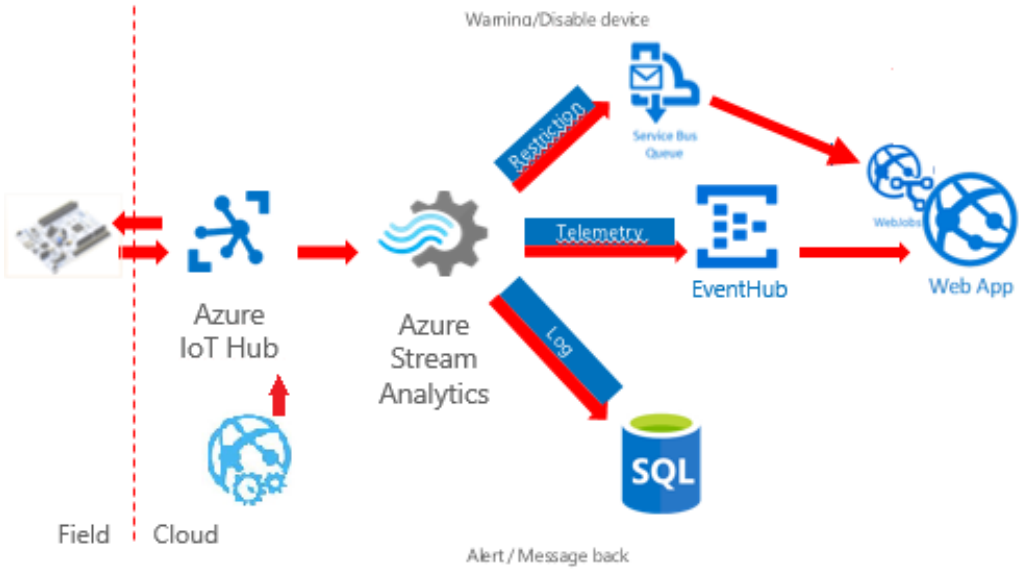

Figura 3.3. Abilitazione/disabilitazione dispositivo

3.1.3 Controllo dispositivi

Per configurare il dispositivo, inviare comandi e invocare metodi, l'applicazione web si serve delle funzionalità offerte dall'IoT Hub.

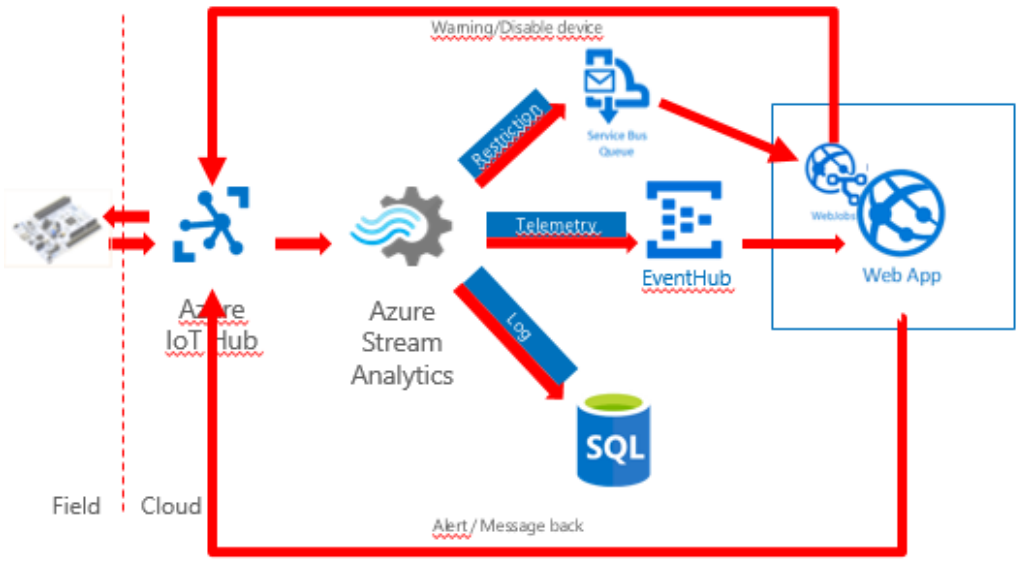

Figura 3.4. Interazione device-cloud

Configurare il dispositivo da web dashboard si traduce nell'aggiornare le desired properties contenute nel twin del dispositivo. Questa sezione è stata pensata da Microsoft proprio per quei casi in cui, lato cloud, si vuole forzare il dispositivo a cambiare il proprio stato.

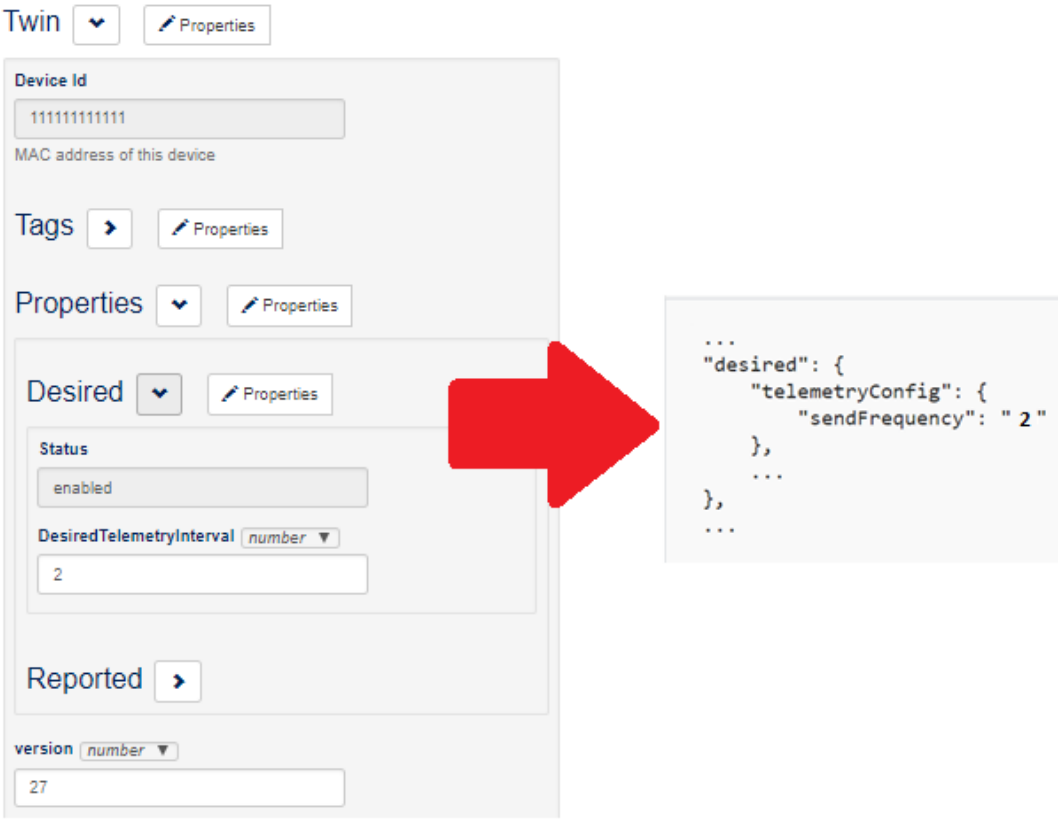

Figura 3.5. Esempio di configurazione twin

Metodi e messaggi cloud-to-device sono funzionalità native messe a disposizione dall'IoT Hub. In fase di design della soluzione, sono state definite alcune funzionalità applicative realizzabili mediante questi strumenti, eccone di seguito alcune:

- *Led on*: accensione di un led su dispositivo.
- *Led off*: spegnimento di un led su dispositivo.
- *Quit*: uscita dall'applicazione del dispositivo.
- *Reboot*: si avvia il reboot del dispositivo.
- *Firmware update*: si avvia il processo di update del firmware del dispositivo.
- *etc.*

In generale, i dispostivi supportano solo un sotto-insieme di queste funzionalità, di conseguenza si è reso necessario dover riportare, lato device, questo tipo di

informazione nel twin associato, in particolare nella sezione reported properties sotto le voci *SupportedMessages* e *SupportedMethods*:

```
"SupportedMethods": {
    "LedOn": 'On a led on device.',
    "ChangeMemory": 'Change memory state.',
    "Quit": 'Quit from application.',
    "PrintMessage--Message-string": 'Display a message.',
    "FirmwareUpdate--PkgUri-string": 'Perform firmware update on device'
},
```

Figura 3.6. Azure portal - Esempio di twin supported methods

Metodi e messaggi supportano il passaggio di parametri, per inferire questo tipo di informazione, nome del comando e nome/tipo dei parametri accettati sono descritti secondo un formato standard. Insieme a questi, il dispositivo riportata anche una breve descrizione del metodo/messaggio human-friendly. Per quei dispositivi che supportano il FOTA e che riportano come metodo supportato il firmware update, all'utente è data la possibilità di caricare su un BLOB storage, attivato in cloud, nuovi firmware da installare sul device mediante invocazione del metodo dedicato.

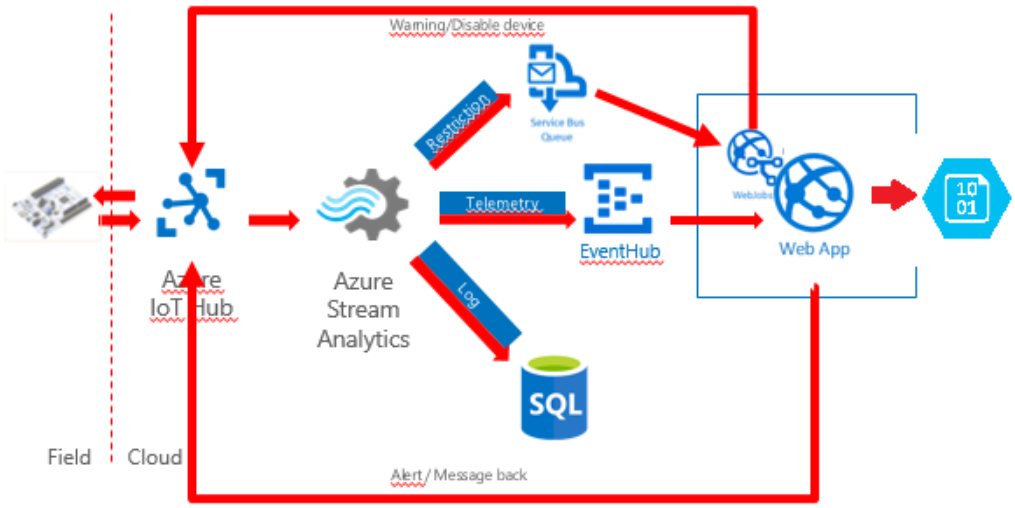

Figura 3.7. Schema architettura cloud completo

3.2 Nuova dashboard web

Nella nuova versione della dashboard web, sviluppata durante il percorso di tesi, si possono individuare principalmente sei macro-sottosistemi:

- User accounting
- Registrazione devices
- Dettaglio del device
- Telemetria
- Regole sull'attività del device

Un utente per utilizzare la dashboard deve necessariamente crearsi un account. Può registrare nel proprio spazio utente uno, o più, dispositivi. All'avvio, il dispositivo si registra presso l'IoT Hub, al fine di comunicare con esso. L'utente può monitorare lo stato, controllare e configurare il proprio dispositivo dall'apposita vista dei dettagli. Sempre da dashboard, l'utente è in grado di osservare i dati di telemetria inviati dal dispositivo nelle ultime 48 ore, opportunamente categorizzati per tipo (umidità, temperatura, etc.). Infine per ogni dispositivo, può specificare una o più regole su un insieme di dati telemetrici, di dimensione opzionale, volte a notificare via email l'utente stesso nel caso in cui vengano soddisfatte.

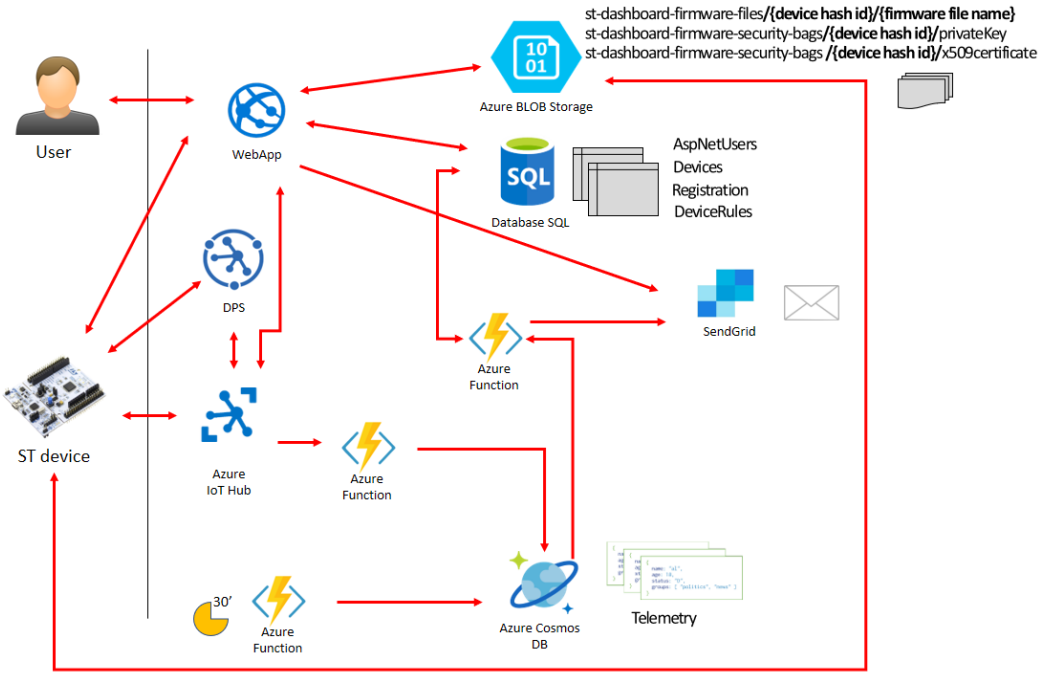

Figura 3.8. Nuovo schema architettura cloud completo

3.2.1 User accounting

Di seguito sono riportate le funzionalità offerte dal sistema di user accounting in termini di user story:

As a customer I want to create a personal account so that data generated from my devices is kept private and only visible to me. As provider of the solution I want to be able to collect information about customers and users of the solution for martketing and promotional purposes

Back-end

Il meccanismo di user accounting è stato realizzato lato back-end utilizzando ASP .NET Core Identity, modulo software sviluppato da Microsoft on top del framework ASP .NET Core.

Identity è un sistema di membership che consente di aggiungere funzionalità di accesso utente all'applicazione web. Sono forniti una serie di meccanismi, già sviluppati e pronti all'uso, per abilitare gli utenti a crearsi un proprio account ed effettuare il login con username e password, oppure mediante providers di accesso

esterni (Facebook, Google, Account Microsoft, Twitter etc.). A seconda dei meccanismi utilizzati, Identity storicizza più o meno informazioni utili per il loro corretto funzionamento.

Figura 3.9. Focus sul sistema di user accounting

Le informazioni utente, sono salvate in un database relazionale, in particolare nella tabella denominata *AspNetUsers* generata automaticamente da Identity. È possibile estenderla alla bisogna in base al modello dati, definito in fase di design, al fine di profilare in modo più dettagliato l'utente applicativo. La modellazione dei dati utente, in questo caso, è stata posticipata a fine tesi, per cui al momento non sono stati definiti campi addizionali a quelli già presenti.

La registrazione utente avviene tramite inserimento di username (=email) e password, oppure mediante provider di accesso esterno (Google). In caso di registrazione in loco, il meccanismo di conferma registrazione invia all'utente un messaggio di posta elettronica all'indirizzo specificato, utilizzando il servizio Azure SendGrid. Questo messaggio conterrà un link ad una call-back esposta dall'applicazione ad un particolare URL pubblico. La call-back, invocata cliccando il link contenuto nel messaggio, finalizzerà il processo di registrazione.

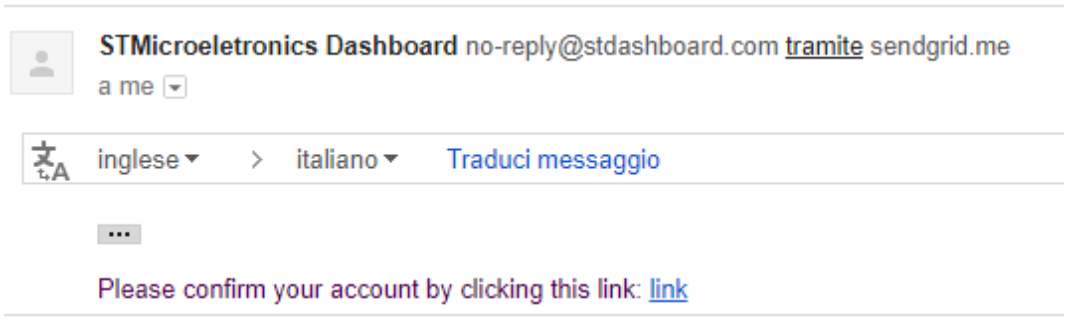

Figura 3.10. Email di conferma registrazione

Front-end

L'interfaccia, tramite la quale l'utente interagisce con il sistema di accounting, si compone principalmente di cinque pagine web: log-in utente, registrazione utente, conferma registrazione esterna, invio mail di conferma registrazione e conferma registrazione avvenuta.

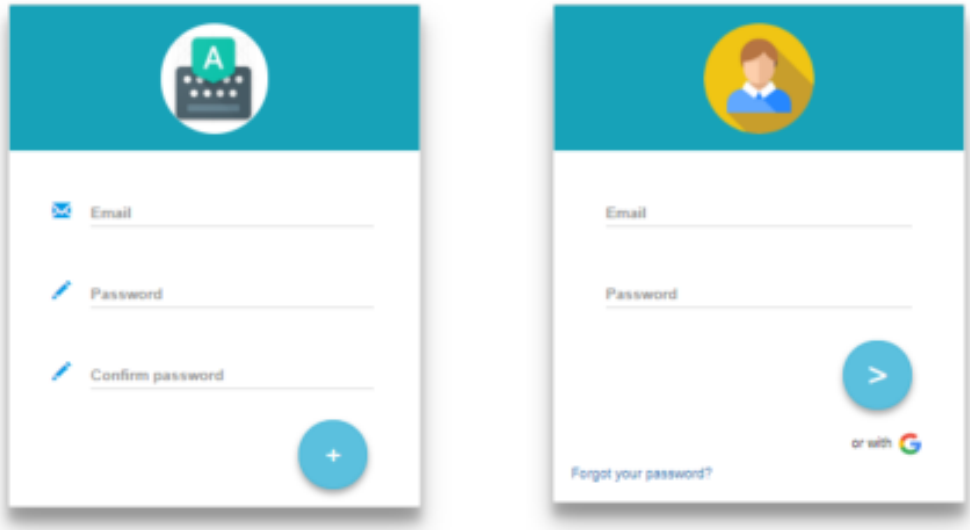

Figura 3.11. Pagine di log-in e registrazione

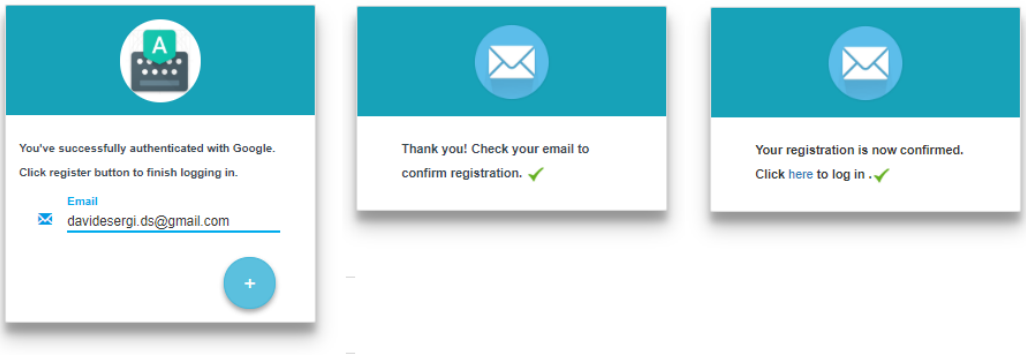

Figura 3.12. Pagine di conferma registrazione esterna, invio email di conferma e registrazione con successo.

3.2.2 Registrazione dispositivi

Di seguito sono riportate le funzionalità offerte dal sistema per la registrazione dei dispositivi in termini di user story:

As a customer I want an easy way to register and connect my device, so that I can start quickly to visualize, store and analyze data in the web dashboard. As a customer I want to be able to register and manage more than one device so that I can prototype more complex use cases. As a solution provider I want that limit the usage of the service to ST boards, so that technical and development resources are not leaked. As a solution provider I want to better track the usage of ST boards, FW and web-dashboard to understand the reception and acceptance of the solution.

Il processo di registrazione dei dispositivi si svolge in due fasi sequenziali tra loro: registrazione lato utente e lato dispositivo.

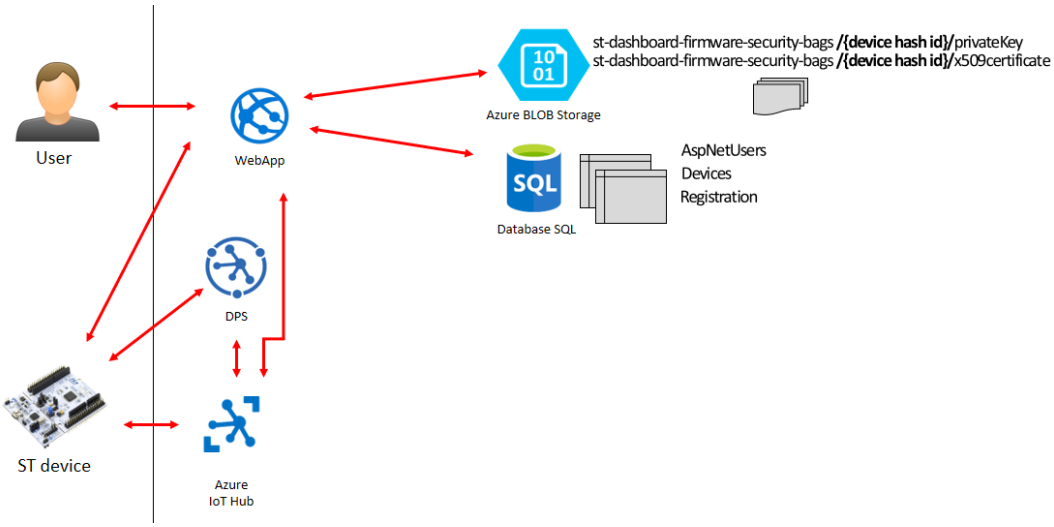

Figura 3.13. Focus sul sistema per la registrazione dei dispositivi

La prima, ha il compito di associare il dispositivo all'account utente che ne fa richiesta, garantendo POP (Proof of Possession) del dispositivo stesso. Il secondo, vede protagonista il dispositivo fisico ed ha il compito di farne il provisioning a livello applicativo e successivamente a livello cloud abilitandolo a comunicare con l'IoT Hub. Il meccanismo, in questo modo, tutela il sistema sia da eventuali dispositivi cloni e sia da utenti malevoli che riescono ad entrare in possesso di QR codes validi.

Back-end

Tutti i dispositivi in gioco sono noti a priori lato back-end, e sono storicizzati nella tabella *Devices* del database relazionale. L'informazione circa l'associazione tra dispositivo e account utente è storicizzata nella tabella *Registration*.

Lato utente

L'utente, per procedere con la registrazione, deve necessariamente possedere il QR code riportato sul blister contenente il dispositivo da registrare. Il QR code corrisponde all'hash dell'identificativo segreto del dispositivo; entrambe le informazioni sono note lato back-end.

Figura 3.14. ST board blister con QR code

Il processo di registrazione si svolge in tre fasi: (Figure 3.15)

1. L'utente scansiona il QR code da dashboard, scatenando richiesta di registrazione per il device identificato dall'id contenuto nel QR code.
2. L'applicazione web controlla se l'id ricevuto è tra quelli noti nel database.
3. Se l'id non esiste, la richiesta viene rifiutata; in caso contrario:
   (a) Si genera una coppia chiave privata – chiave pubblica RSA e relativo certificato X.509 a partire da un root certificate storicizzato nel blob storage.
   (b) Si rendono permanenti chiave privata e certificato nel blob storage ad un percorso dedicato per il dispositivo in oggetto (vedi figura).

A questo punto il dispositivo è stato associato all'utente a livello applicativo, ma non è ancora utilizzabile. Questo perchè non esiste ancora una entry, ad esso relativa, nell'identity register dell'IoT Hub. Il processo di registrazione utente deve precedere necessariamente quello di dispositivo, in quanto, per potersi registrare presso l'hub, ha bisogno di conoscere la propria chiave privata e certificato.

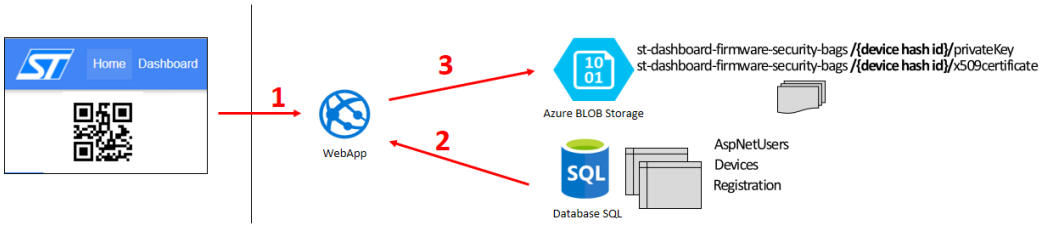

Figura 3.15. Schema registrazione dispositivo lato utente

Lato dispositivo

Questo processo prende in considerazione il dispositivo fisico allo scopo di abilitarlo a comunicare con l'IoT Hub.

In particolare, realizza il provisioning del dispositivo in oggetto a due differenti livelli: applicativo e cloud Azure. Le due operazioni devono essere svolte in sequenza, in quanto il provisioning applicativo è propedeutico a quello cloud.

Provisioning applicativo

Lo scopo di questa operazione consiste nel consegnare in modo sicuro, chiave privata e certificato a chiave pubblica appartenenti al dispositivo, generati al passo precedente. Per ottenere tali informazioni, il dispositivo dovrà autenticarsi presso l'applicazione dimostrando la propria identità.

Di seguito è riportato il processo nel dettaglio delle sue operazioni: (Figure 3.16)

1. Il dispositivo richiede all'applicazione chiave privata e certificato presentando l'hash del suo identificativo.
2. L'applicazione controlla l'effettiva esistenza dell'identificativo tra quelli noti, e:
   (a) Rifiuta la richiesta se l'id non esiste.
   (b) In caso contrario, genera una sfida per il dispositivo richiedente e gliela ritorna.
3. Il dispositivo ricevuta la sfida, la risolve utilizzando il proprio id segreto (condiviso tra le due parti) e ritorna il risultato all'applicazione.
4. L'applicazione, ricevuto il risultato della sfida, calcola nuovamente il risultato della sfida con l'id segreto da essa conosciuto (storicizzato nel database), e confronta i due risultati ottenuti:

(a) Se non coincidono, il dispositivo ha fallito l'autenticazione,

(b) Se coincidono, il dispositivo è autenticato ed è autorizzato a ricevere la propria chiave privata e certificato. A questo punto il dispositivo è considerato provisioned a livello applicativo.

Il dispositivo utilizza TLS con cifratura abilitata, per questo motivo è stato deciso di non cifrare ulteriormente chiave privata e certificato a chiave pubblica.

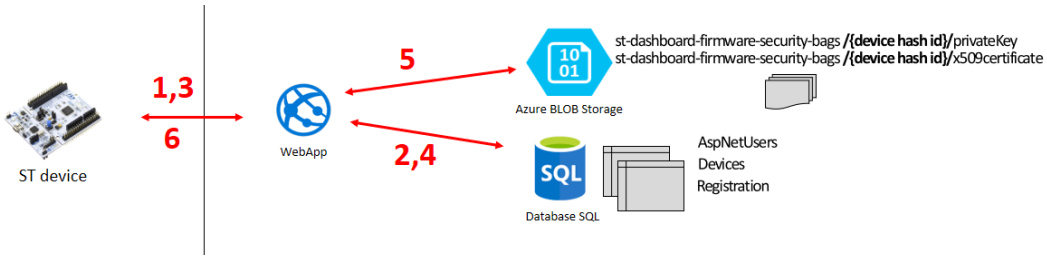

Figura 3.16. Device provisioning applicativo- lato dispositivo

Provisioning cloud

Lo scopo di questa operazione è abilitare il dispositivo a comunicare con l'IoT Hub. A tal proposito, il DPS è stato configurato per accettare richieste di provisiong da dispotivi in possesso di certificato X.509, generato a partire da un dato root certificate. Il root certificate impostato sul DPS coincide con quello utilizzato dall'applicazione, in cloud, per certificare la chiave pubblica generata durante il provisioning del dispositivo a livello applicativo (vedi passo precedente).

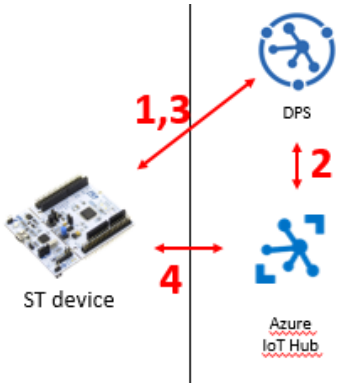

Figura 3.17. Device provisioning cloud- lato dispositivo

Di seguito è riportato il processo nel dettaglio delle sue operazioni: (Figure 3.17)

1. Il dispositivo fa richiesta di registrazione presso l'IoT Hub al DPS, previa autenticazione standard X.509.
2. In caso di autenticazione con successo, il DPS aggiunge, se non esiste, una entry relativa al dispositivo richiedente nell'identity register dell'IoT hub.
3. Il DPS torna al dispositivo l'id con la quale deve presentarsi presso l'IoT Hub, e il riferimento all'hub stesso.
4. Il dispositivo si autentica presso l'IoT Hub indicato dal DPS, mediante autenticazione standard X.509, e, se autorizzato, potrà iniziare a comunicare con esso.

Front-end

L'interfaccia utente permette di scansionare il QR code associato al dispositivo mediante fotocamera. Al termine della scansione viene generata una richiesta di registrazione device per l'utente loggato.

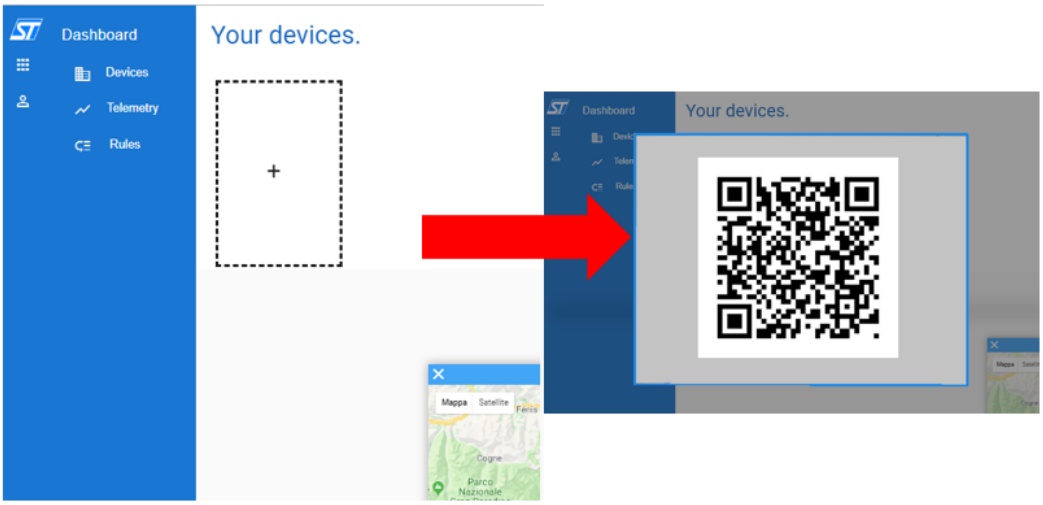

Figura 3.18. Registrazione dispositivo - interfaccia utente

I dispositivi una volta registrati a livello utente si possono presentare in tre differenti stati opportunamente rappresentati a livello grafico: (Figure 3.19).

- Associati all'account utente.
- Provisioned a livello applicativo.

- Provisioned a livello cloud.

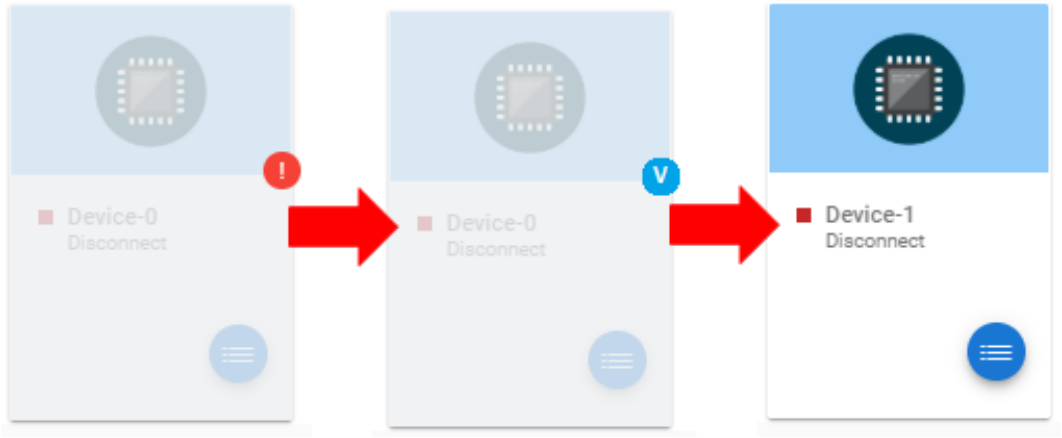

Figura 3.19. Steps registrazione dispositivo: associato allo user account, prvisioned a livello applicativo, provisioned a livello cloud

3.2.3 Dettaglio dispositivo

Di seguito sono riportate le funzionalità offerte dal sistema che realizza il dettaglio dei dispositivi, in termini di user story:

As a customer I want a visualize device status and sensors data in professional way, so that I can understand how the final control panel of a custom design will look like. As a customer I want to have an overview of the registered and connected devices, showing the status of the HW, FW, connectivity and location. As a customer, I want to manage the device according to the exact combination of sensors/actuators I'm using

Il sistema *dettaglio dispositivo* ha lo scopo di abilitare l'utente a monitorare lo stato, controllare e configurare il dispositivo tramite web dashboard. Prima che questo avvenga, il dispositivo deve essere stato opportunamente registrato e deve aver svolto le operazioni di provisioning, di livello applicativo e cloud, con successo.

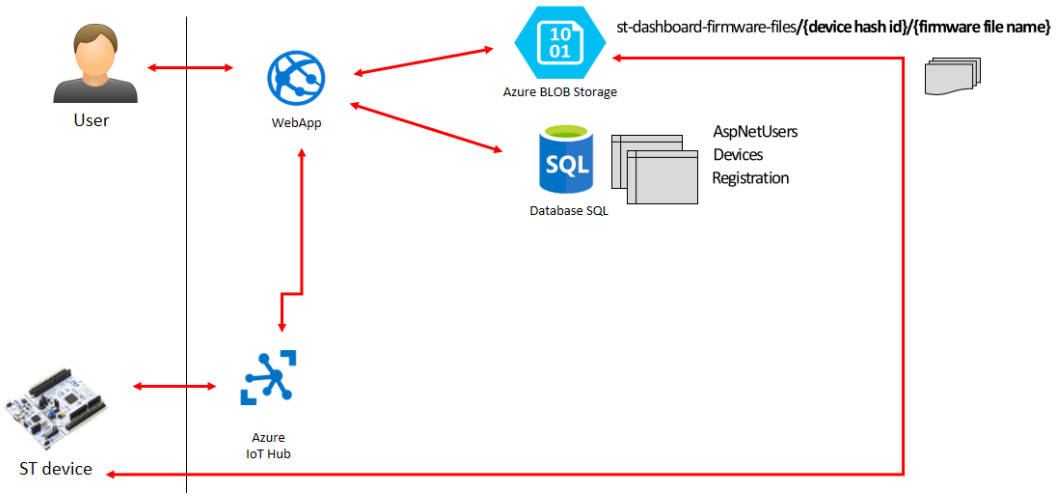

Figura 3.20. Focus su sistema del dettaglio dispositivo

Stato del dispositivo

Questa sezione ha lo scopo di riportare una breve descrizione del dispositivo, in termini di: composizione, stato connettività, memoria e posizione.

A tal scopo, tali informazioni sono ottenute aggregando dati recuperati dal twin del dispositivo e dalla tabella *Devices* contenuta nel database relazionale.

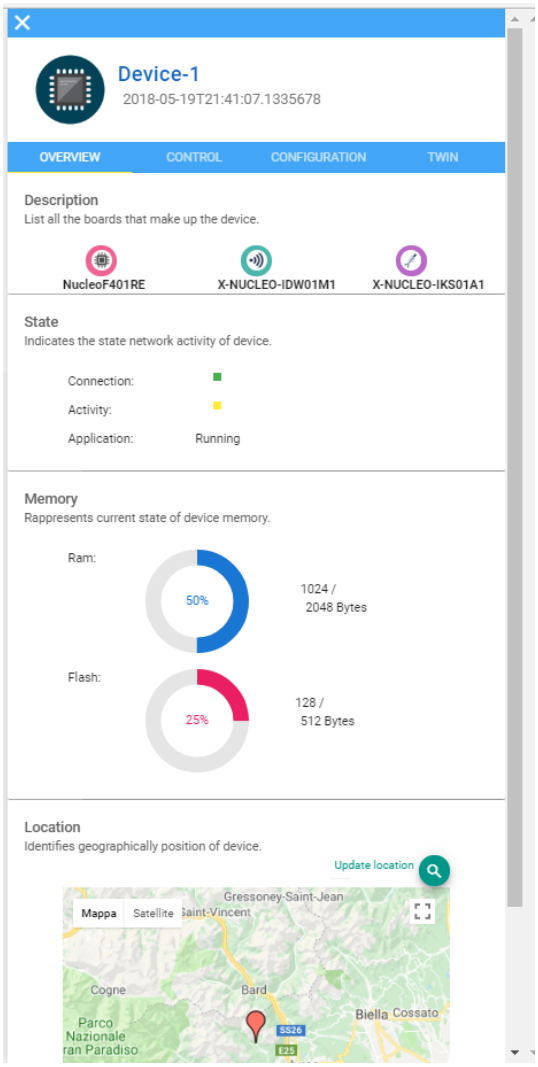

Figura 3.21. Stato dispositivo- interfaccia utente

Descrizione

È riportata la descrizione del dispositivo in termini di boards che lo compongono (scheda di sviluppo ed espansioni), e per ognuna delle quali sono riportati nome e icona tipo (mcu,connect,sense,move-actuate). Queste informazioni sono contenute esclusivamente nelle reported properties del twin associato al dispositivo.

Stato

Sono riportate informazioni circa l'attività in rete del dispositivo, ovvero:

- *Connettività*: indica se il dispositivo è attualmente connesso o meno all'IoT Hub, e coincide con il campo *connectionState* del twin.
- *Attività*: indica se il dispositivo è in stato di idle o active. Tale valore viene tarato sulla frequenza di invio dei dati di telemetria.
- *Stato applicazione*: coincide con il campo *status* riportato nelle reported properties del twin, ed indica lo stato attualmente in essere dell'applicazione del dispositvo (running, stopped, updating Firmware etc.).

Memoria

Indica l'attuale configurazione di ROM e RAM del dispositivo, in termini di dimensione totale e spazio libero espressi in bytes. Anche in questo caso, tutte le informazioni necessarie a questo scopo, sono contenute nelle reported properties del twin.

Posizione

Indica l'attuale posizione geografica del dispositivo, ed è ottenuta direttamente dal twin nel caso in cui il dispositivo equipaggi una scheda di espansione con funzionalità GPS, oppure dal campo *Location* della tabella *Devices* del database (aggiornabile dall'utente).

Controllo dispositivo

Per controllare il dispositivo lato cloud si utilizzano i direct methods, strumenti offerti a tal scopo dall'IoT Hub.

Il firmware FP1-CLD-AZURE1, implementa i seguenti metodi: led-on, led-off, led-blink, quit, reboot, firmware-update, etc. In base alla composizione del dispositivo, un sottoinsieme di questi saranno supportati, e di conseguenza riportati nella sezione *SupportedMethods* delle reported properties del twin.

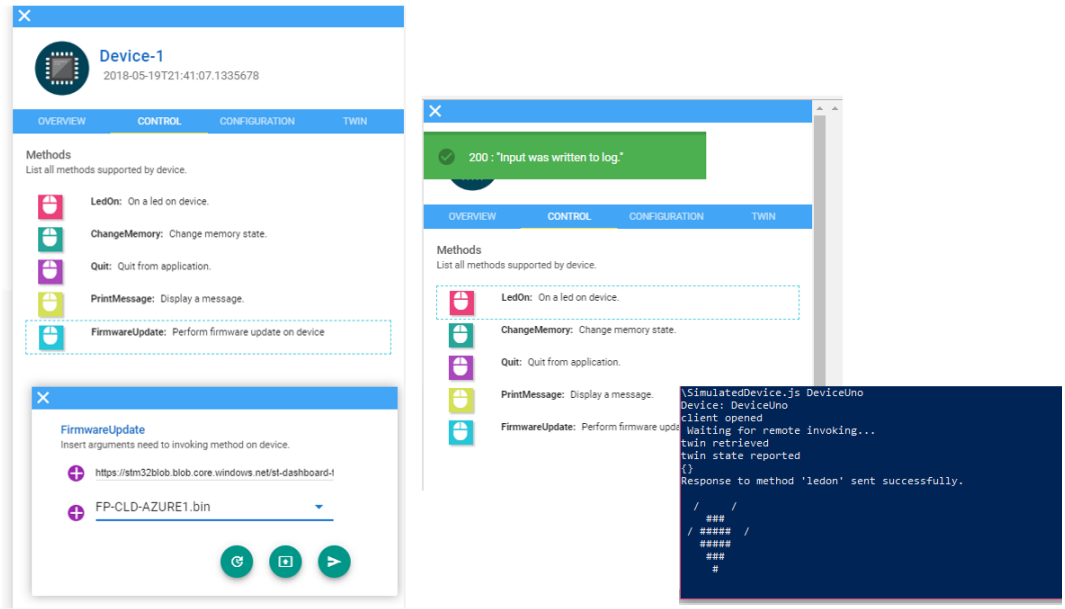

Figura 3.22. Controllo dispositivo - interfaccia utente

Esecuzione di un generico metodo diretto

Un direct method è in pratica una routine applicativa, implementata nel firmware del dispositivo, la quale viene eseguita in modo sincrono al momento dell'invocazione da remoto.

All'invocazione di un direct method vengono eseguite le seguenti operazioni: (Figure 3.23)

1. L'utente richiede l'esecuzione del metodo, specificandone nome e eventualmente parametri di input.

2. L'applicazione web genera una richiesta di esecuzione all'IoT Hub, indicando nel payload nome del metodo ed eventuali parametri di input.

3. L'IoT Hub, a questo punto

   (a) se il dispositivo è connesso, invia la richiesta di esecuzione al topic MQTT dedicato su cui il dispositivo è in ascolto (continua a punto 4.

   (b) se il dispositivo non è connesso, allo scadere del timeout, notifica l'applicazione circa il tentativo fallito di esecuzione, tornando un apposito messaggio di risposta. L'applicazione a questo punto notificherà opportunamente l'utente, e il processo può considerarsi terminato.

4. Il dispositivo riceve la richiesta ed esegue il metodo.
5. Al termine dell'esecuzione, il dispositivo torna all'IoT Hub la risposta contenente il risultato dell'esecuzione.
6. L'IoT Hub, a questo punto, torna all'applicazione il risultato ricevuto dal dispositivo.
7. L'applicazione web notifica l'utente in base al risultato.

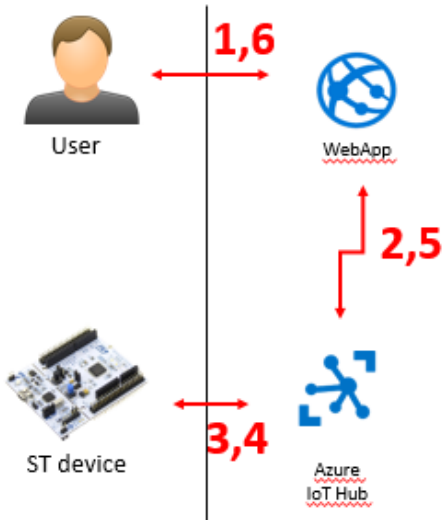

Figura 3.23. Esecuzione metodo - best case

Firmware update

L'operazione di firmware update richiede un'attenzione particolare, in quanto necessita di ulteriori attività di contorno, atte allo svolgimento dell'operazione stessa.
Di seguito è riportata la procedura dettagliata: (Figure 3.24)

1. L'utente carica, tramite web dashboard, il binario firmware per un dato dispositivo.
2. L'applicazione web, storicizza il file passatogli, in uno spazio dedicato al dispositivo sul blob storage ospitato in cloud.
3. A questo punto, l'utente, seleziona il file appena caricato (o un altro file precedentemente caricato), e richiede l'invocazione del metodo di firmware update, passando come parametro di input l'URL pubblico relativo al file.

4. L'applicazione prende in carico la richiesta rigirandola all'IoT Hub.
5. L'IoT Hub a questo punto invoca il metodo indicato sul dispositivo.
6. Il dispositivo esegue il metodo di firmware update, il quale prevede due fasi:
   (a) Scarico del binario, dall'URL passatogli come parametro del metodo.
   (b) Installazione del firmware.
7. Terminata l'esecuzione, il dispositivo torna il risultato all'IoT Hub.
8. L'IoT Hub notifica l'applicazione con il risultato ricevuto dal dispositivo.
9. L'applicazione notifica opportunamente l'utente.

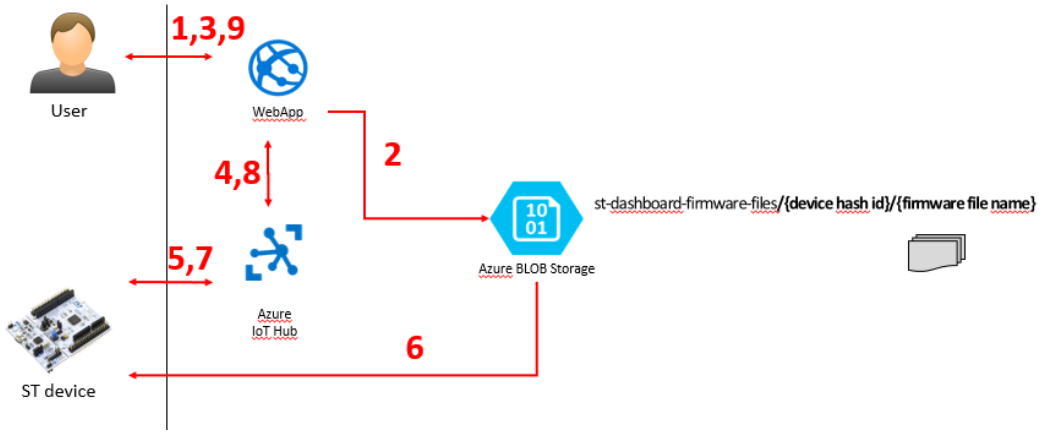

Figura 3.24. Firmware update nel dettaglio delle operazioni - best case

Configurazione dispositivo

La configurazione da remoto del dispositivo avviene modificando opportunamente il contenuto delle desired properties del twin associato, tramite il tab *Configuration* riportato nell'interfaccia utente. In pratica, in base alle modifiche richieste dall'utente, l'applicazione web va a contattare l'IoT Hub cosicché tali modifiche vengano storicizzate permanentemente nel twin del dispositivo. Quest'ultimo è possibile visualizzarlo in sola lettura mediante il tab *Twin*, anch'esso riportato nell'interfaccia utente del dettaglio dispositivo.

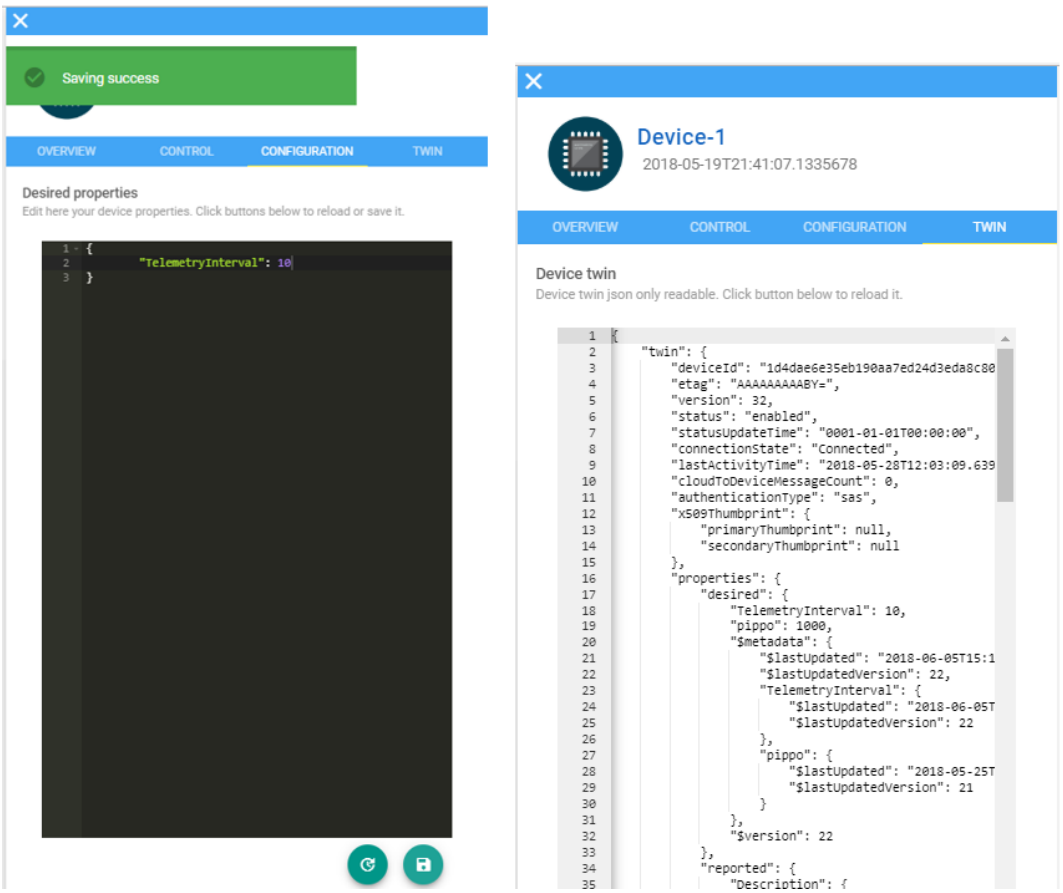

Figura 3.25. Configurazione dispositivo - interfaccia utente

3.2.4 Telemetria

Di seguito sono riportate le funzionalità offerte dal sistema che gestisce la telemetria dei dispositivi in termini di user story:

As a customer I want an easy way to register and connect my device, so that I can start quickly to visualize, store and analyze data in the web dashboard. As a customer I want a visualize device status and sensors data in professional way, so that I can understand how the final control panel of a custom design will look like.

L'obiettivo è storicizzare (per un massimo di 48 ore) e rendere disponibili, agli utenti che ne fanno richiesta, i dati di telemetria inviati dai propri dispositivi registrati e connessi al cloud.

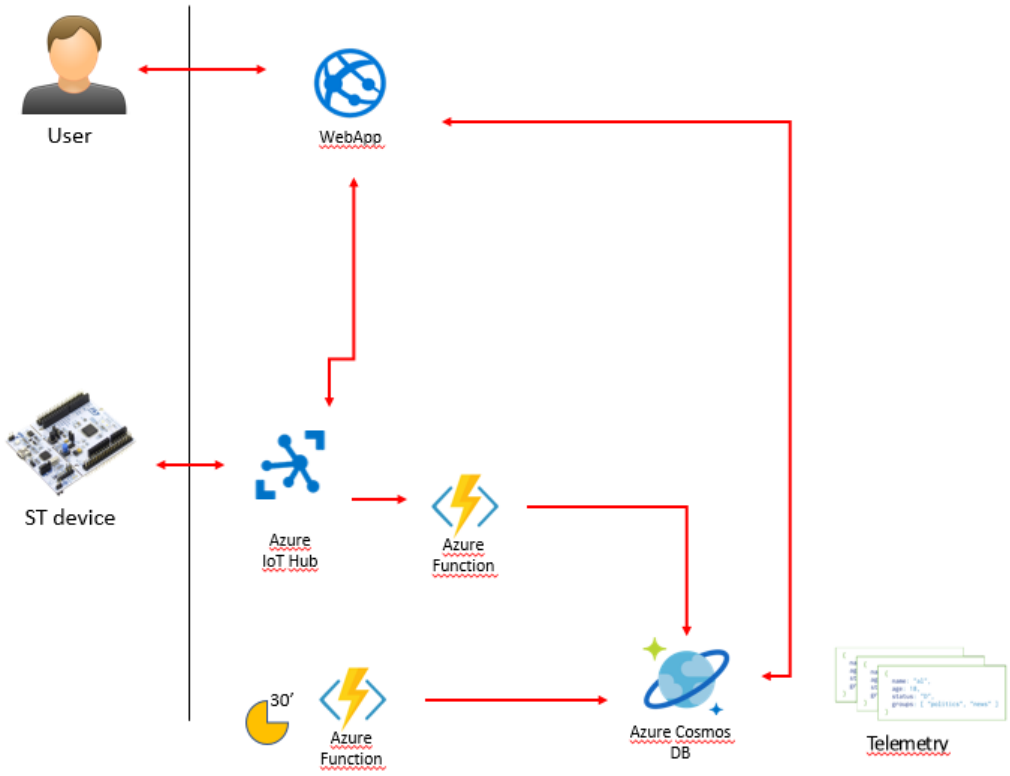

Figura 3.26. Schema completo telemetria

Sistema di visualizzazione

Il *sistema di visualizzazione* (Figure 3.27) ha l'obiettivo di mostrare all'utente i dispositivi che supportano telemetria e i relativi dati da loro inviati al cloud.

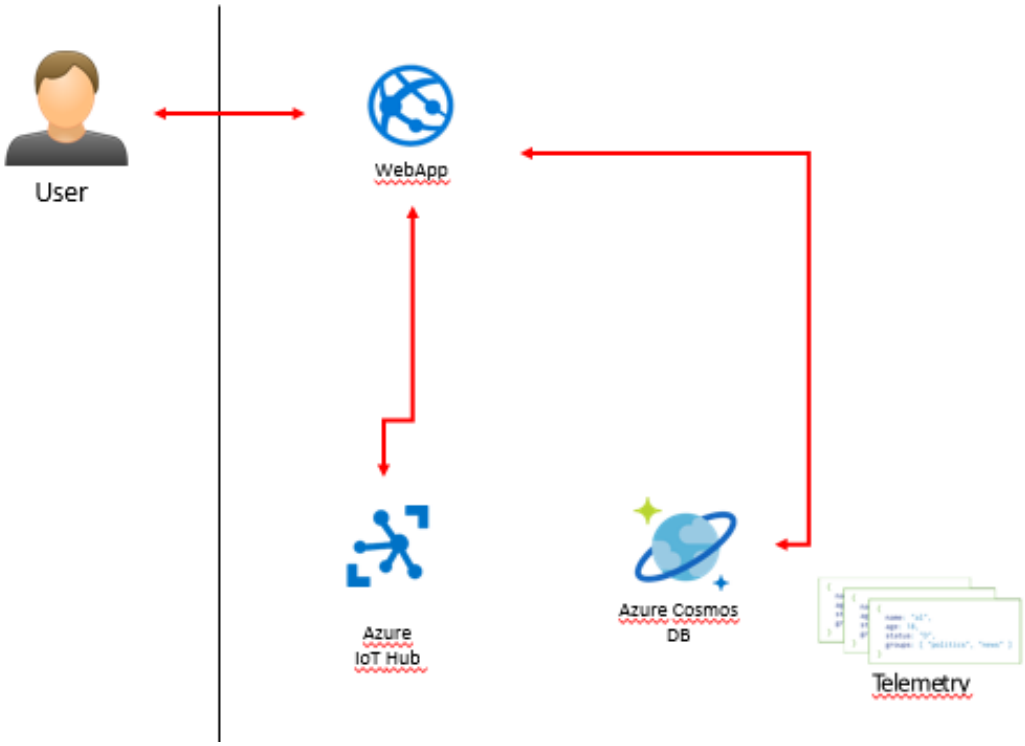

Figura 3.27. Sistema visualizzazione telemetria

Un dispositivo supporta la telemetria se utilizza sensori (inerziali, porssimità, ambientali, etc.) integrati nella scheda di sviluppo (IoT Discovery Board) oppure disponibili attraverso schede di espansione STM32ODE Sense HW. Assieme al dispositivo sono riportate tutte le misure che è in grado di comunicare (umidità, temperatura etc.). L'applicazione ottiene queste informazioni andando a consultare la composizione del dispositivo, precedentemente esaminata.

L'utente può richiedere di osservare i dati di telemetria storicizzati nel database, nel tempo e organizzati per tipo di misura. Può selezionare uno o più dispositivi da monitorare, e una o più misure, osservando sullo stesso grafico dati provenienti da differenti dispositivi.

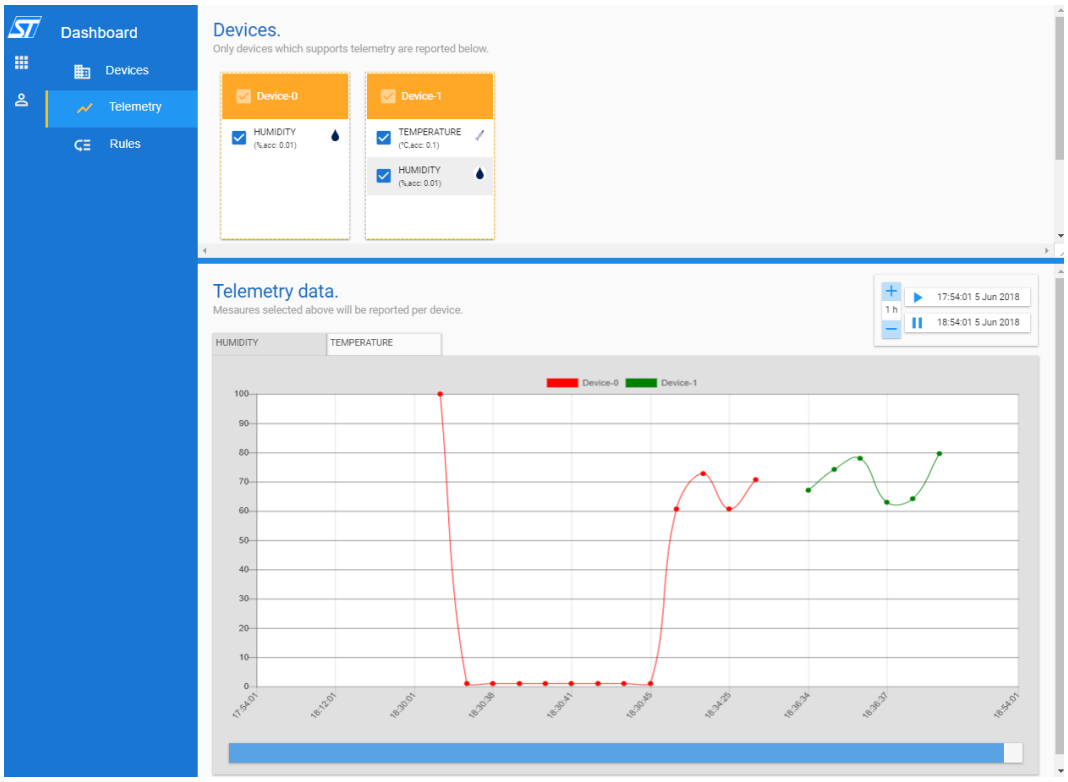

Figura 3.28. Telemetria - interfaccia utente

Sistema di storicizzazione

Il *sistema di storicizzazione* (Figure 3.29) ha l'obiettivo di ricevere i dati provenienti dai dispositivi e storicizzarli nel database no-sql.

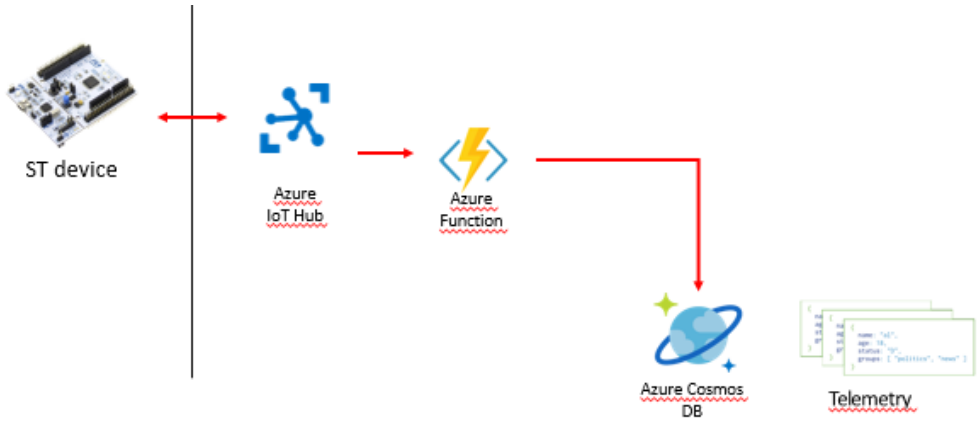

Figura 3.29. Sistema storicizzazione telemetria

La logica necessaria a storicizzare i messaggi nel database è implementata da una Azure Function dedicata. Questa riceve i messaggi che l'IoT Hub ridirige sull'end point di eventi che espone in cloud, e li rende permanenti nel database, senza manipolarli ulteriormente.

3.2.5 Regole sull'attività del dispositivo

Di seguito sono riportate le funzionalità offerte dal sistema che gestisce le regole dei dispositivi in termini di user story:

As a customer, I want to edit some simple rules to automate basic actions and notifications when received sensors data satisfy a given condition

Lo scopo che si vuole raggiungere è di abilitare l'utente a definire, per dispositivo, una o più regole su un insieme di dati telemetrici, al fine di essere notificato via email nel momento in cui le condizioni da esse indicate si verificano.

Per regola si intende una funzione che prende in ingresso n valori reali e ritorna un valore booleano.

$$f(x_0, x_1, \ldots, x_n) : \mathrm{R}^n \to B$$

Figura 3.30. Definizione di regola

Il valore di ritorno è true se si verifica la condizione modellata dalla funzione, false altrimenti.

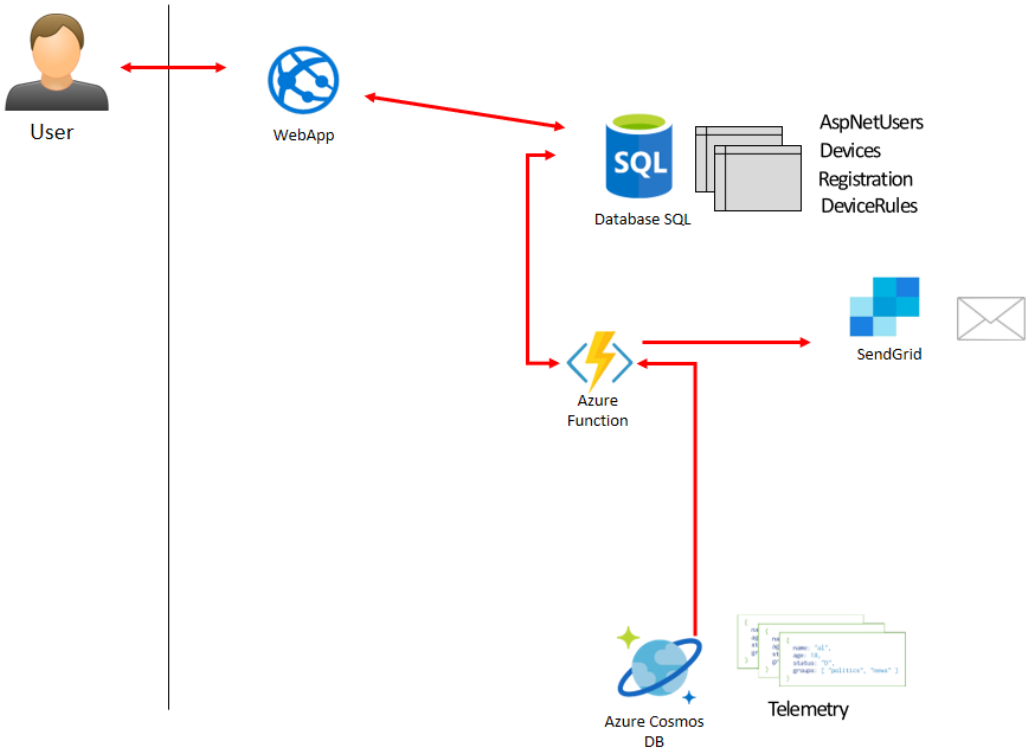

Figura 3.31. Schema completo per la definizione regole

Anche in questo caso si possono individuare due sotto-sistemi: uno dedicato al salvataggio e visualizzazione delle regole che l'utente definisce tramite dashboard, e uno dedicato alla valutazione delle regole stesse. Entrambi i sistemi lavorano sulla tabella *DeviceRules* del database relazionale, nella quale sono storicizzate le regole.

Sistema di storicizzazione regole

Il *sistema di storicizzazione regole* ha l'obiettivo di abilitare l'utente a scrivere concretamente le regole per i dispositivi e storicizzarle nel database relazionale.

Si è deciso di modellare una regola come una funzione Javascript, la quale torna un booleano e prende in input due parametri: lastMessages e $return. Il parametro lastMessages è una funzione che ha il compito di recuperare, da database, gli ultimi n valori relativi a una misura di tipo opportunamente specificato. Il secondo parametro $return, ha il compito di terminare la funzione e tornare l'esito della valutazione della condizione (true o false).

L'utente può inoltre cancellare, aggiornare, mettere in pausa e ri-avviare le regole tramite web dashboard. Mettere in pausa e avviare una regola significa indicare al sistema prendere o meno in considerazione la regola durante la valutazione.

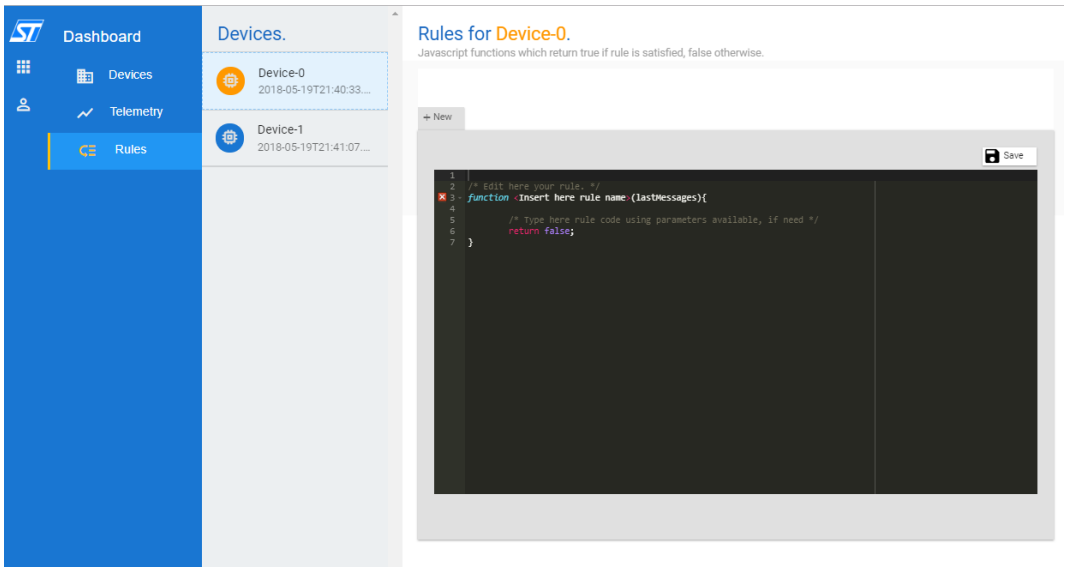

Figura 3.32. Definizione regole - interfaccia utente

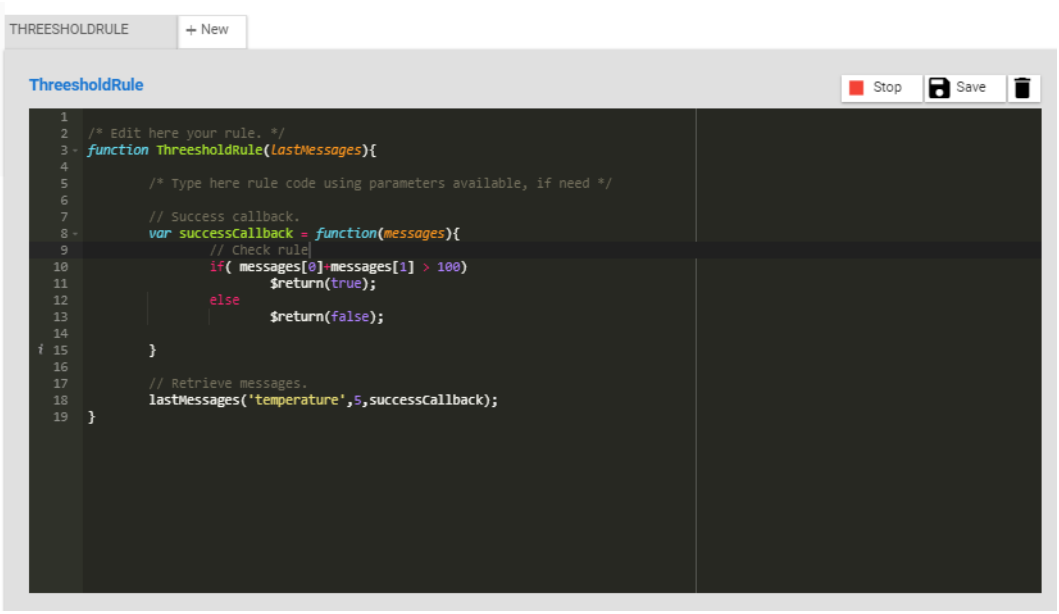

Figura 3.33. Esempio regola - interfaccia utente

Sistema di valutazione

Il *sistema di valutazione* ha l'obiettivo di valutare le regole definite dagli utenti per i dispositivi. A tal proposito è stata istanziata una apposita Azure Function che funge da engine Javascript.

In particolare, la funzione, ad ogni inserimento nel database no-sql, viene notificata dal sistema, ricevendo il messaggio oggetto dell'operazione. A questo punto, la funzione legge l'id del dispositivo mittente, recupera tutte le regole attive definite per quel dispositivo dalla tabella *DeviceRules*, e le esegue una alla volta

Ad ogni esecuzione, se è ritornato true, invia la mail di notifica all'utente possessore del dispositivo mediante Azure SendGrid. Nel corpo della mail sono indicati hash dell'identificativo del dispositivo e corpo della funzione che implementa la logica applicativa della regola. Infine, la tabella *DeviceRules* viene conseguentemente aggiornata segnando la regola come "da non eseguire" (loked=true), questo per limitare il numero di email inviate dal sistema.

Figura 3.34. Email di notifica

Capitolo 4

Implementazione

4.1 Http REST API

La dashboard web è stata realizzata come SPA (Single Page Application), per questo motivo è stato necessario modellare e implementare un'interfaccia REST Http allo scopo di far comunicare front-end e back-end.

Il provisioning applicativo del dispositivo prevede che esso interagisca con il cloud al fine di recuperare i dati necessari per richiedere al DPS si essere registrato presso l'IoT Hub. A tal scopo, anche in questo caso sono stati pubblicati end-points REST che il dispositivo andrà a contattare al suo avvio.

4.1.1 Registrazione dispositivo

Insieme di end-points necessari ad:

- Abilitare gli utenti a registrare i propri dispositivi.
- Abilitare i dispositivi a effettuare l'operazione provisioning applicativo.

URL	Request method	Request body	Request code	Request body	Meaning
user/devices/ {device hash id}	PUT		200		Ok: dispositivo associato con successo all'utente
			403		Unauthorized: utente non autorizzato
			404		Not found
/provisioning/ {device hash id}	GET		200		Ok: inizio autneticazione dispositivo
			404		Not found
	PUT	Challenge con valore e risultato	200	Chiave privata e certificato X.509 (security bag).	Ok: dispositivo provisioned con successo a livello applicativo
			403		Unauthorized: utente non autorizzato
			404		Not found

Strutture dati

Challenge : contiene i dati necessari a svolgere l'autenticazione del dispositivo basata sul meccanismo a sfida simmetrica.

- *ChallengeValue*: stringa contenente il valore nonce della sfida utilizzato durante l'operazione di autenticazione.
- *ChallengeResult*: stringa contenente il risultato della sfida.

Security bag : contiene i dati ottenuti come risultato dell'operazione di provisioning applicativo del dispositivo.

- *DeviceHashId*: stringa contenente il valore nonce della sfida utilizzato durante l'operazione di autenticazione.
- *PrivateKey (array di byte)*: stringa contenente il risultato della sfida.
- *Certificate X.509 (array di byte)*: certificato X.509.

4.1.2 Dettaglio dispositivi

Insieme di end-points utilizzato dalla applicazione dashboard per:

- Ottenere e aggiornare lo stato del dispositivo.
- Richiedere l'esecuzione di metodi su dispositivo.
- Caricare firmware su blob storage.
- Recuperare e aggiornare il twin del dispositivo.

URL	Request method	Request body	Request code	Request body	Meaning
user/devices [?basicinformations=true/false]	GET		200	Lista dei dispositivi	Ok: elenco dispositivi utente, con descrizione minima o estesa
			403		Unauthorized: utente non autorizzato
user/devices/ {device hash id} [?basicinformations=true/false]	GET		200	Dispositivo	Ok: dispositivo utente, con descrizione minima o estesa
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
user/devices/ {device hash id}/method	POST	Modello richiesta esecuzione metodo	200	Modello risultato esecuzione metodo	Ok: esito esecuzione metodo su dispositivo
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato

URL	Request method	Request body	Request code	Request body	Meaning
user/devices/ {device hash id}/firmwares	POST	Modello firmware	200		Ok: nuovo file firmware salvato su blob storage
			403		Unauthorized: utente non autorizzato
			404		Not found
	GET		200	Lista firmwares	Ok: elenco firmwares caricati per il dispositvo
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
user/devices/ {device hash id}/location? latitude=value &longitude=value	PUT		200		Ok: posizione dispositivo utente aggiornata.
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato

URL	Request method	Request body	Request code	Request body	Meaning
user/devices/ {device hash id}/twin	GET		200	Twin dispositivo	Ok: twin aggiornato.
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
	PUT	Frammento twin	200		Ok: twin aggiornato
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato

Strutture dati

Dispositivo : descrive il dispositivo in oggetto, e si compone all'occorrenza dei seguenti campi.

- *Name (stringa)*: nome utente del dispositivo.
- *HashSerialId (stringa)*: hash dell'identificativo del dispositivo.
- *RegistrationTime (stringa)*: istante di registrazione del dispositivo.
- *State (oggetto)*: informazioni di stato del dispositivo.
  - *AppProvisioning (booleano)*: vero se il dispositivo è provisioned lato applicazione.
  - *Provisioning (booleano)*: vero se il dispositivo è provisioned lato IoT Hub.
  - *Connection (intero)*: indica se il dispositivo è connesso o meno al cloud (0=non definito,1=connesso,2=disconnesso).
  - *Comunication (intero)*: indica se il dispositivo è attivo o in stato di idle (0=non definito,1=attivo,2=idle).
  - *Application (stringa)*: indica lo stato dell'applicazione riportato dal dispositivo.

- *Package (stringa)*: indica il tipo di pacchetto con il quale è stato distribuito il dispositivo.
- *RegistrationType (stringa)*: indica il tipo di registrazione che associa dispositivo e utente (admin, manager, sharer). Di fatto questa informazione non è al momento utilizzata.
- *Description (oggetto)*: oggetto 'Description' estratto dalle reported properties del twin del dispositivo.
- *SupportedMethods (collezione di oggetti)*: metodi supportati dal dispositivo. Ogni oggetto metodo contiene le seguenti informazioni
  - *Method (stringa)*: descrizione metodo nel formato standard nome metodo[–nome parametro-tipo parametro]*.
  - *Description (stringa)*: breve descrizione del metodo.
- *Memory (oggetto)*: stato memorie del dispositivo.
  - *Ram (oggetto)*: stato RAM
  - *Available (intero)*: spazio attualmente libero in bytes.
  - *Size (intero)*: dimensione totale in bytes.
  - *Rom (oggetto)*: stato ROM
  - *Available (intero)*: spazio attualmente libero in bytes.
  - *Size (intero)*: dimensione totale in bytes.
- *Rules (collezione di oggetti)*: regole associate al dispositivo. Ogni oggetto regola contiene i seguenti campi
  - *Name (stringa)*: nome della regola.
  - *Body (stringa)*: corpo della regola.
  - *Stop (booleano)*: indica se la regola è attiva o meno.

Modello richiesta esecuzione metodo : descrive una richiesta di esecuzione metodo su dispositivo.

- *MethodName (stringa)*: nome del metodo.
- *Arguments (dizionario di oggetti)*: parametri di input per il metodo da eseguire (chiave=nome del parametro, valore=valore del parametro).

Modello risposta esecuzione metodo : descrive il risultato dell'esecuzione di un metodo.

- *Status (intero)*: codice risposta.
- *Result (stringa)*: risultato.

Modello firmware : descrive la richiesta di upload di un firmware.

- *Name (stringa)*: nome del firmware.
- *Uri (stringa)*: URI pubblico in cui è locato il file.
- *File (array di byte)*: contenuto del file.

Twin : twin del dispositivo.

Frammento twin (stringa) : modifica da apportare al twin del dispositivo.

4.1.3 Telemetria

Insieme di end-points utilizzato dalla dashboard per:

- Recuperare l'elenco dei dispositivi che supportano telemetria.
- Recuperare dati di telemetria per tipo di misura e finestra temporale.

URL	Request method	Request body	Request code	Request body	Meaning
user/devices/ telemetry	GET		200	Lista dei dispositivi	Ok: dispositivi utenti che supportano telemetria.
			403		Unauthorized: utente non autorizzato
user/devices/ telemetry/ {device hash id}/ {telemetry mesaure name}]	GET		200	Lista dati telemetria	Ok: lista dati telemetria per dispositivo, misura indicata e finestra temporale indicata.
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato

Strutture dati

Dispositivo : vedi sopra.

Dato telemetria : descrive un generico dato di telemetria.

- *DeviceId (stringa)*: hash dell'identificativo del dispositivo.
- *Timestamp (intero)*: timestamp messaggio di telemetria espresso in millisecondi trascorsi dalla UTC Unix Epoch.
- *Data (oggetto)*: messaggio (crudo) di telemetria.

4.1.4 Dettaglio dispositivi

Insieme di end-points, utilizzati dalla dashboard, volti ad aggiungere, aggiornare, cancellare una regola per un dato dispositivo.

URL	Request method	Request body	Request code	Request body	Meaning
user/devices/ {device hash id}/ rules	GET		200	Dispositivo	Ok: dispositivo con regole
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
	POST	Modello regola	200		Ok: regola aggiunta al dispositivo utente con successo
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
	PUT	Modello regola	200		Ok: regola aggiornata con successo
			403		Unauthorized: utente non autorizzato
			404		Not found: dispositivo utente non trovato
	DELE TE		200		Ok: regola rimossa con successo
			403		Unauthorized: utente non autorizzato
			404		Not Found: dispositivo o regola non trovati

Strutture dati

Dispositivo : vedi sopra.

Modello regola : descrive una regola per un dispositivo.

- *FunctionName (stringa)*: nome regola.
- *FunctionBody (stringa)*: corpo regola.
- *FunctionStop (booleano)*: vero se la regola deve essere attivata o meno.

4.2 Data storage

La soluzione sviluppata si compone principalmente di tre sistemi volti allo storaging dei dati, implementati dai servizi Azure SQL, Azure Cosmos DB e Azure BLOB Storage.

4.2.1 Azure SQL

Implementa il database relazionale sul quale si appoggia l'applicazione, e si compone delle seguenti tabelle:

- AspNetUsers
- Registration
- Devices
- DeviceRules

Di seguito è riportato lo schema relazionale del database nel dettaglio delle

- Entità individuate.
- Relazioni tra le entità e relative cardinalità.
- Attributi delle tabelle, con indicazione circa il tipo dato, nullabilità, e chiavi.

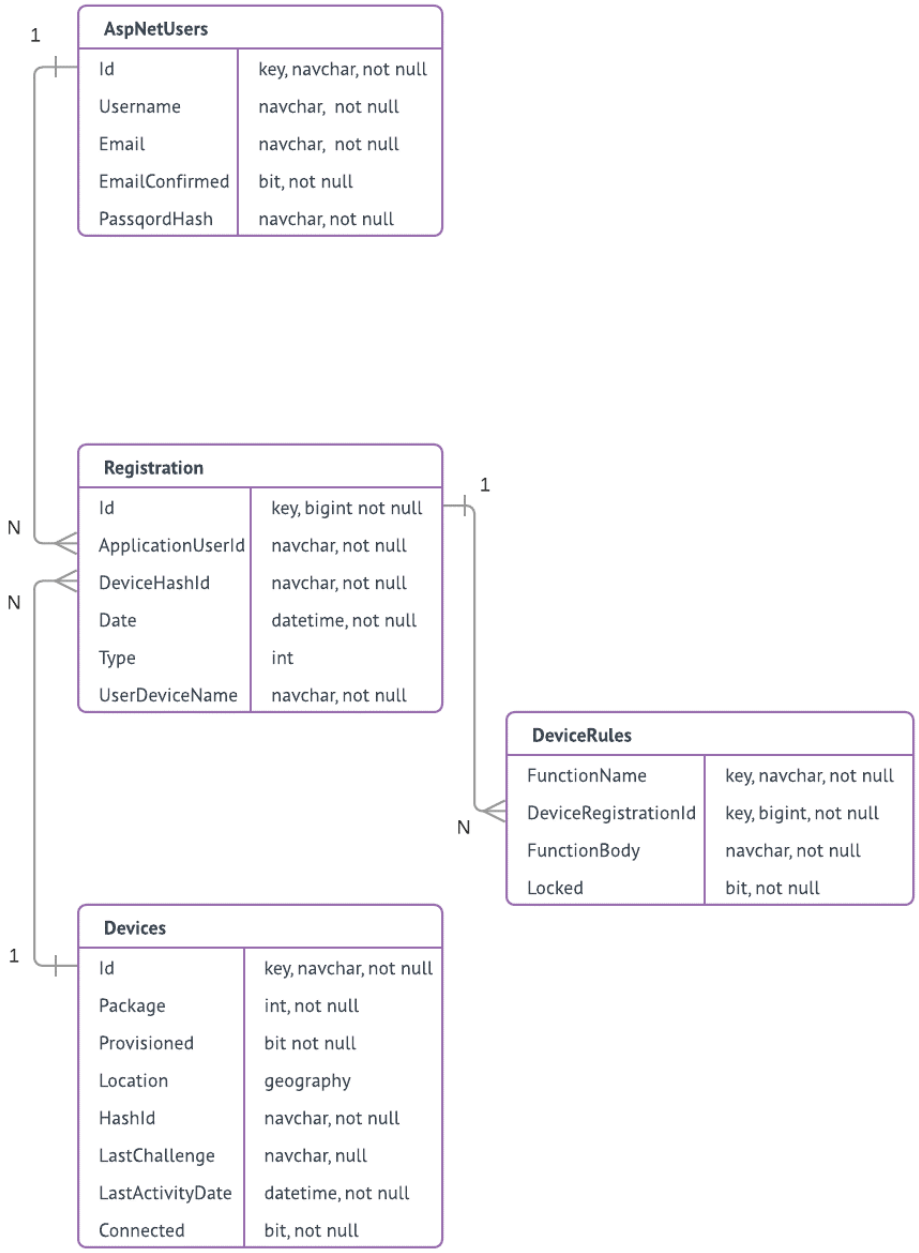

Figura 4.1. Modello ER

AspNetUsers

La tabella *AspNetUsers* contiene gli utenti noti al sistema. Lo schema della tabella si compone di:

- *ID*: id dell'utente.
- *Email*: email dell'utente.
- *Username*: coincide con il campo Email.
- *EmailConfirmed*: indica se il processo di registrazione utente è stato finalizzato.
- *PasswordHash*: password offuscata.

Devices

Tutti i dispositivi in gioco sono noti a priori lato back-end, e sono storicizzati nella tabella *Devices* del database relazionale. Lo schema della tabella si compone di:

- *ID*: rappresenta il segreto condiviso tra back-end e dispositivo, ed è alla base del meccanismo di registrazione sicura del dispositivo stesso.
- *Package*: modalità con la quale è stato distribuito il dispositivo.
- *Provisioned*: indica se il dispositivo è provisioned a livello applicativo o meno.
- *Location*: riporta la posizione geografica del dispositivo, nel caso in cui sia sprovvisto di funzionalità GPS.
- *HashId*: prodotto a partire da l'ID segreto, mediante funzione di hash non invertivile, identifica univocamente il device ed è possibile trasferirlo in rete senza che il segreto di partenza possa essere compromesso.
- *LastChallenge*: istante in cui è stata generata l'ultima sfida inviata al dispositivo.
- *LastActivityDate*: istante in cui è stato ricevuto l'ultimo dato di telemetria dal dispositivo.
- *Connected*: indica se il dispositivo è connesso o meno all'IoT Hub. Questa informazione è ottenuta in base allo stato di connessione riportato nel device twin e l'arrivo di messaggi da dispositivo.

Registration

L'associazione tra dispositivo e account utente è storicizzata nella tabella *Registration*. Lo schema della tabella si compone di:

- *ID*: identificativo della registrazione.
- *ApplicationUserId*: riferimento all'utente.
- *DeviceId*: riferimento al device.
- *Date*: istante in cui è avvenuta la registrazione.
- *Type*: indica se l'utente è registrato per il dispositivo come Admin, Manager o Sharer. In pratica, questa informazione non è utilizzata, ma può essere utile in sviluppi futuri nel caso in cui si voglia mettere in piedi un meccanismo di condivisione dei dispositivi tra utenti del sistema.
- *UserDeviceName*: nome del dispositivo assegnato dall'utente.

DeviceRules

Le regole utente da associare ai dispositivi sono storicizzate nella tabella *DeviceRules*. Lo schema della tabella si compone di:

- *DeviceRegistrationId*: riferimento alla registrazione utente-dispositivo.
- *FunctionName*: nome identificativo della funzione/regola.
- *FunctionBody*: corpo della funzione/regola.
- *Locked*: valore booleano che indica se la regola dev'essere valutata oppure no.

La foreign key *DeviceRegistrationId* è stata promossa come chiave, assieme al campo *FunctionName*, per poter definire una regola per più dispositivi in contemporanea. Questa potenzialità al momento non è sfruttata, in quanto all'utente è data la possibilità di definire una regola alla volta per ogni dispositivo registrato.

4.2.2 Azure Cosmos DB

Rappresenta il database No-Sql che il sistema utilizza allo scopo di storicizzare i messaggi di telemetria ricevuti da tutti i dispositivi connessi al cloud. Il modello dati scelto è MongoDB, per cui i dati storicizzati sono esposti come documenti BSON. Un messaggio di telemetria si presenta nel formato seguente:

- *"id"*: hash dell'identificativo segreto del dispositivo.

- *"time"*: timestamp del messaggio in millisecondi passati dalla UTC Unix Epoch.
- *"mesaure name"*: valore della misura riportata.
- *"mesaure name"*: valore della misura riportata.
- ...

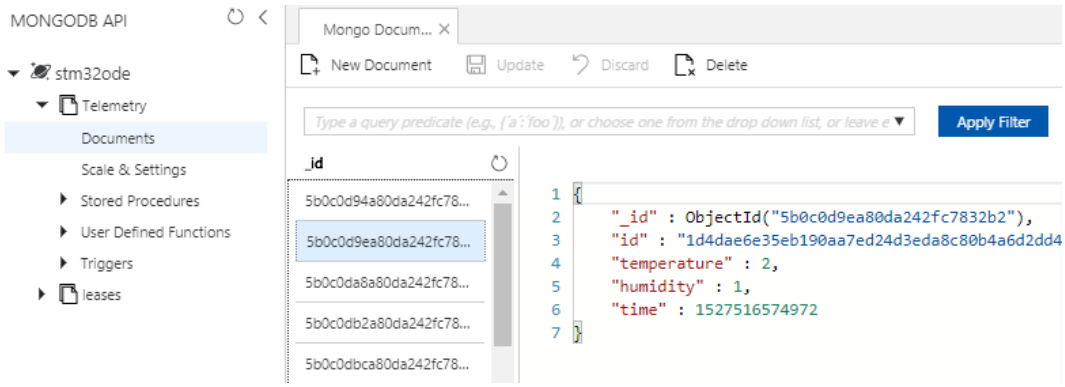

Figura 4.2. Azure Portal - Azure Cosmos DB data explorer

4.2.3 Azure BLOB Storage

È utilizzato come archivio file nel quale sono storicizzati:

- Files contenenti i firmware dei dispositivi caricati dall'utente.
- Files PEM contenenti chiave privata e certificato a chiave pubblica per i dispositivi.
- Files PEM contenenti chiave privata e certificato root da utilizzare per certificare le coppie chiavi privata e chiave pubbliche generata al volo dall'applicazione durante la registrazione dei dispositivi.

I percorsi nella quale sono locati i file sono:

- st-dashboard-firmware-files/{device hash id}/{firmware file name}
- st-dashboard-firmware-security-bags/{device hash id}/privateKey
- st-dashboard-firmware-security-bags /{device hash id}/x509certificate

I file sono categorizzati per dispositivo; i file firmware cono accessibili da dispositivo mediante URI pubblico; mentre chiave privata e certificato hanno accessibilità ristretta al solo back-end dell'applicazione web.

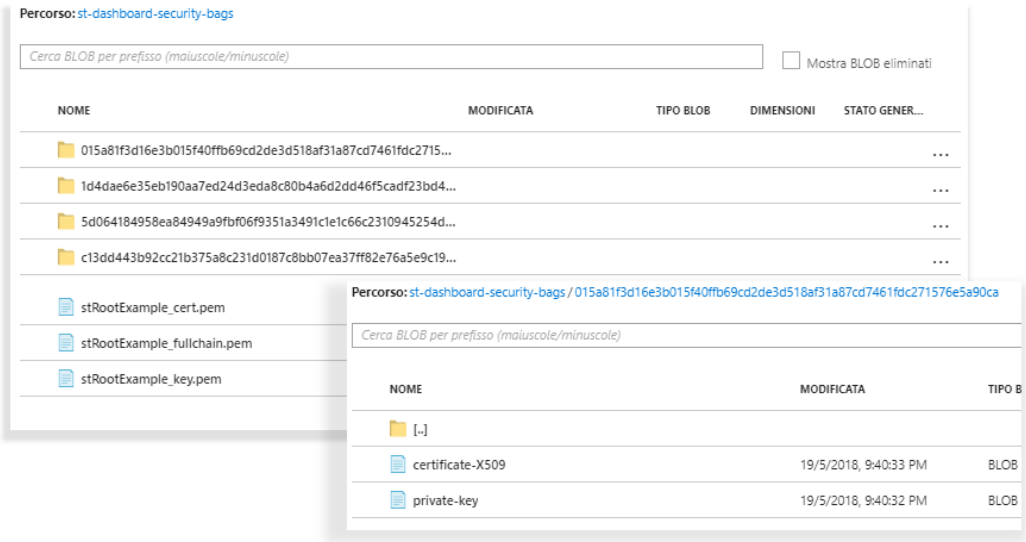

Figura 4.3. Azure Portal - Private keys e X.509 certificates file in BLOB storage

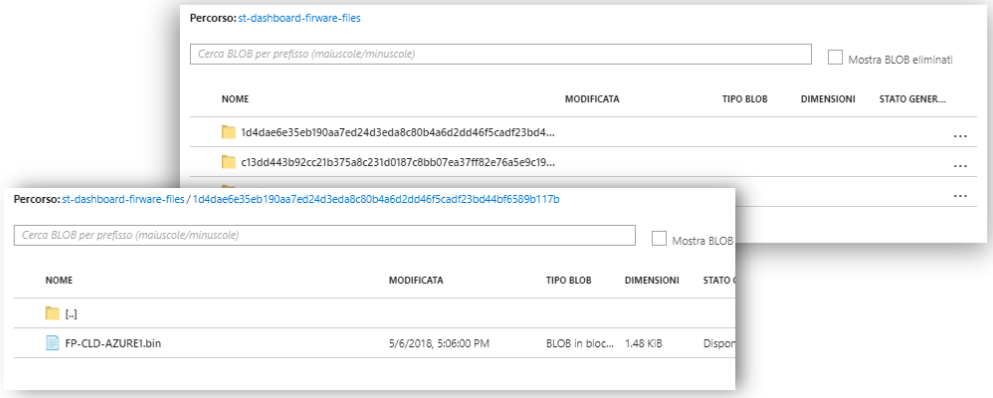

Figura 4.4. Azure Portal - Firmware files per devices

4.3 Modello standard Twin

Il twin è un oggetto dati utilizzato dall'applicazione per configurare e muovere di stato un dispositivo. A tal proposito, in fase di design è stato definito un modello dati standard per le sezioni desired e reported properties contenute nel twin.

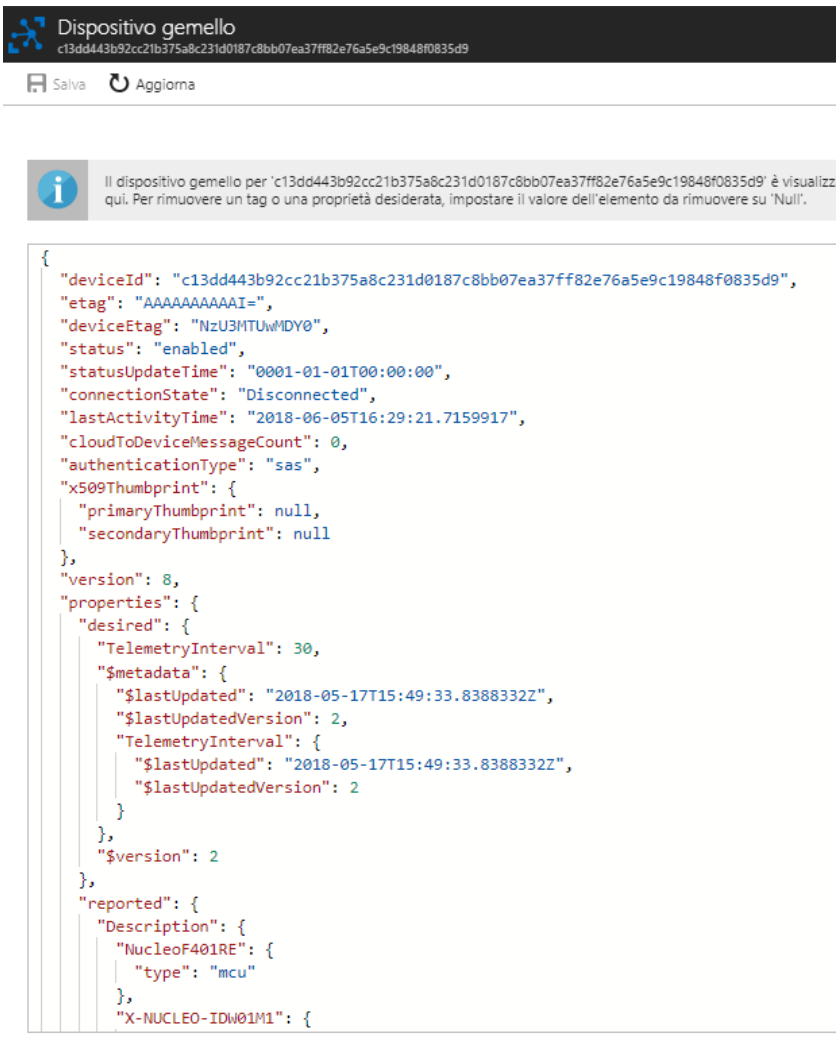

Figura 4.5. Azure Portal - Device twin

Di seguito è riportato in dettaglio delle informazioni che le due sezioni possono contenere.

Status (stringa) : indica lo stato in cui si trova l'applicazione del dispositivo.

TelemetryInterval (intero) : intervallo di tempo, espresso in secondi, che intercorre tra due messaggi di telemtria inviati dal dispositivo. Può essere inserito sia nelle reported che nelle desired properties.

SupportedMethods (oggetto) : esprime i metodi supportati dal dispositivo. Ogni supported method è espresso nel formato

*{ method name } [- - { input parameter name } - { input parameter type }]**

Questo approccio permette di definire nuovi metodi senza la necessità di dover la logica applicativa dell'applicazione web. Questo oggetto può essere contenuto esclusivamente nelle reported properties del twin.

Description (oggetto) : descrive la composizione del dispositivo in termini di boards, e le boards stesse. Il formato dell'oggetto Description è il seguente

"Description": { " boardName": { " type ": " mcu/connect/sense/move-actuate", ["subtype"]: " [wifi/radio/bluetooth/. . .]", [" mesaures"]:{ " mesaureName": { " unit " :value, " accurancy": value }, . . . } }

dove:

- *BoardName*: indica il nome della scheda equipaggiata dal dispositivo.
- *Type*: indica la famiglia di schede a cui appartiene la scheda. In particolare:
  - *Connect*: indica la famiglia STM32 ODE Connect HW, schede di espansione in grado di offrire connettività di vario genere.
  - *Move-acruate*: indica la famiglia STM32 ODE Move-Actuate HW, schede di espansione in grado di offrire controllo motore.
  - *Sense*: indica la famiglia STM32 ODE Sense HW, schede di espansioni in grado di offrire funzionalità di rilevamento.
  - *Mcu*: indica la famiglia di schede di sviluppo Nucleo e IoT Discovery.
- *SubType*: campo opzionale che ha il compito di portare informazioni aggiuntive circa la scheda.
- *Mesaures*: campo presente solo nel caso di schede di tipo sense; ha il compito di descrivere le misure riportate in termini di nome, unità di misura e accuratezza.

L'oggetto Description è esclusivamente riportato dal dispositivo nella sezione reported properties.

Location (oggetto) : indica la posizione geografica in cui il dispositivo si trova, in termini di latitudine e longitudine. Questa informazione è riportata dal dispositivo, nelle reported properties, nel caso in cui equipaggi una scheda di espansione avente funzionalità GPS (ovvero di tipo connect e sottotipo gnss).

4.4 Azure Functions

Il back-end dell'applicazione web ha lo scopo principale di curare l'interazione con il di front-end utente, andando ad interagire opportunamente con i servizi Azure attivi in cloud. La logica aggiuntiva necessaria a realizzare le funzionalità di salvataggio e cancellazione dei dati di telemetria, ed esecuzione delle regole e invio mail di notifica sono implementate da tre Azure Function attive nella sottoscrizione.

4.4.1 StorageTelemetryFunction

Questa funzione, scritta in C#, ha il compito di salvare i dati telemetria provenienti dall'IoT Hub nel database Azure Cosmos DB. Ha anche il compito di aggiornare il database relazionale, in particolare il campo LastActivityDate, della tabella Devices, del dispositivo autore del messaggio.

```
// Function invoked at each incoming telemetry message
[FunctionName("StorageTelemetryFunction")]
public static async Task Run(string eventHubMessage,TraceWriter log)
{

    // Log
    log.Info($"\n{eventHubMessage}\n");
    // Deserialize message
    IDictionary<string,object> deserializedeventhubmessage
        = JsonConvert.DeserializeObject<IDictionary<string,object>>(eventHubMessage);
    // Get client
    MongoClient client = GetMongoClient();
    // Get database
    IMongoDatabase database = client.GetDatabase(telemetrydbname);
    // Get collection
    IMongoCollection<BsonDocument> collection
        = database.GetCollection<BsonDocument>(telemetrycollection);
    // Build document from deserialized message
    BsonDocument documenteventhubmessage = new BsonDocument(deserializedeventhubmessage);
    // Insert document
    await collection.InsertOneAsync(documenteventhubmessage);

}
```

Figura 4.6. Azure Portal - StorageTelemetryFunction

L'attivazione della funzione, ad ogni messaggio in arrivo dall'hub, è gestita dal sistema mediante il meccanismo di binding, configurato tramite il file di configurazione function.json. I parametri di rilievo riportati sono:

- *Type*: binding di tipo trigger.
- *Name*: nome del parametro di input contenente il messaggio da processare.

- *Direction*: direzione del binding, input in questo caso.
- *Connection*: stringa di connessione relativa al IoT Hub.
- *Path*: nome dell'istanza concreta dell'hub all'interno del namespace puntato dalla stringa di connessione riportata.
- *ConsumerGroup*: gruppo di consumo logico per i messaggi in uscita dall'IoT Hub.

function.json

```
{
  "bindings": [
    {
      "type": "eventHubTrigger",
      "name": "eventHubMessage",
      "direction": "in",
      "path": "test-ds-iothub",
      "connection": "test-ds-iothub_events_IOTHUB",
      "consumerGroup": "functions",
      "cardinality": "many"
    }
  ],
  "disabled": false
}
```

Figura 4.7. Azure Portal - StorageTelemetryFunction configurations

4.4.2 DeleteTelemetryFunction

Questa funzione, scritta in C#, ha il compito di cancellare periodicamente i dati di telemetria presenti nel database e datati più di 48 ore. In particolare, la funzione è schedulata dal sistema ogni 30 minuti, e seleziona i messaggi da eliminare in base al valore del campo *time* da essi riportato.

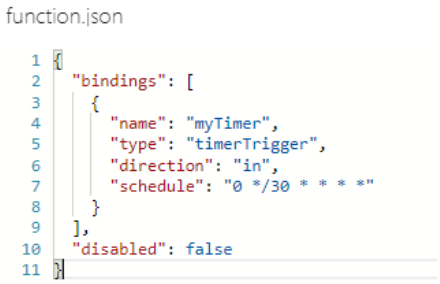

Figura 4.8. Azure Portal - TelemetryDeleteFunction configurations

La schedulazione periodica della funzione è gestita dal sistema mediante il meccanismo di binding, configurato tramite il file di configurazione function.json. I parametri di rilievo riportati sono:

- *Name*: nome del trigger.
- *Type*: tipo del trigger, in questo caso timer.
- *Direction*: direzione del trigger, anche in questo caso input.
- *Schedule*: indica l'intervallo o instante di schedulazione della funzione tramite un formato standard. In questo caso ogni 30 minuti.

4.4.3 RulePerformerTelemetryFunction

Questa funzione, scritta in Node.js, ad ogni messaggio inserito nel database, ha il compito di valutare tutte le regole associate al dispositivo autore del messaggio. In particolare, legge l'hash dell'identificativo del dispositivo contenuto nel messaggio, ne recupera le relative regole attive (locked=false) dalla tabella DeviceRules, e le esegue una alla volta. In base al valore di ritorno invierà o meno le mail di notifica all'utente associato.

```
40
41 /* Connect to */
42 function RetrieveAndPerformDeviceRules(context,devicehashid){
43     //Log
44     context.log("[RetrieveAndPerformDeviceRules] Start ");
45     var toemail = "";
46
47     /* 1- Retrieve all rules for device and runs them */
48     // Define request
49     var request = new Request(
50     "   SELECT au.email as email, dr.functionname as functionname , dr.functionbody as functionbody \
51             FROM Registration AS r \
52                 JOIN DeviceRules AS dr ON r.id = dr.deviceregistrationid \
53                 JOIN AspNetUsers AS au ON r.applicationuserid = au.id \
54                 JOIN Devices AS d ON d.id = r.devicehashid \
55             WHERE d.hashid = '"+devicehashid+"' \
56             AND dr.locked = 0 ",
57         function(err, rowCount, rows)
58         {
59             context.log(rowCount + ' row(s) returned');
60             return ;
61         }
```

Figura 4.9. Azure Portal- RulePerformerTelemetryFunction - lastMessages function

Le regole ricevono due parametri di input, ovvero due funzioni di utilità definite nel corpo della Azure Function. In particolare RetrieveDataForRule, permette di recuperare gli ultimi n messaggi di una data misura, mentre checkcallback permette di ritornare alla Azure Function l'esito della esecuzione della regola e in base a questo decidere di notificare l'utente.

```
var checkcallback = function(satisfied){

        context.log("Rule satisfied:" + satisfied);
        if(satisfied===true){
            //Send mail
            var message = {
                "personalizations": [ { "to": [ { "email": destinationemailaddress } ] } ],
                // "from": { email: from_email_address },
                // "subject": subject_email,
                "from": {email: "no-reply@stmdashboard.com "},
                "subject" : "STM Dashboard - Rule notification",
                "content": [{
                    "type": 'text/plain',
                    "value": "Rule triggered: "
                                +functionname
                                + " for device "+ devicehashid+"\n"+functionbody
                }]
            };

            context.done(null, message);
            return;
        }

        context.done(null, null);
        return;
}
```

Figura 4.10. Azure Portal- RulePerformerTelemetryFunction - $return function

L'attivazione della funzione, ad ogni messaggio storicizzato nel database, è gestita dal sistema mediante il meccanismo di binding, configurato opportunamente tramite il file di configurazione function.json. I parametri di rilievo riportati sono:

- *Type*: binding di tipo trigger.
- *Name*: nome del parametro di input contenente i messaggi aggiunti da processare.
- *Direction*: direzione di input del trigger.
- *Connection*: stringa di connessione relativa al servizio Azure Cosmos DB attivo nella sottoscrizione.
- *DatabaseName*:nome del database contenuto.
- *CollectionName*: collezione contenuta nel database specificato.
- *LeaseCollectionName*: collezione di utilità necessaria al funzionamento della Azure Function.

```
{
  "bindings": [
    {
      "type": "cosmosDBTrigger",
      "name": "inputDocuments",
      "connectionStringSetting": "stm32ode-azurecosmos-mongoapi_DOCUMENTDB",
      "databaseName": "stm32ode",
      "collectionName": "Telemetry",
      "leaseCollectionName": "leases",
      "createLeaseCollectionIfNotExists": true,
      "direction": "in"
    }
  ],
  "disabled": false
}
```

Figura 4.11. Azure Portal- RulePerformerTelemetryFunction - configurations

Nel momento in cui una regola è soddisfatta e la notifica inviata, la Azure Function aggiorna il campo Locked della tabella DeviceRules relativo alla regola in oggetto, impostandolo a false. Questo è fondamentale per evitare l'invio spropositato di email di notifica.

4.5 Struttura applicazione web

Di seguito è riportata la struttura del progetto dell'applicazione web in termini di back-end e front-end.

4.5.1 Back-end

Il back-end della soluzione è stato sviluppato seguendo, il più possibile, un approccio modulare. In particolare, nella soluzione sviluppata si possono individuare tre componenti principali:

- *Controllers*: hanno lo scopo di gestire le richieste HTTP che arrivano dalla rete, coordinando i servizi necessari a risolvere la richiesta e tornare il risultato dell'operazione.
- *Services*: implementano la logica business dell'applicazione interagendo opportunamente con altri servizi e repositories necessari.
- *Repositories*: rappresentano lo strato di interfaccia tra l'applicazione e il data storage.

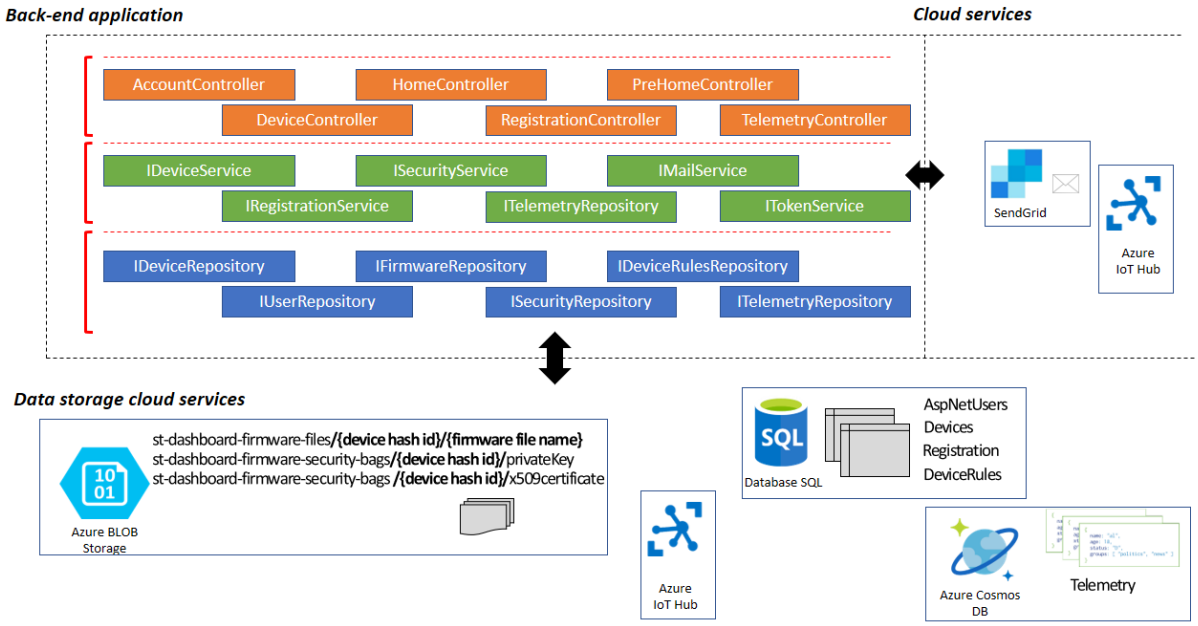

Figura 4.12. Struttura back-end

Controllers

- *PreHomeController*: ha il compito di gestire le richieste per la sezione di accesso dell'applicazione web.
- *HomeController*: ha il compito di ritornare a chi ne fa richiesta l'intera applicazione di front-end (SPA).
- *AccountController*: implementa la logica necessaria a gestire registrazione, log-in, log-out e conferma degli accounts utente.
- *RegistrationController*: espone gli end-points REST per la registrazione dei dispositivi gestendo opportunamente le richieste in arrivo.
- *DeviceController*: espone gli end-points REST per il dettaglio e regole dei dispositivi gestendo opportunamente le richieste in arrivo.
- *TelemetryController*: espone gli end-points REST per la telemetria dei dispositivi gestendo opportunamente le richieste in arrivo.

Services

- *IDeviceService*: implementa la logica necessaria ad ottenere la vista completa del dispositivo, aggiornarne stato e regole associate, controllarlo mediante invocazione metodi e caricare firmware.
- *RegistrationService*: implementa la logica necessaria a registrare un dispositivo per un utente, ed effettuare il provisioning applicativo del dispositivo.
- *ITelemetry*: implementa la logica necessaria a raccogliere dati di telemetria per dispositivi, misure e finestra temporale specificati.
- *ISecurityService*: servizio di utilità utilizzato per la generazione di chiave pubblica, chiave privata e relativo certificato X.509.
- *IMailService*: servizio di utilità per l'invio di email mediante il servizio SendGrid.
- *ITokenService*: servizio di utilità per la gestione dei JWT token necessari per l'autenticazione delle richieste HTTP REST avanzate dall'applicazione di front-end

Repositories

- *IDeviceRepository*: recupera e aggiorna i dati dei dispositivi. In particolare, interagisce con IoTHub, per accedere al twin, e la tabella Devices del database relazionale.
- *IDeviceRulesRepository*: recupera e aggiorna le regole dei dispositivi, interagendo con la tabella DeviceRules del database relazionale.
- *ITelemetryRepository*: recupera i dati di telemetria inviati dai dispositivi dal database Azure Cosmos DB.
- *IUserRepository*: recupera e aggiorna i dati circa gli utenti del sistema, interagendo con la tabella AspNetUsers del database relazionale.
- *IFirmwareRepository*: recupera e aggiunge file firmware nel BLOB storage.
- *ISecurityRepository*: recupera e aggiunge chiavi private e certificati X.509 nel BLOB storage.

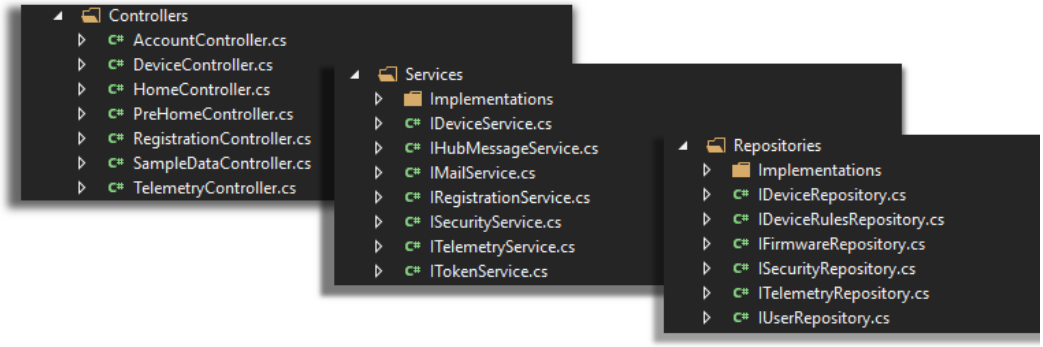

Figura 4.13. Back-end application files

4.5.2 Front-end

L'applicazione SPA che realizza la dashboard web, è stata strutturata in componenti e servizi, interagenti tra loro, al fine di renderla il più modulare possibile.

Componenti

I componenti sono oggetti, definiti nell'ambito del framework Vue, i quali contengono del codice HTML, CSS e Javascript, e che è possibile riutilizzare in più parti della stessa applicazione. La gerarchia dei componenti con la quale è stata strutturata l'applicazione è riportata in figura.

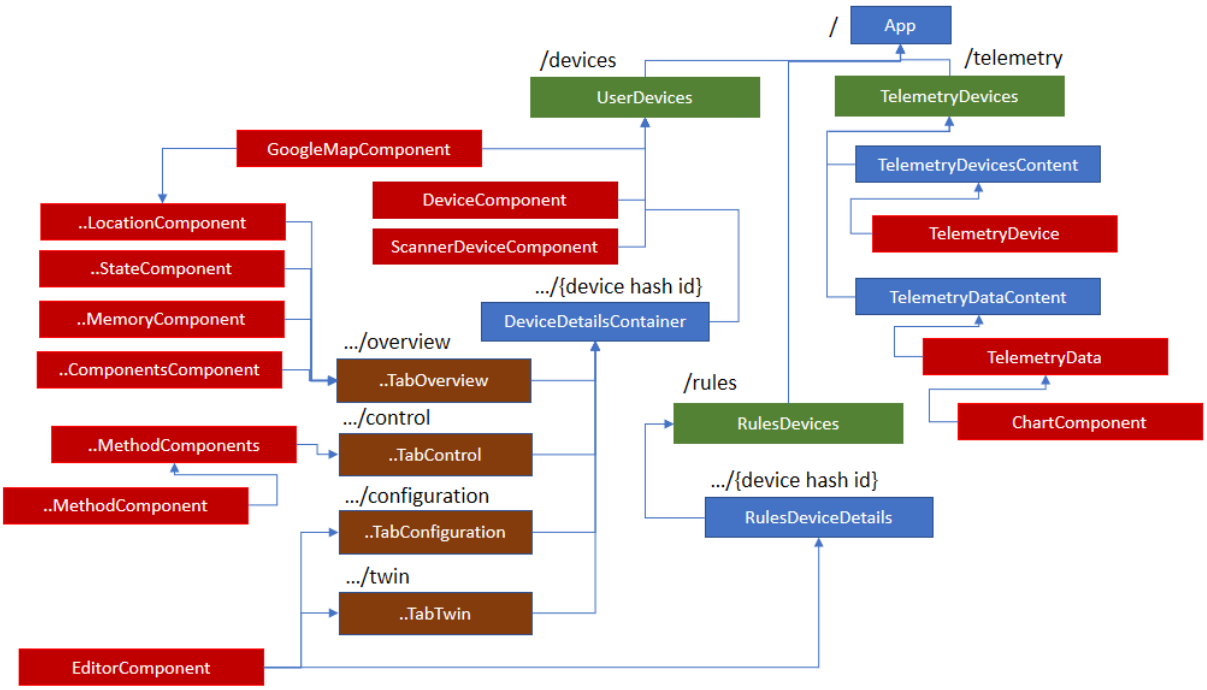

Figura 4.14. Front-end - Architettura componenti

Di seguito sono riportate le funzionalità offerte da ogni componente:
Componenti navigabili:

- *App*: entry point dell'applicazione Vue.
- *UserDevices*: vista in cui sono riportati tutti i dispositivi registrati per l'utente attualmente loggato, con possibilità di registrarne di nuovi.
- *TelemetryDevices*: vista in cui vengono riportati i dispositivi che supportano telemetria, e i relativi dati di telemetria opportunamente tracciati su grafo.
- *RulesDevices*: vista in cui sono riportati la lista dei dispositivi registrati e relative regole con possibilità di aggiunta, rimozione e aggiornamento delle stesse.
- *DeviceDetailsContainer*: vista dedicata al dispositivo selezionato, ne mostra lo stato, configurazione, twin, e permette di controllarlo mediante invocazione metodi.
- *DeviceDetailsTabOverview*: mostra nome, instante di registrazione e stato del dispositivo selezionato.

- *DeviceDetailsTabControl*: richiede e mostra i metodi supportati dal dispositivo e permette di invocarli passando all'occorrenza i parametri necessari.
- *DeviceDetailsTabConfiguration*: richiede e mostra le desired properties del dispositivo selezionato permettendone l'aggiornamento.
- *DeviceTabTwin*: richiede e mostra in sola lettura il twin associato al dispositivo selezionato.

Componenti statici:

- *DeviceComponent*: mostra graficamente un dispositivo registrato per l'utente riportando nome, stato connessione/attività e stato applicazione.
- *ScannerDeviceComponent*: permette di scansionare un QR code.
- *EditorComponent*: permette di visualizzare ed editare del testo su editor.
- *GoogleMapComponent*: permette di visualizzare graficamente un array di posizioni su mappa Google.
- *MethodComponents*: mostra la lista di metodi supportati dal dispositivo selezionato.
- *MethodComponent*: mostra un generico metodo supportato offrendo la possibilità di inserire, se necessario, parametri di input.
- *ComponentsComponent*: mostra graficamente la descrizione del dispositivo selezionato.
- *MemoryComponent*: mostra graficamente lo stato della memoria del dispositivo selezionato.
- *StateComponent*: mostra lo stato del dispositivo in termini di connettività, attività e stato applicazione.
- *LocationComponent*: mostra la posizione del dispositivo selezionato, permettendone l'aggiornamento.
- *TelemetryDevicesContent*: richiede e mostra la lista dei dispositivi che supportano telemetria.
- *TelemetryDataContent*: richiede e mostra su grafo i dati di telemetria per i dispositivi e misure selezionate.
- *TelemetryDevice*: mostra graficamente dispositivo e misure supportate dando all'utente possibilità di selezione.

- *TelemetryData*: mostra e richiede graficamente nel tempo, i dati di telemetria circa i dispositivi e misure selezionate.
- *ChartComponent*: mostra su grafico bidimensionale, valori di ascisse e ordinate passategli come array.

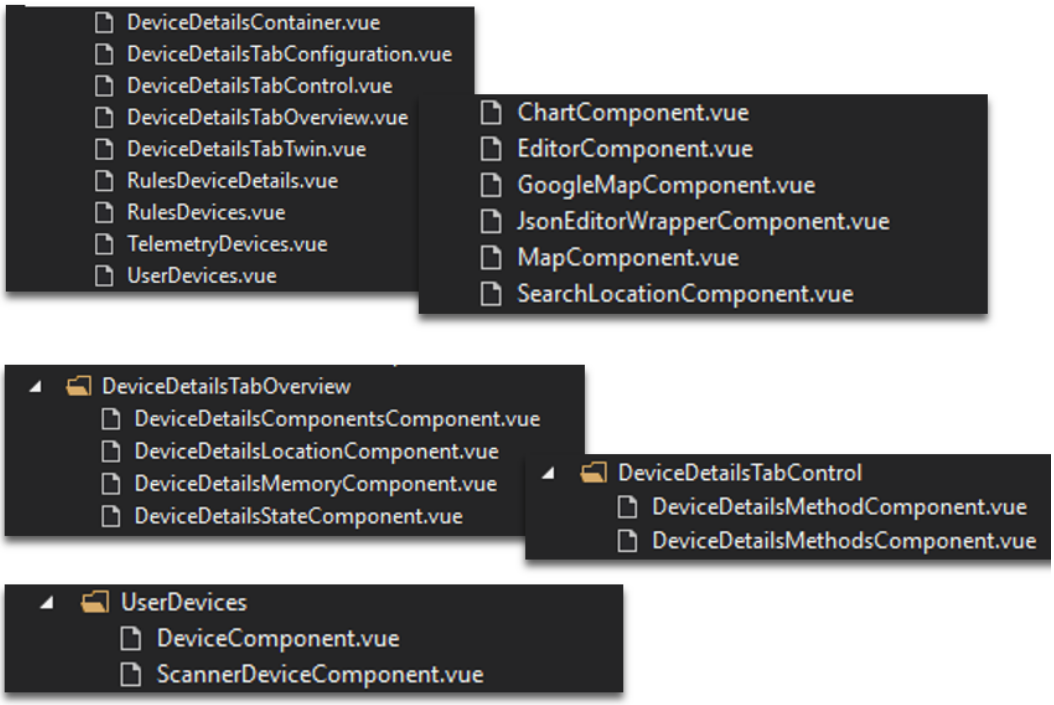

Figura 4.15. Front-end application component files

Servizi

I servizi sono utilizzati dai componenti per poter interagire con l'applicazione di back-end e non solo, e sono:

- *AuthenticationService*: ha il compito di ottenere il JWT token necessario per effettuare richieste HTTP REST autenticate.
- *DeviceService*: ha il compito di ricevere, aggiornare lo stato dei dispositivi, invocare metodi etc.
- *RegistrationDeviceService*: ha il compito di registrare dispositivi per l'utente loggato.

- *TelemetryService*: ha il compito di richiedere dati di telemetria in base a dispositivi, misure e intervallo di tempo specificato.
- *WebService*: pensato per offrire funzionalità web che vanno oltre alle funzionalità offerte dal back-end specifico della soluzione. Contiene solamente il servizio di risoluzione indirizzi necessario ad aggiornare la posizione del dispositivo.
- *NetworkTypeConverter*: ha il compito di trasformare le strutture dati ottenute come risultato delle richieste fatte in rete, in oggetti standard definiti a livello applicativo.
- *RestEndpoints*: costruisce al volo gli URLs da contattare.

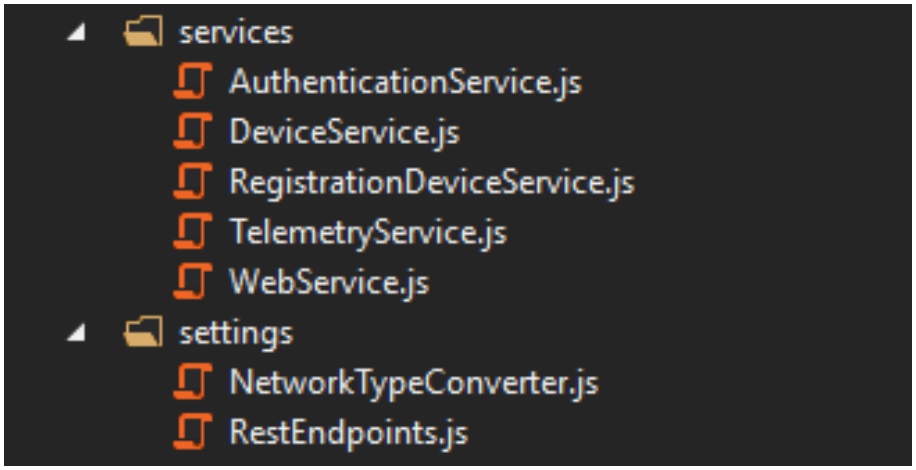

Figura 4.16. Front-end application service files

Capitolo 5

Conclusioni e sviluppi futuri

Conclusione

Durante il percorso di tesi è stata sviluppata una applicazione web che permette agli utenti a monitorare, configurare e controllare le schede elettroniche ST in loro possesso. La soluzione web è stata realizzata interamente in cloud, sfruttando i servizi messi a disposizione dalla piattaforma Microsoft Azure.

Le board ST in grado di comunicare con il back-end della soluzione alla famiglia di schede di sviluppo STM32 Nucleo e IoT Discovery Kit, e possono essere arricchite opportunamente da una o più schede di espansione per integrare sensoristica e moduli di connettività. Un firmware dedicato e disponibile a corredo delle schede (FP-CLD-AZURE1) implementa la logica necessaria per comunicare con il cloud Microsoft Azure, e con la soluzione sviluppata.

Le principali funzionalità della soluzione web implementata durante il periodo di tesi sono state le seguenti:

- Account utente.
- Registrazione di più dispositivi per utente tramite QR code.
- Storicizzazione della telemetria e visualizzazione nel tempo dei dati provenienti da uno o più dispositivi.
- Inserimento regole di notifica utente al fine di riconoscere particolari eventi telemetrici.

Le principali difficoltà riscontrate durante il lavoro sono state le seguenti:

- Modellare uno schema di registrazione dispositivo sicuro, tutelando la soluzione da possibili dispositivi cloni e/o furti di QR code validi.
- Rendere opportunamente dinamica l'applicazione web alle variazioni del twin.
- Progettare un motore Javascript robusto al fine di valutare le regole definite per i dispositivi.

Sviluppi futuri

A breve termine sono previste alcune attività aventi l'obbiettivo di consolidare la soluzione e permetterne la pubblicazione in versione rilascio. Di seguito sono riportate le attività in questione:

- Definizione di un profilo utente più dettagliato.
- Correzione e miglioramento di alcuni aspetti tecnici implementativi: in particolare verrà rimosso il meccanismo di provisioning applicativo dei dispositivi; infatti l'obiettivo a brevissimo termine è rilasciare una nuova versione firmware in grado di auto-generarsi un certificato X.509 valido a partire da un certificato root caricato sul dispositivo, mediante architetture recentemente proposte da Microsoft (RIoT).
- Testing dell'applicazione web.
- Revisione, pubblicazione del codice su repository pubblico e stesura documentazione.: questo al fine di abilitare i clienti a replicare facilmente sulla propria sottoscrizione Azure la soluzione sviluppata.

Sono previste anche attività a medio/lungo termine per sviluppare ulteriormente l'utilizzabilità della soluzione e sfruttare le informazioni statistiche raccolte:

- Introduzione di nuove funzionalità per la dashboard: sharing di dispositivi tra utenti del sistema, permettere all'utente di categorizzare i dispositivi registrati, aggiornamento in parallelo del firmware per più dispositivi, etc.
- Unione dell'account utente ST.com con quello della dashboard web: questo permetterà di profilare in modo puntuale e per utente l'utilizzo dei sistemi ST, aprendo la possibilità a operazioni pubblicitarie mirate per lo specifico utente e alla definizione di analytics sui dati statistici raccolti dalla dashboard (tipo di board, versioni del firmware, quantità dei dati generati per device, sensori utilizzati etc.).

Bibliografia

[1] URL: https://www.economyup.it/glossario/internet-of-things-definizione/.

[2] URL: https://cloudsecurityalliance.it/wp-content/uploads/2012/06/Art.-Balboni.pdf.

[3] URL: http://www.analysysmason.com/Research/Content/Reports/IoT-value-chain-Feb2017-RDME0/sample-pages-and-table-of-contents/.

[4] URL: https://it.wikipedia.org/wiki/Software_as_a_service.

[5] URL: https://it.wikipedia.org/wiki/Platform_as_a_service.

[6] URL: https://azurecomcdn.azureedge.net/cvt-7a7046a2ef74a36e357d4ac44f70b3d1bb5724fd6dadee4fdeeb4f79556ad056/images/page/overview/what-is-saas/what-is-saas.png.

[7] URL: https://www.zdnet.com/article/cloud-providers-ranking-2018-how-aws-microsoft-google-cloud-platform-ibm-cloud-oracle-alibaba-stack/.

[8] URL: http://www.st.com/en/ecosystems/stm32-open-development-environment.html.

[9] URL: http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-nucleo.html?querycriteria=productId=SC2003.

[10] URL: http://www.st.com/en/evaluation-tools/b-l475e-iot01a.html.

[11] URL: http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-nucleo-expansion-boards.html?querycriteria=productId=SC2006l.

[12] URL: http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-ode-function-packs.html?querycriteria=productId=SC2102.

[13] URL: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-overview.

[14] URL: https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview.

[15] URL: https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings.

[16] URL: https://docs.microsoft.com/it-it/azure/iot-hub/about-iot-hub.
[17] URL: https://docs.microsoft.com/it-it/azure/iot-hub/iot-hub-devguide-identity-registry.
[18] URL: https://docs.microsoft.com/it-it/azure/iot-hub/iot-hub-csharp-csharp-twin-getstarted.
[19] URL: https://docs.microsoft.com/it-it/azure/iot-hub/iot-hub-devguide-direct-methods.
[20] URL: https://docs.microsoft.com/en-us/azure/iot-dps/about-iot-dps.
[21] URL: https://docs.microsoft.com/it-it/azure/cosmos-db/introduction.
[22] URL: https://docs.microsoft.com/it-it/azure/storage/common/storage-introduction.
[23] URL: https://docs.microsoft.com/it-it/azure/storage/blobs/storage-blobs-introduction.
[24] URL: https://docs.microsoft.com/it-it/aspnet/core/?view=aspnetcore-2.1.
[25] URL: https://docs.microsoft.com/it-it/ef/core/.
[26] URL: https://it.wikipedia.org/wiki/Object-relational_mapping.
[27] URL: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176.
[28] URL: https://it.wikipedia.org/wiki/Single-page_application.