UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progettazione e realizzazione di un database cloud per monitorare la qualità delle acque costiere e fluviali tramite sensori IoT

Relatore
Dr. Filippo Campagnaro

Laureando
Michele Scapinello

ANNO ACCADEMICO 2021-2022

Ai miei genitori

Sommario

Il cambiamento climatico negli ultimi anni è diventato un problema sempre più presente. In tutto il pianeta si registrano fenomeni come aumento della temperatura, innalzamento del livello dei mari e perturbazioni atmosferiche sempre più rare e sempre più violente. L'operazione di monitorare l'ambiente è diventata quindi necessaria al fine di poter intervenire, per quanto possibile, nelle aree in cui l'impatto che questo fenomeno sta avendo è maggiore. Il progetto per il quale questa tesi è stata elaborata è volto al monitoraggio dei bacini idrici, dolci o salati, attraverso l'utilizzo di sensori IoT per verificare l'impatto che il cambiamento climatico sta avendo su essi. Nello specifico in questa tesi andremo a trattare la parte di memorizzazione dei dati misurati dai sensori attraverso la progettazione di un cloud database. La realizzazione di un cloud database favorisce quindi l'utilizzo di un servizio di cloud computing che possa avere caratteristiche adatte a poter sviluppare il database e permette di risparmiare nei costi di sviluppo e gestione di quest'ultimo. Per far fronte a questa richiesta si è deciso di utilizzare i servizi offerti da Amazon Web Services, più precisamente il servizio DynamoDB che offre la possibilità di creare database non relazionali di tipo chiave-valore. Dopo aver completato il processo di progettazione del database si è passati alla fase di implementazione mediante l'utilizzo del pannello di controllo offerto direttamente dal servizio e anche tramite le Software Development Kits messe a disposizione da AWS per il linguaggio Python. In conclusione sono stati proposti possibili sviluppi futuri del progetto attraverso l'utilizzo di altri servizi offerti da AWS utili al nostro caso di studio.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il monitoraggio di un bacino idrico è uno strumento importante per conoscere la salute dell'ambiente e viene utilizzato dagli scienziati come strumento di controllo per permettere, dove necessario, di programmare operazioni di risanamento [1]. Il monitoraggio, in campo scientifico, prevede di osservare e controllare le variazioni delle proprietà chimiche e fisiche variabili rispetto al tempo attraverso delle misurazioni e di immagazzinare tali informazioni per renderle disponibili a sistemi analitici e personale qualificato. L'opera di monitoraggio manuale di un bacino idrico richiede un impegno di personale e di tempo considerevoli, senza considerare gli eventuali rischi che il personale potrebbe incontrare in zone poco accessibili e gli imprevisti che la natura può creare. Con lo svilupparsi delle tecnologie dell'informazione, tuttavia è diventato possibile eseguire queste operazioni in modo continuo ed efficace grazie alla vasta gamma di sensori disponibili in commercio, ognuno dei quali è dedicato a misurare una proprietà specifica. Un sensore nel significato più classico del termine, è un singolo apparecchio in grado di rilevare le variazioni di una grandezza fisica nel tempo come ad esempio un sensore di temperatura o un sensore di umidità [2]. Nel testo di questa tesi con il termine sensore invece, identifichiamo un sistema composto da diversi sensori collegati tutti alla stessa unità centrale e quindi in grado di rilevare diverse proprietà attraverso i singoli sensori e di salvare i dati in apposite strutture. La necessità di effettuare analisi statistiche e qualitative dei bacini idrici rende necessario raccogliere un quantitativo considerevole di dati, che andrebbero persi o mischiati se non gestiti correttamente e immagazzinati in appositi sistemi, chiamati database. Molti di quest'ultimi sono oggi messi a disposizione di utenti e aziende attraverso i cloud services, servizi che permettono di interagire, elaborare ed archiviare i dati, attraverso l'utilizzo della rete. Molto spesso si tende ad utilizzare il termine cloud per indicare un servizio che offre la possibilità di salvare i nostri dati online e accedervi da qualsiasi dispositivo, come i più utilizzati Google Drive, OneDrive e iCloud drive. Da questa considerazione si deduce che un cloud service è un servizio fornito da remoto in grado di essere eseguito sul nostro dispositivo personale riducendo notevolmente l'utilizzo delle sue risorse hardware ma sfruttando la connessione di rete. Con il continuo evolversi del settore industriale e con la possibilità di poter usufruire di infrastrutture di rete migliori, sempre più aziende hanno iniziato ad affacciarsi al mondo del cloud computing e nell'Unione Europea tra il 2020 e il 2021, si è assistito ad un incremento dell'utilizzo dei servizi cloud di circa il 5% [3]. L'utilizzo di questi servizi di solito prevede un costo in base alla quantità di risorse richieste ed al tempo di utilizzo ma molto spesso questo può variare in base a quale fornitore si decide di utilizzare. Al giorno d'oggi il più grande è Amazon, che attraverso la sua azienda, Amazon Web Services, copre oltre il 50% del mercato

[4]. In questa tesi andremo ad analizzare quale modello di database sia più indicato alle nostre esigenze e successivamente sarà presentata una soluzione per la nostra realtà di sviluppo. Nello specifico nel Capitolo 2 sarà analizzata la struttura di un database basato sul modello logico relazionale e quella di un modello logico non relazionale, evidenziando i vantaggi e le limitazioni che si potrebbero incontrare nel loro utilizzo. Nel Capitolo 3 verrà descritto il servizio di cloud database che sarà utilizzato per lo sviluppo del progetto e come esso è stato strutturato per favorire un'implementazione più semplice e una gestione meno costosa. Nel Capitolo 4 si presenterà un manuale per l'implementazione del database e infine nel Capitolo 5 si proporranno una serie di implementazioni possibili con altri servizi cloud che potrebbero operare in sinergia con quello da noi utilizzato.

Capitolo 2

Database

Un database, in italiano base di dati, è un sistema di archiviazione dati elettronico che nel caso di ingenti quantità di dati salva quest'ultimi in servizi di archiviazione cloud e cluster di computer. L'architettura di un database può essere divisa in tre livelli [5]:

- il livello fisico, con il quale identifichiamo l'hardware, ovvero i dischi fisici che contengono i dati e ne garantiscono la persistenza, cioè la capacità di durare nel tempo, e il processore per l'elaborazione dei dati stessi;
- il livello logico, attraverso il quale si identifica la struttura logica del database, come i dati vengono salvati al suo interno e quali sono i vincoli da rispettare su di essi;
- il livello software, si riferisce ai programmi capaci di eseguire le fondamentali operazioni di create, read, update, delete (CRUD), indispensabili per la gestione e manipolazione dei dati. I software che possiedono tali caratteristiche sono definiti database management system (DBMS), e operano attraverso il linguaggio di programmazione SQL.

2.1 Progettazione di un database

La progettazione di una base di dati è una delle componenti chiave del processo di sviluppo, in quanto si definiscono la struttura e le caratteristiche proprie del database, ma se effettuata senza le dovute accortezze può risultare un'operazione complicata. Per facilitare lo sviluppo quindi, è utile visualizzare le procedure di realizzazione da un punto di vista più ampio. Il primo passo è quello di raccolta e analisi dei requisiti che consiste nell'individuare le funzionalità che il sistema dovrà svolgere. Attraverso l'interazione con gli utenti si stila una descrizione dei dati e delle operazioni che si andranno ad effettuare su di essi. Si passa successivamente alla fase di progettazione dove vengono definite le caratteristiche del database, qui si definisce "cosa devono rappresentare i dati", un'operazione che viene spesso effettuate in parallelo all'analisi dei requisiti, e "come deve memorizzare i dati il database", anche chiamata progettazione logica del database. Quando la fase di progettazione è completata si passa a quella di implementazione in cui si produce il codice per il funzionamento del database e dei programmi destinati a gestirlo. Infine dopo un'operazione di collaudo il database viene messo in funzione ed è pronto ad eseguire il suo lavoro. La descrizione appena fornita non deve per forze essere eseguita in sequenza. Può infatti capitare che in fase di progettazione alcuni aspetti definiti

precedentemente vengano rivisitati per rispettare alcuni vincoli imposti o per semplificare le fasi successive.

Analisi dei requisiti e progettazione dei dati

L'ambiente di sviluppo da analizzare è, come definito dal titolo, il monitoraggio delle acque attraverso sensori IoT. Per prima cosa è importante definire quali proprietà della materia è richiesto misurare per avere poi una panoramica sui tipi di dato che andremo ad inserire all'interno del database. Visto che il progetto è destinato sia a corpi di acqua salata, come mari e lagune, che a corpi di acqua dolce, come fiumi e laghi, i dati da raccogliere sono di vario tipo. Per il momento sappiamo che i sensori utilizzati rilevano le seguenti proprietà dell'acqua:

- conducibilità elettrica, indice della salinità dell'acqua [6];
- temperatura [7];
- livelli di ossigeno nell'acqua [8];
- pressione, in base alla profondità [9];
- pH [10];
- quantità di solidi dissolti nell'acqua per milione di unità [11].

I dati vengono rilevati dai sensori, inviati tramite una rete wireless ad un server cloud e poi inseriti all'interno del database. Le proprietà diverse richiedono un numero diverso di misurazioni, ad esempio il livello di ossigeno può essere misurato ogni 15 minuti a differenza della temperatura dove una misura per ogni ora è sufficiente. Se proviamo a dare un senso in termini di tipo di dato alle proprietà prima elencate, possiamo dire che tutte prevedono la restituzione di un singolo valore che possiamo decidere se essere un intero o una stringa a seconda se si decida di memorizzare l'unità di misura o meno. Considerandoli in un aspetto generale, si può affermare che i dati sono semplici, composti da un singolo valore. Per distinguere le misure tra di loro all'interno del database è utile indicare anche:

- il sensore che effettua la misurazione, attraverso il codice univoco che identifica il prodotto (ID);
- l'istante di tempo in cui si è effettuata la misura;
- le coordinate del sensore.

Prendiamo ora in considerazione eventuali aggiornamenti alle misurazioni che possono essere eseguiti una volta terminato il progetto o in fase di sviluppo. Al fine di favorire una più ricca e dettagliata misura delle proprietà dell'acqua e del suo ambiente si potrebbero aggiungere sensori per rilevare la distribuzione spaziale e temporale della fauna ittica, rilevamenti batimetrici per lo studio della morfologia dei fondali marini o sensori per la misurazione di eventi geologici. Alcune di queste misurazioni, a differenza di quelle indicate precedentemente, non sono dei semplici dati a singolo valore ma richiedono un'elaborazione più complicata, in quanto è più complicata la misura stessa. Si parla quindi di strutture di dati complesse, come ad esempio mappe, array o insiemi di centinaia di dati da memorizzare all'interno del database. Questo fattore avrà un ruolo decisivo nella scelta del modello di database da utilizzare.

2.2 Il modello relazionale

Il modello relazionale, definito da Edgar F. Codd nel 1970 [12], è un modello logico utilizzato per la rappresentazione o strutturazione dei dati di un database. La struttura fondamentale del modello relazionale è la relazione, intesa a livello matematico come sottoinsieme di un prodotto cartesiano tra due insiemi, rappresentata graficamente da una tabella, composta da un numero variabile di righe, dove ogni riga viene identificata con il termine tupla, e un numero definito di colonne identificate da un nome, dette anche attributi. Una relazione del modello relazionale viene identificata con la sintassi $R(X)$ dove con R si indica il nome della relazione mentre, il termine X identifica un insieme di attributi ordinati $X = \{X_1, X_2, \ldots, X_n\}$ ai quali viene di solito assegnato un nome e un dominio, $dom\{X_j\}$, che deve essere rispettato in fase di inserimento dei dati.

STUDENTE(Matricola, Nome, Cognome, Data di Nascita)

Figura 2.1: Rappresentazione di una relazione.

Nell'esempio si nota che l'attributo Matricola risulta essere sottolineato. La sottolineatura di un attributo lo identifica come chiave della relazione, ovvero, tale attributo ha il vincolo di dover essere univoco per ciascuna tupla, poiché la identifica all'interno della relazione stessa.

Matricola	Nome	Cognome	Data di nascita
1204511	Michele	Rossi	10/10/1996
1245322	Mario	Bianchi	21/07/1998
1198745	Luca	Verdi	30/05/1997

Figura 2.2: Rappresentazione grafica di una relazione.

In alcuni schemi relazionali è inoltre possibile trovare chiavi che fanno riferimento ad attributi di altre relazioni, necessari per evidenziare che tra le relazioni è presente un collegamento. Tali chiavi vengono definite chiavi esterne. Per sviluppare correttamente un modello relazionale è necessario rispettare dei vincoli, definiti vincoli di integrità, all'interno della tabella, collegati al concetto di chiave prima introdotto, ma anche vincoli tra le relazioni.

2.2.1 Vincoli di integrità

I vincoli di integrità sono proprietà che devono essere soddisfatte durante la progettazione del modello relazionale di un database, in caso contrario si potrebbe incorrere in gravi errori di coerenza tra i dati, come la presenza di tuple duplicate all'interno di una relazione. Un vincolo può essere visto come un predicato che, collegato a determinati aspetti di una relazione, o agli elementi che la compongono, può assumere valore vero o falso, a seconda che venga rispettato o meno. I vincoli relazionali, possono essere distinti in due categorie:

- vincoli intrarelazionali: vincoli sulla singola istanza o sul singolo valore di una tupla;

- vincoli interrelazionali: vincoli tra relazioni, importante per la coerenza e l'integrità dei dati.

Vincoli intrarelazionali

I vincoli intrarelazionali sono i vincoli da rispettare all'interno della relazione stessa, tutte le istanze di una relazione devono soddisfare tutti i vincoli che sono definiti sulla relazione stessa. Questi vincoli sono:

- il vincolo di dominio che indica un dominio di appartenenza dom(X) per l'attributo X;
- il vincolo di chiave primaria molto importante in quanto permette di identificare univocamente una singola tupla all'interno della tabella;
- il vincolo di chiave simile al vincolo di chiave primaria con la differenza che in questo caso è possibile assumere valori nulli.

Vincoli interrelazionali

Il vincolo interrelazionale o vincolo di integrità referenziale è estremamente importante. Viene definito tra le relazioni ed è necessario al fine di preservare la coerenza nelle relazioni tra tabelle. Si basa sul concetto di chiave esterna. Tale vincolo impone che, all'interno di una tabella, il campo dichiarato come chiave esterna (foreign key) possa assumere solamente valori appartenenti al dominio della chiave primaria presente nella tabella "padre".

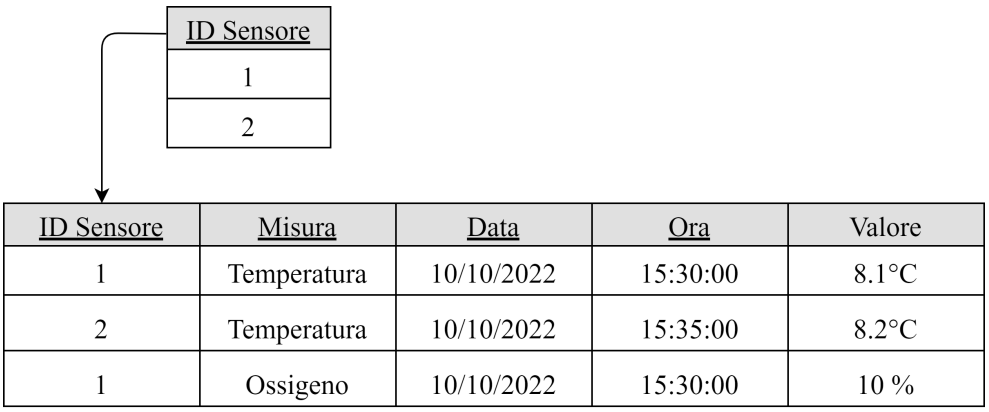

Figura 2.3: Rappresentazione grafica del modello relazionale.

2.2.2 Il linguaggio SQL

Il linguaggio Structured Query Language (SQL), è un linguaggio di programmazione nato nel 1974 per mano di Donald Chamberlain, che si basa sull'argomento teorico dell'algebra relazionale e, attraverso delle stringhe chiamate query, interrogazioni in italiano, permette di accedere ai dati memorizzati all'interno del livello fisico del database, attraverso l'utilizzo di parole chiave o simboli, chiamate operatori che si collegano ai concetti dell'algebra relazionale. Il linguaggio SQL, si identifica in 4 strutture principali:

- Data Definition Language (DDL) per creare, modificare ed eliminare le strutture del database;
- Data Query Language (DQL), per derivare informazioni dai dati contenuti nel database;
- Data Manipulation Language (DML), permette di inserire, modificare ed eliminare dati all'interno del database;
- Data Control Language (DCL), abilita il programmatore a fornire i permessi agli utenti per poter utilizzare DML, DDL e anche lo stesso DCL.

Attraverso una precisa sintassi è possibile quindi interrogare il database per effettuare diverse operazioni. SQL offre la possibilità di effettuare anche query nidificate per affinare la ricerca all'interno del database.

2.3 Le limitazioni del modello relazionale

Come è stato discusso precedentemente, nel modello relazionale i dati vengono memorizzati all'interno di tabelle e si utilizza il linguaggio SQL per manipolarli. Nonostante sia tra i più usati nelle applicazioni software risulta essere limitato per alcuni casi di sviluppo. Consideriamo ora la proprietà di scalabilità ovvero la capacità di un database di mantenere elevate le proprie prestazioni all'aumentare della quantità di dati, che si può tradurre in velocità di risposta delle query e costi di utilizzo contenuti in contesto cloud database. Scalando un database orizzontalmente si aggiungono nuovi database che funzionano in modo parallelo tra di loro, con la stessa struttura dell'originale. Scalando verticalmente invece, si va ad aumentare la potenza di calcolo del database attraverso aggiornamenti hardware. Nel contesto del cloud database, entrambe le logiche di scaling prevedono dei costi nel momento in cui vengono attuate. Non è semplice definire se una sia più costosa dell'altra, in quanto ciò dipende dall'utilizzo che il database è destinato ad avere, dalla sua struttura, che come si vedrà in seguito può portare a limitazioni in alcuni casi, e al budget di sviluppo disponibile per il progetto.

Considerando il modello relazionale l'approccio di scalabilità orizzontale può portare a situazioni di difficoltà. Dividere un database relazionale in più macchine aumenta il carico di query da dover eseguire per estrarre i dati, che in alcuni casi si traduce in un aumento dei costi, se essi si basano sul numero di letture e scritture che si effettua sul database. Per estrarre dati specifici, molte volte è necessario collegare due tabelle tra di loro attraverso gli operatori del linguaggio SQL e quindi, nel caso in cui il database sia diviso in due o più macchine, sarebbe necessario estrapolare i dati da ciascuno di essi per poi unirli sotto un unico risultato. Operazione che diventa ancora più dispendiosa e complessa se si considera che nel cloud database è necessario effettuare anche delle chiamate web per ottenere i dati di partenza.

Un'altra limitazione del modello relazionale è dovuta al fatto che i dati al suo interno sono strutturati. In fase di progettazione vengono definite tabelle con attributi di numero definito, ai quali si assegnano dei valori contenuti in un dominio. Questo può risultare vantaggioso dal punto di vista della manipolazione e della facilità di interrogazione ma presenta lacune in quanto i formati di dati supportati sono limitati ed è richiesta una gestione speciale in alcuni casi. Prendiamo ad esempio un dato di tipo array, ovvero una sequenza, ordinata o non, di dati, che possono essere anche eterogenei tra di loro. Salvare un array all'interno di un database relazionale risulta

essere un'operazione complicata. Non è possibile infatti memorizzare l'intero array come valore ed è quindi necessario dividere i suoi elementi e salvarli singolarmente all'interno del database, causando un aumento considerevole delle risorse utilizzate e di query da eseguire, nonché dei costi di gestione e di utilizzo.

2.4 NoSQL

I database NoSQL [13], acronimo che sta ad indicare Not only SQL, sono diventati sempre più popolari grazie alla loro struttura libera, non relazionale, praticamente priva di vincoli. Le aziende al giorno d'oggi utilizzano database NoSQL per salvare enormi quantità di dati ed interrogare i database con milioni di query, non solo tramite linguaggio SQL, ma anche attraverso un accesso ad oggetti. Eliminando le relazioni tra le tabelle, i dati risultano essere indipendenti l'uno dall'altro, e all'interno del database sono salvati attraverso una coppia key-value, chiave-valore.

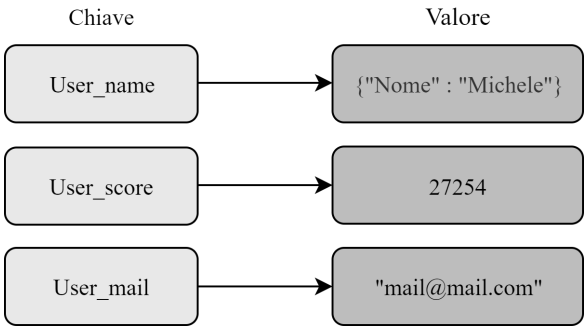

Figura 2.4: Rappresentazione grafica di un database chiave-valore.

Ogni singolo oggetto del database è identificato univocamente tra tutti da una chiave univoca e ad ogni chiave corrisponde un valore, che può presentarsi come un intero file di testo o di altre tipologie. Questa struttura permette di effettuare uno scaling orizzontale in modo semplice, se le risorse di un database non dovessero più essere sufficienti per gestire il carico di lavoro, inserire un secondo database parallelo al primo per dividere il carico di lavoro non causerebbe alcun tipo di problemi.

Un'altra caratteristica dei database NoSQL è quella di essere schemeless cioè privi di schemi. Ciò significa che i dati all'interno di un database posso essere di natura diversa tra di loro, presentandosi quindi con uno schema non strutturato, i cosiddetti unstructured data, garantendo flessibilità a differenza del modello relazionale. A seconda di come i dati vengono salvati all'interno di un database NoSQL, è possibile distinguere una serie di principali tipologie, elencate come segue [14]:

- database orientati al documento: i dati vengono salvati sotto forma di documenti e ogni documento ha una chiave che lo identifica all'interno del database. Di solito si tratta di documenti JSON, BSON, XML e in alcuni casi anche PDF;

- database a grafo: in questo caso vengono utilizzati nodi e archi per rappresentare ed archiviare le informazioni. I nodi ospitano gli elementi, gli archi invece, le relazioni tra i nodi;

- database chiave-valore: è il modello più semplice di database non relazionale. La chiave è l'identificatore univoco del dato che può essere un semplice oggetto o una struttura più complicata;

- database orientati a colonne: in questo modello i dati vengono organizzati in colonne a differenza del modello relazionale che salva i dati riga per riga.

2.5 Le limitazioni dei database NoSQL

Se da una parte un modello dati non relazionale può risultare vantaggioso sotto alcuni punti di vista, ci sono comunque dei problemi di cui tener conto anche in questo caso.

La struttura non relazionale, e la conseguente indipendenza dei dati, può portare ad una mancata integrità dei dati all'interno del database, con la presenza di dati duplicati e quindi uno spreco di risorse che sarebbe evitabile nel caso di utilizzo di un database relazionale che normalizza i dati secondo i vincoli imposti. A differenza del modello relazionale, che utilizza il linguaggio SQL per estrapolare i dati, nei modelli NoSQL tale operazione può essere eseguita diversamente. È possibile accedere ai dati attraverso una programmazione ad oggetti, mirata per un modello specifico di database ma questo risulta essere un problema se non si conosce la natura e la struttura del database, poiché per un utente esterno risulterebbe complicato effettuare operazioni di ricerca al suo interno se non attraverso un programma guidato che si occupa di tradurre in query le richieste dell'utente.

2.6 Selezione del modello da utilizzare

Dopo aver analizzato i modelli di database indicati per lo sviluppo della nostra applicazione è il momento di scegliere quale tra quelli menzionati sia il più adatto. I punti fondamentali che hanno condizionato la scelta del modello sono stati la tipologia e struttura dei dati da raccogliere, l'eventuale necessità di effettuare operazioni di scaling e la flessibilità nel caso di aggiornamenti futuri dell'applicazione. Abbiamo visto come un database relazionale sia più adatto a scalare verticalmente, mentre come un database non relazionale possa scalare sia verticalmente che orizzontalmente. Per il caso preso in esame, lo sviluppo in orizzontale è più indicato in quanto il nostro obiettivo è quello di raccogliere diverse tipologie di dati non note tutte a priori al momento della progettazione per poi effettuare analisi su di essi in un secondo momento. Ecco che il modello non relazionale quindi risulta essere quello più favorevole per questo scopo. Senza le relazioni tra le tabelle il nostro database è libero di accogliere qualsiasi tipo di dato, senza dover considerare eventuali problemi derivanti da dati ridondanti e senza dover effettuare un quantitativo troppo elevato di query, che andrebbero ad impattare anche sul costo di utilizzo e mantenimento del database stesso. Da questa considerazione è possibile trarre un altro aspetto vantaggioso sull'utilizzo di tale modello, ovvero la libertà sulla struttura dati. Come indicato nel Capitolo 2.4, il modello relazionale è limitato a contenere solamente alcuni tipi di dato, definiti comunemente dati strutturati. L'esempio proposto sulla difficoltà di memorizzare un array di dati in un modello relazionale mette in evidenza le affermazioni precedenti. In un ambiente di sviluppo in cui gli aggiornamenti sui dati da raccogliere potrebbe avvenire in qualsiasi momento, la struttura non relazionale risulta essere più vantaggiosa. Se utilizzassimo il modello relazionale potrebbe essere necessario dover riadattare il database alle nuove implementazioni e la riprogettazione del modello relazionale sarebbe un'operazione costosa in termini di tempo e risorse da dover effettuare. Ecco che la capacità del modello non relazionale di memorizzare dati non strutturati, indipendenti tra di loro, risulta essere molto van-

taggiosa. Ciascun dato potrebbe essere salvato attraverso una coppia chiave-valore o tramite l'utilizzo di documenti, come i file JSON, in cui è possibile memorizzare dati in modo semplice, per poi accedervi tramite degli algoritmi di programmazione. Per quanto esposto quindi, la scelta di sviluppo si è indirizzata verso l'utilizzo del modello non relazionale. La tipologia di modello e la sua progettazione saranno presentate nel capitolo seguente.

Capitolo 3

Cloud computing e AWS

Amazon Web Services (AWS) è un'azienda del gruppo Amazon che fornisce servizi di cloud computing. Il cloud computing consiste nella distribuzione su richiesta di risorse per lo sviluppo IT tramite internet, con un costo che si modella in base al consumo. Alcuni dei vantaggi del cloud computing possono essere riassunti brevemente nei seguenti punti [15]:

- agilità, intesa come la possibilità di sviluppare un nuovo software senza doversi preoccupare del mantenimento delle risorse;
- elasticità, ovvero l'abilità di poter adattare le risorse in base ai livelli di attività del software;
- costi contenuti, spese fisse evitate in favore di una spesa variabile.

AWS offre più di 200 servizi a livello globale nei settori di cloud computing, con lo scopo di gestire le componenti di un'infrastruttura IT in modo che l'utente possa concentrarsi sullo sviluppo e la crescita di essa. Le tipologie principali di cloud computing sono identificate in tre livelli [16] discussi nel seguito.

- Infrastructure as a Service (IaaS), del quale fanno parte i servizi dedicati alla disposizione di risorse hardware come ad esempio server fisici o dischi fissi di memoria. L'utente tuttavia non deve occuparsi dell'aggiornamento di questi ma semplicemente addattare le risorse quando necessario.
- Platform as a Service (PaaS), dove sia la parte hardware che software sono ospitate dal fornitore e l'utente si occupa di gestire solamente le applicazioni e i dati.
- Software as a Service (SaaS), identifica i servizi che offrono software gestiti direttamente dal fornitore, dove l'utente finale interagisce solamente con il programma, senza occuparsi di aggiornamenti.

3.1 AWS DynamoDB

AWS offre una serie di servizi per la memorizzazione dei dati, il che si collega alla nostra necessità di sviluppo. Sono presenti sia database basati sul modello relazionale, sia database NoSQL e l'utente può scegliere il più adatto a seconda delle esigenze. Per lo sviluppo del nostro database si è scelto di adottare il servizio DynamoDB e dopo un'analisi della sua struttura si è passati ad una progettazione

adatta a memorizzare i dati richiesti per gestire al meglio i costi e il salvataggio dei dati. DynamoDB è un servizio che offre un database NoSQL completamente gestito, di tipo chiave-valore. Tendenzialmente, se fossimo in possesso del nostro database, sarebbe necessario gestire la parte hardware, la parte di sicurezza dei dati e il successivo aggiornamento e mantenimento del database. Attraverso DynamoDB queste operazioni non sono di competenza dell'utente ma vengono eseguite direttamente dal fornitore del servizio. DynamoDB fornisce una memoria virtuale potenzialmente illimitata ed è in grado di gestire grandi quantità di traffico di comandi senza impattare negativamente sulle prestazioni [17]. Per poter progettare al meglio il database è stata eseguita un'analisi della sua struttura e dei suoi componenti principali.

Tabelle, oggetti e attributi

Per rappresentare la struttura fisica del database non relazionale che andremo ad utilizzare è utile identificare tre concetti base, che ci permettono di dare un'interpretazione grafica di come i dati vengono salvati all'interno del database. Tale rappresentazione può essere vista come una struttura concettuale dei dati che viene definita attraverso il semplice utilizzo di [17]:

- tabelle, simili a quelle di un database relazionale, dove vengono salvati i dati. Una tabella è intesa come una collezione di oggetti;
- oggetti, intesi come collezione di attributi e contenuti nelle tabelle. Simile al concetto di tupla nei database relazionali, in DynamoDB non esistono limiti al numero di oggetti che si possono salvare in una tabella;
- attributi, compongono gli oggetti, simile alle colonne di un database relazionale.

Le tabelle supportano le classiche operazioni di create, update e delete che possono essere utilizzate tramite prompt dei comandi, opportune librerie di programmazione o semplicemente utilizzando il pannello di controllo che il servizio AWS mette a disposizione dell'utente che rende più semplice la gestione del database. Nel caso della creazione di una nuova tabella si procede attraverso una guida passo a passo, tramite una semplice interfaccia grafica, in cui ci verrà chiesto di specificare un nome per la tabella, una chiave primaria e altre opzioni più avanzate.

Chiave primaria

Nel momento in cui si procede alla creazione della tabella attraverso il pannello di controllo del servizio DynamoDB, è richiesto di specificare un valore, definito chiave primaria, che sarà necessario per identificare univocamente gli oggetti all'interno della tabella, infatti non possono esistere due elementi all'interno della tabella con lo stesso valore di chiave primaria. Con DynamoDB abbiamo la possibilità di strutturare la chiave primaria in due modi [17]:

- partition key, una chiave primaria semplice composta da un singolo attributo. In una tabella con una chiave primaria semplice non possono esistere due oggetti con la stessa partition key. Questa chiave viene definita partition key poiché il suo valore viene utilizzato in una funziona hash che determina un valore utilizzato per individuare la partizione fisica in cui viene salvato l'oggetto;

- partition key e sort key, una chiave primaria complessa composta dalla concatenazione di due attributi, uno definito partition key, l'altro definito sort key. In questo caso è possibile che alcuni oggetti all'interno della tabella abbiano lo stesso valore di partition key e che di conseguenza siano salvati tutti nella stessa partizione fisica, tuttavia vengono distinti attraverso la sort key e ordinati in base al valore di quest'ultima.

In entrambi i casi non possono esistere due valori di chiave primaria uguali all'interno della stessa tabella. Confrontando le due strutture possiamo dire che l'utilizzo di una chiave primaria complessa renderebbe più semplici e logiche le query di ricerca sui dati contenuti nel database poiché sarebbe possibile utilizzare un attributo in più assegnato alla sort key come variabile. La possibilità di poter aver partition key con lo stesso valore rende disponibile la scelta di alcuni attributi per tale campo che invece non sarebbe possibile utilizzare se si utilizzasse un approccio basato sulla chiave primaria semplice.

Tipi di dato supportati

La struttura non relazionale del servizio DynamoDB permette di memorizzare dati di praticamente qualsiasi tipo divisibili nelle seguenti categorie [17]:

- scalari, che rappresentano un singolo valore consistono in numeri, stringhe, valori binari, valori booleani e valori nulli;
- documenti, intesi come strutture di dati complesse con attributi annidati. Fanno parte di questa categoria le strutture mappa e lista;
- insiemi, interpretabili come una collezione di dati scalari, definiti set.

Quando viene creata una tabella e la sua corrispondente chiave primaria, è necessario definire il tipo di dato della chiave. In questo caso vengono imposti dei vincoli sulla selezione del tipo e la chiave primaria è abilitata a supportare solamente i dati di tipo stringa, numerico o binario. Una volta fatto ciò non è necessario definire il tipo di dato degli altri attributi grazie alla struttura schemaless e senza relazioni a differenza del modello relazionale dove è necessario specificare il tipo di dato per tutti gli attributi in fase di creazione del database.

Sistema di interrogazione del database

Per interrogare il servizio DynamoDB ed estrapolare i dati di nostro interesse abbiamo a disposizione due approcci [17]:

- PartiQL, un linguaggio compatibile con SQL per effettuare le operazioni di create, read, update e delete. Puo essere utilizzato attraverso la console di AWS, riga di comando o tramite le API dedicate;
- DynamoDB API, librerie di codice che permettono attraverso un approccio ad oggetti di accedere ed estrapolare i dati.

Si è deciso di utilizzare le DynamoDB API per effettuare le operazioni di inserimento e interrogazione del database poiché lo stesso servizio quando interrogato fornisce come risposta alle query una collezione di oggetti o un singolo oggetto, a seconda del comando utilizzato e della query effettuata. Nel sisma di interrogazione

sono presenti alcune limitazioni imposte sugli attributi identificati come partition key e sort key. Per la partition key è obbligatorio specificare il nome e un solo valore attraverso un'operazione di uguaglianza, ciò significa che gli operatori matematici e di confronto non sono ammessi. Per quanto riguarda la sort key invece le operazioni applicabili sono diverse ma è possibile utilizzarne solamente una per interrogazione tra le seguenti [17]:

- operatori matematici come =, <, >, >=, <=;
- begins_with (a, substr), che restituisce vero se l'attributo inizia con una sequenza di caratteri particolare;
- operatore BETWEEN a AND b che restituisce vero se l'attributo è compreso nel range specificato.

Le librerie di DynamoDB mettono a disposizione dei metodi per poter filtrare ulteriormente i risultati ottenuti. L'operazione di filtraggio dei dati può essere effettuata solamente sui campi non indicati come partition key e sort key e concede la possibilità di applicare un numero a piacere di filtri mantenendo tuttavia il vincolo di poter applicare un'operazione per attributo, senza impattare negativamente sulle risorse utilizzate per effettuare l'operazione di query. Le operazioni vengono eseguite nell'ordine secondo cui prima viene eseguita la query in base alla partition key e, in modo facoltativo anche in base alla sort key, dopodiché si applicano gli eventuali filtri scelti dall'utente. L'operazione di filtraggio può utilizzare tutti gli operatori che sono applicabili alle chiavi con l'aggiunta di [17]:

- operatore non uguale (<>);
- operatore OR;
- operatori insiemistici CONTAINS e IN;
- operatore EXISTS;
- operatore NOT EXISTS
- operatore SIZE.

La sequenza con cui vengono eseguite le operazioni di query e filtraggio fa notare un piccolo dettaglio relativo alle letture che si effettuano. Il problema di non strutturare la chiave primaria considerando le limitazioni imposte in fase di interrogazione si presenta nel fatto che le operazioni di query non eseguite correttamente potrebbero restituire oggetti che al loro interno non presentano dati di nostro interesse traducendosi in uno spreco di risorse, evitabile invece se tramite partition key e sort key si riuscisse a selezionare da subito i dati desiderati senza il bisogno di operazioni di filtraggio successive. L'approccio migliore sarebbe quindi quello di utilizzare solamente i valori della chiave primaria per interrogare il database.

3.2 Progettazione DynamoDB per il caso di studio

Nel Capitolo 1 della tesi è stato esplicato lo scopo del nostro database ovvero quello di memorizzare i dati provenienti dalle misurazioni effettuate da sensori posizionati in ambiente idrico. In questo capitolo abbiamo presentato la struttura del servizio che verrà utilizzato per svolgere tale compito ed è quindi ora necessario definire alcune caratteristiche che il database dovrà soddisfare per facilitare l'inserimento e l'interrogazione dei dati. Dopo un primo periodo di immagazzinamento dei dati, essi verranno estrapolati attraverso le query, alcune delle quali potrebbero essere:

- restituisci i valori misurati da un determinato sensore;
- restituisci il valore medio di una determinata proprietà;
- restituisci i valori di una determinata proprietà misurati in un determinato lasso di tempo.

Un altro aspetto da considerare è che non saranno presenti query di aggiornamento o di cancellazione dei dati, l'utente finale a cui è destinato il servizio è interessato solamente all'estrapolazione dei dati secondo dei vincoli imposti in fase di query. Sulla base di queste informazioni e delle limitazioni presenti nel sistema di interrogazione del database è stato scelto di utilizzare un approccio basato su chiave primaria complessa in quanto, una chiave primaria semplice composta da singolo attributo non sarebbe stata sufficiente a identificare univocamente gli oggetti all'interno del database. La chiave primaria complessa si divide quindi in partition key e sort key. La sua struttura è descritta nei paragrafi successivi.

ID e proprietà dell'acqua come partition key

La partition key è il campo che identifica la partizione in cui i dati vengono salvati e fa parte della struttura della chiave primaria che identifica gli oggetti all'interno del database. Considerando che stiamo utilizzando una chiave primaria complessa sappiamo che è possibile avere all'interno della stessa tabella due o più oggetti con la stessa partition key. Nel nostro caso si è deciso di utilizzare una concatenazione di due attributi come partition key, l'attributo che identifica il sensore (ID) e l'attributo che identifica la proprietà dell'acqua misurata (proprietà). Concatenando i due attributi si ottiene una partition key strutturata come segue:

$$\text{Partition Key} = \text{"ID\_Sensore"} + \text{"\_"} + \text{"proprietà\_dell'acqua"}$$

In questo modo le operazioni di selezione della partition key in fase di query ci permetteranno di selezionare in modo preciso una determinata proprietà rilevata da un determinato sensore.

Timestamp come sort key

La sort key è la parte della chiave primaria complessa che viene utilizzata per ordinare gli elementi all'interno della tabella e assieme alla partition key permette di identificare univocamente qualsiasi oggetto all'interno della tabella. Quando un sensore rileva il valore di una proprietà, assieme ad esso viene anche salvato il

timestamp, ovvero la data e l'ora in cui è avvenuto l'evento. Sfruttando questa caratteristica è possibile utilizzare il timestamp come sort key per il nostro database. Così facendo saremo in grado di distinguere tutte le possibili misurazioni all'interno del database in quanto ogni singola misurazione verrà identificata dalla partition key, con una struttura tipo:

$$\text{Chiave primaria} = \text{Partition Key} + \text{Sort Key} = \text{"ID sensore"} + \text{"\_"} + \text{"proprietà"} + \text{"\_"} + \text{"timestamp"}$$

Essendo utilizzato nel campo sort key, il timestamp potrà essere manipolato attraverso gli operatori descritti nel paragrafo "Sistema di interrogazione del database". In questo modo si potranno manipolare i dati anche in base alla data e all'ora in cui sono stati raccolti.

Struttura elementi salvati in DynamoDB

Per dare l'idea di come saranno salvati i dati all'interno della tabella viene mostrato uno schema concettuale. In caso fosse necessario è ovviamente possibile memorizzare altre informazioni oltre a quelle visualizzate in figura, in questo caso viene mostrato solamente il campo valore.

ID_proprietà (PK)	timestamp (SK)	Valore
1_temperatura	10/06/2022_18:30:00	12.3 °C
1_temperatura	10/06/2022_18:40:00	12.2 °C
2_ossigeno	10/06/2022_18:30:00	10 %

Figura 3.1: Rappresentazione tabellare di un database chiave-valore.

Capitolo 4

Implementazione

In questo capitolo andremo a descrivere il processo di implementazione del database progettato nel capitolo precedente. Vedremo che tale operazione è possibile attraverso due diversi approcci e andremo a descrivere le operazioni necessarie per eseguire correttamente entrambi. Il capitolo sarà diviso in sezioni ordinate in modo tale che al termine della lettura sia possibile avere gli strumenti necessari per far funzionare correttamente l'ambiente di lavoro. Verranno descritti l'approccio che utilizza il pannello di controllo del servizio messo a disposizione da AWS per l'utilizzo del database e tratteremo anche l'utilizzo delle Software Development Kit (SDK) e delle Application Programming Interface (API) per lo sviluppo di eventuali software destinati alla gestione del database. Le SDK sono fornite gratuitamente da AWS attraverso il pacchetto denominato boto3 per diversi linguaggi di programmazione. Nello specifico il codice che sarà mostrato utilizzerà le SDK per il linguaggio di programmazione Python.

4.1 Configurazione credenziali

Per poter utilizzare il servizio DynamoDB è necessario possedere un account AWS. Per soddisfare questa esigenza ci basta effettuare la registrazione sul sito attraverso una mail valida e dopo aver completato la registrazione sarà possibile usufruire dei vari servizi offerti e iniziare a sviluppare le applicazioni. Per poter utilizzare i servizi è consigliato di effettuare un passaggio successivo relativo alla gestione delle credenziali, sia per l'utilizzo delle SDK che per l'utilizzo tramite pannello di controllo.

Creazione di un profilo IAM

L'AWS Identity and Access Management (IAM) è un servizio che permette di creare e gestire gli utenti autenticati e autorizzati all'utilizzo delle risorse e dei servizi AWS. Quando effettuiamo la prima registrazione di un account AWS attraverso l'utilizzo di una mail stiamo creando il cosiddetto *root user* ovvero l'utente principale che ha la possibilità di utilizzare tutti i servizi e le risorse messe a disposizione senza alcuna limitazione, nonché accedere alle informazioni relative alla fatturazione dei pagamenti e ai metodi di pagamento stessi. Utilizzare le credenziali di accesso del root user per lo sviluppo e l'utilizzo di applicazioni potrebbe comportare dei rischi come ad esempio un furto di identità o una clonazione dei numeri della carta di credito e quindi, per evitare questi problemi il servizio IAM permette di creare degli utenti secondari stabilendo quali siano le operazioni che tali utenti possono eseguire,

andando quindi a limitare le risorse e i servizi ai quali possono accedere [18]. Per creare un utente IAM è necessario accedere all'interfaccia web tramite root user, successivamente, attraverso la barra di ricerca situata in alto, inseriamo la parola chiave 'IAM' per iniziare ad usare il servizio.

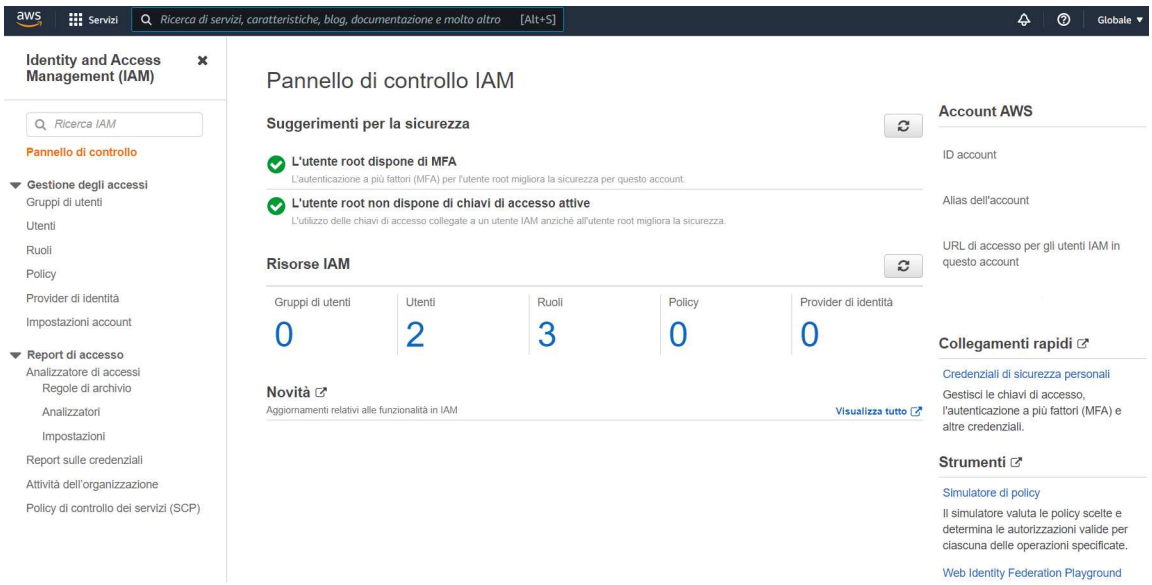

Figura 4.1: Pannello di controllo IAM.

Dal pannello di controllo mostrato in Figura 4.1, navigando nel menu a sinistra sotto la voce 'Gestione degli accessi', selezioniamo la sezione 'Utenti'. Attraverso questa pagina sarà possibile gestire gli utenti che abbiamo creato e crearne di nuovi seguendo una serie di operazioni guidate nelle quali ci verrà chiesto di scegliere un nome per l'utente, il tipo di accesso, che si divide in un accesso tramite linguaggio di programmazione e quindi destinato all'utilizzo delle SDK e API ed un accesso diretto al pannello di controllo AWS, ed infine di stabilire quali siano i servizi che tale utente sarà abilitato ad utilizzare. A seguito della creazione di un utente, in base al tipo di accesso selezionato, saranno fornite delle credenziali che dovranno essere utilizzate per accedere al pannello di controllo dei servizi o per collegare il nostro software al servizio desiderato.

Accesso tramite utente IAM da pagina web

Come per una normale procedura di login, per accedere attraverso un utente IAM creato dal root user è necessario conoscere l'ID dell'account root user, il nome dell'utente IAM con cui si esegue l'accesso e la password corrispondente, generata al momento della creazione dell'utente. Una volta effettuato correttamente tale operazioni avremo accesso ai servizi per i quali l'utente in questione è stato abilitato.

Accesso tramite linguaggio di programmazione

A differenza di un accesso diretto effettuato tramite una pagina di login con l'utilizzo di credenziali, l'approccio tramite linguaggio di programmazione richiede un'operazione di configurazione aggiuntiva. Una volta completata la procedura di creazione di un utente IAM vengono forniti un ID chiave di accesso e una chiave di accesso segreta che dovranno essere utilizzate per collegare il software al servizio AWS di cui l'utente ha necessità [19]. Il collegamento ai servizi AWS attraverso

la programmazione può essere effettuato in diversi modi, per avere una spiegazione dettagliata su tutti i metodi disponibili si consiglia di consultare le guide [19]. In questa sezione verranno invece trattati due metodi per compiere questa operazione, uno fortemente sconsigliato e l'altro invece più indicato per un programmatore. I metodi che andremo a visionare sono:

- passaggio delle credenziali come parametri del metodo `client`;
- utilizzo un file per condividere le credenziali.

Nel primo caso le credenziali vengono fornite attraverso un approccio *hard coded*, le nostre chiavi di accesso vengono direttamente scritte all'interno del metodo risultando dunque visibili nel caso in cui qualcuno riesca ad accedere al codice sorgente. L'esempio mostrato di seguito non contiene credenziali hard coded ed è consigliato di evitare questo tipo di approccio nella maggior parte dei casi.

```
1 import boto3
2 client = boto3.client(
3    'dynamodb',
4    aws_access_key_id = ACCESS_KEY,
5    aws_secret_access_key = SECRET_KEY,
6    region_name = YOUR_REGION_NAME)
```

Codice 4.2: Creazione di una variabile client.

Attraverso il Codice 4.2 è possibile inizializzare un'istanza client collegata al servizio che verrà da noi utilizzato. I parametri che vengono forniti alla funzione sono la chiave di accesso attraverso il campo `aws_access_key`, la chiave segreta tramite `aws_secret_access_key` e la regione in cui la nostra tabella è ospitata tramite `region_name` il quale valore è ottenibile consultando la pagina principale di AWS dopo aver effettuato il login.

L'approccio hard coded risulta sicuramente essere il più semplice tra quelli disponibili in quanto attraverso una semplice funzione riusciamo a instaurare un collegamento con il servizio scelto per usufruire delle sue funzionalità tuttavia, inserire dati sensibili all'interno del codice risulta essere una soluzione poco affidabile nella maggioranza dei casi infatti, se un soggetto estraneo riuscisse ad avere accesso al codice sorgente del software entrerebbe in possesso molto facilmente delle credenziali utilizzate e questa è sicuramente una situazione che si vuole evitare. Un altro metodo, più sicuro di quello appena descritto, prevede di utilizzare un file condiviso al cui interno salvare le credenziali degli utenti da noi scelti. Per poter utilizzare correttamente questo metodo è necessario effettuare alcune operazioni preliminari che andremo brevemente a descrivere in seguito. Prima di tutto è necessario installare AWS Command Line Interface per poter utilizzare i comandi AWS attraverso il prompt dei comandi, e successivamente iniziare il processo di prima configurazione attraverso il comando `aws configure` mediante il quale verranno creati i file `credentials.ini` e `config`. Una volta completata questa prima operazioni all'interno del file `credentials` sarà possbile inserire e salvare le chiavi di accesso e le chiavi segrete degli utenti scelti identificando queste ultime attraverso un nome. È importante sapere che questi valori, nome utente, chiave di accesso e chiave segreta, sono gli unici valori ammessi all'interno di questo file. Un esempio della struttura del file è mostrato di seguito.

Dopo aver eseguito correttamente le operazioni precedentemente descritte andremo ad utilizzare il metodo `Session()` per creare una sessione, intesa come un login,

```
[default]
aws_access_key_id = ACCESS_KEY
aws_secret_access_key = SECRET_KEY

[sviluppatore]
aws_access_key_id = ACCESS_KEY_2
aws_secret_access_key = SECRET_KEY_2

[cliente]
aws_access_key_id = ACCESS_KEY_3
aws_secret_access_key = SECRET_KEY_3
```

Codice 4.3: Credenziali di alcuni utenti salvate nel file `credentials`

attraverso le credenziali dell'utente specificato all'interno del metodo mediante il campo `profile_name` e successivamente sarà possibile inizializzare le istanze client e/o resource, a seconda delle operazioni che decideremo di eseguire, collegandoci al servizio con il quale siamo interessati a lavorare.

```
import boto3

session = boto3.Session(profile_name='sviluppatore')
dynamodb_client = session.client('dynamodb')
```

Codice 4.4: Utilizzo del metodo `Session`.

L'approccio consigliato è quello di utilizzare il file condiviso con le credenziali. In questo modo evitiamo di inserire i nostri dati sensibili all'interno del codice sorgente e attraverso l'utilizzo di un singolo file è possibile mantenere memoria di tutte le credenziali degli utenti e tale file è sufficiente per lavorare con tutte le SDKs fornite da AWS.

4.2 Creazione della tabella

Per immagazzinare i dati all'interno del database c'è sicuramente bisogno di creare la tabella destinata ad ospitare le informazioni rilevate. Per effettuare tale operazione abbiamo a disposizione due approcci che anche in questo caso sono tramite il pannello di controllo AWS del servizio DynamoDB oppure tramite linguaggio di programmazione con l'utilizzo dei metodi messi a disposizione dalle SDK.

Pannello di controllo

La possibilità di creare una tabella attraverso il pannello di controllo del servizio è sicuramente l'approccio più user-friendly. Per iniziare effettuiamo il login all'interno di AWS e tramite la barra di ricerca posta in alto selezioniamo il servizio DynamoDB. Dopo questo passaggio ci troviamo di fronte al pannello di controllo del servizio attraverso il quale è possibile monitorare la situazione delle nostre tabelle. Per iniziare il processo di creazione clicchiamo sopra il pulsante con la scritta 'Crea tabella' a destra della pagina, e saremo reindirizzati ad una sezione che prevede una serie di textbox da compilare in cui vengono richieste le caratteristiche della tabella quali nome, partition key ed eventuale sort key.

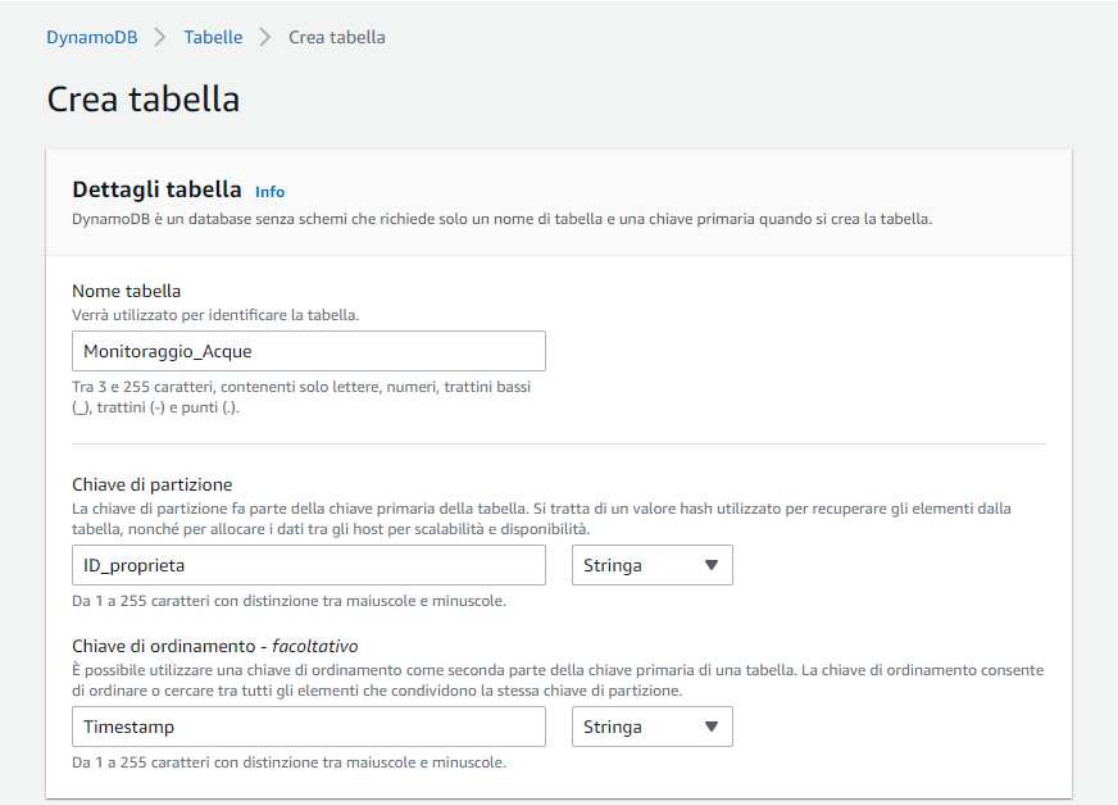

Figura 4.5: Creazione di una tabella tramite console di comando.

Dopo aver compilato i campi con le informazioni richieste ed aver confermato la creazione della nostra tabella saremo reindirizzati in una pagina nella quale verrà mostrata l'effettiva creazione della nostra tabella.

Linguaggio di programmazione

Per procedere alla creazione della tabella attraverso il linguaggio di programmazione Python abbiamo bisogno di utilizzare il metodo `create_table()` specificando alcuni parametri che andremo a vedere in seguito. Prima di effettuare questa operazione tuttavia abbiamo bisogno di utilizzare il metodo `resource`, il quale funzionamento è uguale a quello del metodo `client`, e che ci permette di istanziare una variabile che si colleghi al servizio e ci consenta di lavorare con il database stesso.

Dal Codice 4.6 si vede come i parametri passati alla funzione siano:

- TableName, che corrisponde al nome che si vuole assegnare alla tabella;
- KeySchema, che contiene informazioni relative alla chiave primaria. Nel campo AttributeName si indica il nome delle chiavi mente nel campo KeyType viene specificato quale chiave è destinata ad essere partition key, tipo HASH, o sort key, tipo RANGE;
- AttributeDefinitions, nel quale si indicano attraverso il campo AttributeType il tipo di dato delle chiavi. Le possibilità sono 'S' di stringa, 'N' per indicare un numero intero e 'B' se la chiave ha valore binario;
- ProvisionedThroughput, dove vengono indicate le capacità in lettura e scrittura del database ovvero quante righe della tabella è possibile leggere o scrivere in un solo secondo. Il valore predefinito è impostato a 5.

```
dynamodb_resource = boto3.resource('dynamodb')

def create_table():
   try:
      dynamodb_table = dynamodb_resource.create_table(
         TableName = 'Monitoraggio_Acque',
         KeySchema = [
                     {
               'AttributeName': 'ID_property',
               'KeyType': 'HASH'
            }
                     {
               'AttributeName' : 'Timestamp',
               'KeyType' : 'RANGE'
            }
         ],
         AttributeDefinitions = [
                     {
               'AttributeName': 'ID_property',
               'AttributeType': 'S'
            },
            {
               'AttributeName' : 'Timestamp',
               'AttributeType' : 'S'
            }
         ],
         ProvisionedThroughput = {
            'ReadCapacityUnits': 5,
            'WriteCapacityUnits': 5
         })
      dynamodb_table.wait_until_exists()
      return dynamodb_table
   except dynamodb_client.exceptions.ResourceInUseException as err:
      dynamodb_table = dynamodb_resource.Table('Monitoraggio_Acque')
      return dynamodb_table
```

Codice 4.6: Creazione di una tabella.

Attraverso questo codice specifico per il nostro caso abbiamo considerato anche che la tabella destinata ad essere creata non fosse già presente tra quelle disponibili andando a gestire l'eccezione `ResourceInUseException`. Abbiamo quindi descritto quali sono le procedure principali per creare una tabella del servizio DynamoDB. Specifichiamo che i parametri mostrati nel Codice 4.6 non sono tutti ma sono quelli strettamente necessari per effettuare correttamente l'operazione di creazione di una tabella tramite il metodo `create_table`. Per avere una descrizione completa del metodo si consiglia di consultare le documentazioni presenti online [19].

4.3 Inserimento dei dati

Una volta creata la tabella nella quale andremo a memorizzare i dati possiamo descrivere il processo di inserimento di quest'ultimi. Anche in questo caso l'approccio

può essere direttamente effettuato dal pannello di controllo del servizio o tramite programmazione mediante l'utilizzo delle SDK.

Pannello di controllo

L'inserimento dei dati tramite pannello di controllo è un processo che richiede un impiego di tempo considerevole, soprattutto nel caso in cui la quantità di dati da inserire sia elevata. Questo deriva dal fatto che al momento il pannello di controllo del servizio DynamoDB non offre la possibilità di importare dati da file di tipo JSON, CSV o simili e quindi il processo di inserimento deve essere effettuato singolarmente e a mano per ciascun dato. Per iniziare è necessario accedere al pannello di controllo dal quale è possibile recarsi alla sezione 'Esplora elementi'. Da qui, selezionando la tabella nella quale vogliamo inserire i nostri dati, clicchiamo il bottone 'Crea voce' per essere reindirizzati al modulo per inserire i dati.

Figura 4.7: Pagina di inserimento dei dati

In questa pagina abbiamo la possibilità di inserire i dati attraverso un modulo di compilazione, in cui inizialmente sono presenti solo i campi della partiton key e della sort key. Nel caso in cui l'utente sia interessato ad inserire altri dati è necessario aggiungere un nuovo attributo mediante il pulsante dedicato, scegliendo il tipo di dato e un nome che identifichi l'attributo all'interno della tabella. È inoltre possibile registrare i dati effettuando la stesura di un file JSON ed anche in questo caso è necessario scrivere i nuovi attributi all'interno della struttura. Una volta completata la compilazione del modulo o la stesura del file JSON l'operazione va confermata e in seguito ad un controllo sulle variabili effettuato in automatico il dato verrà inserito all'interno del database.

Linguaggio di programmazione

La possibilità di creare un algoritmo in grado di eseguire proceduralmente l'inserimento dei dati all'interno del database favorisce l'utilizzo della programmazione. Le librerie mettono a disposizione il metodo `put_item` attraverso il quale è possibile effettuare operazioni di inserimento all'interno della tabella. Nel nostro ambiente di sviluppo inoltre consideriamo che i dati registrati dai sensori vengano formattati in file JSON per poter accedere facilmente ai loro valori. L'utilizzo di questo approccio facilita notevolmente l'inserimento dei dati, permettendoci così di risparmiare tempo. La struttura di tale file è stata progettata per favorire l'inserimento e l'estrapolazione dei dati ma non è sicuramente l'unica opzione disponibile. Nel nostro caso abbiamo deciso di salvare alcune caratteristiche principali dei sensori all'inizio della struttura del file, identificandole attraverso un sistema chiave-valore. I dati raccolti sono invece salvati all'interno dell'array di array `data_collected` in cui

```
data = {
   'sensor_ID' : int,
   'latitude' : string,
   'longitude' : string,
   'battery_status' : int,
   'data_collected' :
      {
         'property' : [
            {
               'timestamp' : string,
               'value' : decimal
            }
         ]
      }
   }
```

Codice 4.8: Struttura file JSON.

ad ogni proprietà che il sensore può misurare corrisponde un array dove all'interno vengono memorizzate le misurazioni effettuate attraverso il timestamp e il valore rilevato.

Il Codice 4.8 mostrato di seguito è un esempio di come sia possibile inserire dati all'interno del database progettato per il nostro caso di studio attraverso un algoritmo dedicato. I parametri che vengono forniti al metodo in questo caso consistono solamente nell'oggetto da inserire formattato secondo le caratteristiche richieste. Per una spiegazione dettagliata sul funzionamento del metodo `put_item` si consiglia di visionare le documentazioni dedicate [19].

```
def insert_items(sensor):
   for property, value in sensor['data_collected'].items():
      partition_key = str(sensor['sensor_ID']) + '_' + property
      for element in value:
         sort_key = element['timestamp']
         dynamodb_table.put_item(
            Item = {
               'ID_property' : partition_key,
               'Timestamp' : sort_key,
               'Measure' : element['measure'],
               'Latitude' : sensor['latitude'],
               'Longitude' : sensor['longitude']
               })
```

Codice 4.9: Inserimento di dati in una tabella.

4.4 Interrogazione del database

Interrogare il database significa inviare una richiesta di erogazione dei dati sulla base di determinate caratteristiche che il dato restituito deve avere. Durante la progettazione del database nel Capitolo 3 abbiamo evidenziato come il servizio DynamoDB presenti alcuni vincoli nel momento in cui si va ad eseguire un operazione di query al suo interno come il vincolo di poter specificare solamente una partition key per query e di avere operazioni e comparatori limitati sugli attributi dei dati.

Come per i casi precedenti possiamo scegliere di utilizzare direttamente il pannello di controllo del servizio DynamoDB oppure utilizzare le SDK che anche in questo caso è la scelta più consigliata.

Pannello di controllo

L'interrogazione del database attraverso il pannello di controllo è un'operazione abbastanza intuitiva e semplice da eseguire. Mediante la compilazione di un modulo nel quale è richiesto di specificare i vincoli che i dati devono soddisfare, possiamo effettuare operazioni di query all'interno della tabella o scansionare l'intera tabella. La differenza tra le due operazioni sta nel fatto che l'operazione di scansione restituisce tutte le righe presenti all'interno della tabella mentre l'operazione di query restituisce le righe che soddisfano una determinata uguaglianza rispetto alla partition key e facoltativamente soddisfi dei vincoli rispetto alla sort key. Per effettuare le operazioni di scansione o query all'interno della tabella è necessario recarsi alla sezione 'Esplora elementi' dal pannello di controllo del servizio, selezionare la tabella dalla quale si vogliono estrapolare i dati e compilare il modulo a seconda delle esigenze.

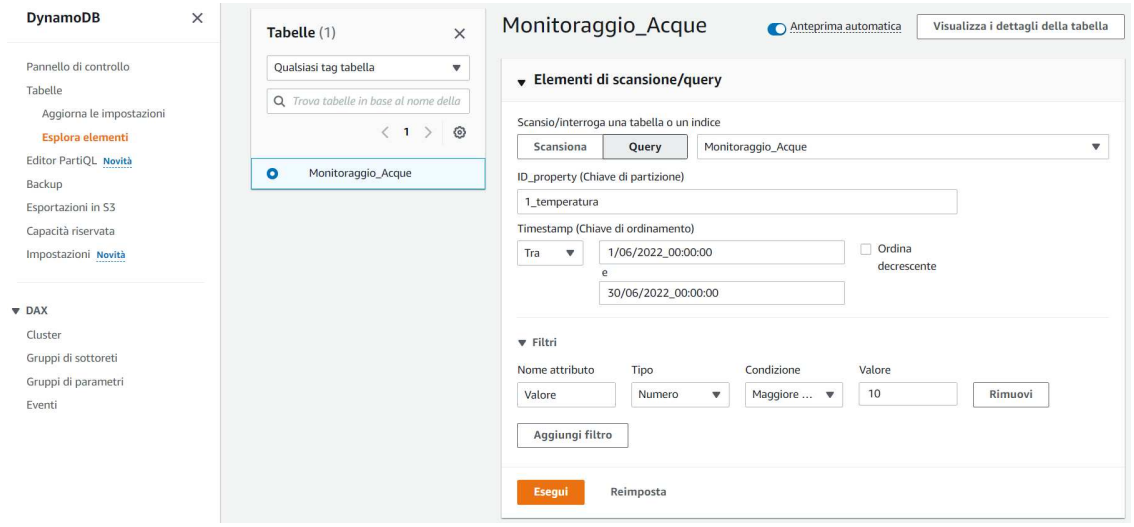

Figura 4.10: Interrogazione del database attraverso pannello di controllo

Nelle operazioni di interrogazione attraverso il pannello di controllo, sia per le operazioni di scansione che per quelle di query è possibile filtrare i risultati andando a imporre delle condizioni sugli attributi delle righe della tabella. Le operazioni eseguibili sono quelle descritte nel Capitolo 3 ed è possibile effettuare una sola operazione per attributo. Una volta eseguita la query o la scansione della tabella sarà possibile visualizzare le voci restituite e nel caso l'utente fosse interessato a scaricare tali informazioni ci basterà selezionare dal menu a tendina l'opzione 'Scarica i risultati in CSV' per ottenere un file formattato con all'interno le voci restituite dalla query eseguita.

Linguaggio di programmazione

Come per l'inserimento dei dati all'interno della tabella anche in questo caso la possibilità di creare un algoritmo che elabori le richieste di erogazione dei dati in modo sequenziale ci permette di effettuare interrogazioni multiple risparmiando tempo rispetto all'utilizzo della console di comando, nella quale sarebbe necessario

specificare per ogni query una partition key. I metodi per eseguire le richieste dei dati dalle tabelle sono `query` per eseguire appunto le operazioni di query e `scan` per eseguire la scansione di una tabella. È consigliabile utilizzare il metodo `query` poiché attraverso esso riduciamo notevolmente le operazioni di lettura all'interno della tabella infatti, il metodo `scan` restituisce tutte le righe della tabella che si traduce in molte operazioni di lettura che potrebbero diventare dispendiose dal punto di vista economico.

```
def general_query(sensors, properties):
   for id in sensors:
      for property in properties:
         PK = str(id) + '_' + str(property)
         response = dynamo_db_table.query(
            KeyConditionExpression = Key('ID_property').eq(PK))
         results.append(response['Items'])
   return results
```

Codice 4.11: Query tramite partition key.

Il Codice 4.11 è stato progettato in modo da eseguire anche più di una query tramite singolo comando. Attraverso una semplice interfaccia grafica è possibile andare a selezione i sensori e le proprietà di cui l'utente vuole conoscere i valori e tramite un semplice algoritmo andare a creare la partition key specifica in grado di soddisfare l'uguaglianza richiesta.

```
def general_query(sensors, properties, start_date, end_date):
   for id in sensors:
      for property in properties:
         PK = str(id) + '_' + str(property)
         response = dynamo_db_table.query(
            KeyConditionExpression = Key('ID_property').eq(PK)
               & Key('Timestamp').between(start_date, end_date))
         results.append(response['Items'])
   return results
```

Codice 4.12: Query tramite partition key e sort key.

Il Codice 4.12 invece è un esempio di come sia possibile utilizzare la sort key per filtrare ulteriormente i risultati delle query. In questo caso specifico utilizziamo l'operatore `between` per specificare che la data di raccolta dei dati deve essere compresa tra due valori specifici. Anche in questo caso, attraverso un algoritmo dedicato, potrebbe essere possibile scaricare i dati restituiti dalle query all'interno di un file CSV o JSON che poi verrà utilizzato dal personale qualificato per effettuare le analisi dei dati.

Capitolo 5

Conclusioni e sviluppi futuri

In conclusione è possibile affermare che lo svolgimento di questa tesi ha rispettato la richiesta di progettare un database dedicato all'archiviazione dei dati rilevati da sensori posti in ambiente acqueo. Inoltre, la possibilità di sviluppare un prototipo usufruendo gratuitamente del servizio DynamoDB ha permesso, attraverso il Capitolo 4, di descrivere le due diverse procedure utili a implementare e utilizzare il database. L'implementazione è inoltre servita a dimostrare che le scelte effettuate nella fase di progettazione sulla chiave primaria, composta da partition key e sort key, consentono di identificare univocamente tutti i dati all'interno del database senza creare doppioni e permettono di interrogare il database in modo corretto rispettando le limitazioni evidenziate nel Capitolo 3. Tale capitolo e il Capitolo 4 possono essere ripresi per ottenere le informazioni essenziali a conoscere e poter usufruire del servizio DynamoDB. Nella tesi non sono state effettuate stime relative al costo di utilizzo del servizio in quanto il progetto sull'uso dei sensori per il monitoraggio delle acque è ancora in fase di studio e le informazioni necessarie a tale scopo non sono ancora disponibili.

La tesi si è concentrata sulla parte destinata a memorizzare le informazioni rilevate dai sensori ma aspetti come ad esempio la trasmissione dei dati non sono stati trattati. La scelta di utilizzare un servizio fornito da AWS è stata dettata anche dall'eventuale possibilità di sviluppare altri aspetti del progetto attraverso l'utilizzo dei servizi forniti dall'azienda. Di seguito introdurremo brevemente alcune soluzioni che potrebbero essere utili a semplificare e favorire lo sviluppo del progetto.

5.1 AWS IoT

AWS IoT è una raccolta di servizi utili allo sviluppo di soluzioni per l'Internet of Things (IoT) destinate all'industria, alla domotica e alla vita comune [20]. Attraverso i servizi messi a disposizione è possibile partire dallo sviluppo dei dispositivi ai quali collegare i sensori. AWS infatti mette a disposizione un sistema operativo per microcontrollori e microprocessori chiamato FreeRTOS che opera in tempo reale (Real Time Operating System) e semplifica la connessione di questi. Anche la sicurezza e la gestione dei dispositivi sono sicuramente aspetti da considerare e al fine di poter soddisfare anche questa necessità AWS fornisce i servizi Device Defender e Device Management. Tra i punti chiave di una soluzione IoT troviamo la comunicazione dei dispositivi. Anche in questo caso utilizzando il servizio offerto da AWS, IoT Core, è possibile far comunicare in modo sicuro e collegare facilmente i dispositivi tra di loro o ad altri servizi AWS. Una volta raccolta una quantità di dati esaustiva attraverso i servizi che offrono sistemi di archiviazione è possibile effet-

tuarne un analisi, anche in tempo reale, attraverso algoritmi o tecniche di machine learning applicabili tramite l'utilizzo di servizi destinati a tale scopo. La scelta di quali servizi utilizzare rimane un compito riservato agli sviluppatori e resta il fatto che non sia indispensabile sviluppare un intero progetto attraverso AWS ma l'utilizzo di questi risulta essere un'opzione valida per rendere più semplice soddisfare i processi e le meccaniche che un progetto IoT deve rispettare. Tra quelli elencati il servizio IoT Core copre un ruolo rilevante in quanto permette uno sviluppo assistito della comunicazione tra i dispositivi e i servizi, processo non banale che se sviluppato in modo errato o con errori potrebbe causare problemi di funzionamento.

AWS IoT Core

AWS IoT Core è un servizio di controllo offerto da AWS che permette di connettere i dispositivi della rete ai servizi AWS o ad altri dispositivi svolgendo la funzione di mediatore dei messaggi e di gateway di dispositivi. Quando dei dispositivi IoT vengono connessi alla rete si decide anche il protocollo che essi andranno ad utilizzare per le comunicazioni. AWS IoT supporta una serie di protocolli attraverso i quali far comunicare i dispositivi della rete, tali protocolli sono:

- Message Queue Telemetry Transport (MQTT);
- MQTT tramite Web Socket Services (MQTT via WSS);
- HyperText Transfer Protocol - Secure (HTTPS);
- Long Range Wide Area Network (LoRaWAN).

Analizzando la documentazione inerente a questo argomento viene specificato come la capacità del servizio di svolgere il ruolo di mediatore (o broker) sia collegata al protocollo MQTT ed alle sue funzionalità publish and subscribe. Nel caso invece si decida di utilizzare il protocollo LoRaWAN, il servizio mette a disposizione la possibilità di creare una serie di regole attraverso le quali attivare altri servizi a seguito di determinati eventi. Nelle documentazioni [20] vengono specificate quali sono le possibili regole invocabili e tra esse è presente anche la regola 'Inserisci i dati raccolti da un dispositivo IoT all'interno di un database DynamoDB'. È quindi possibile, attraverso l'utilizzo del sistema di regole di IoT Core e di DynamoDB semplificare i passaggi per il salvataggio dei dati evitando di creare applicativi su misura per svolgere questa funzione, ma utilizzando la procedura guidata che anche in questo caso è resa disponibile attraverso un pannello di controllo. IoT Core svolge quindi una serie di ruoli centrali all'interno della realtà di sviluppo che potrebbero essere complicati e longevi se progettati dalla base. Utilizzare il servizio farebbe risparmiare quindi tempo e risorse ai programmatori che avrebbero eventualmente tempo di concentrarsi su altri aspetti del progetto in modo da fornire un prodotto finale completamente funzionante e facile da utilizzare.

Bibliografia

[1] "Monitoraggio e qualità acque." https://www.isprambiente.gov.it/it.

[2] "Treccani.it - vocabolario treccani on line," 2011. https://www.treccani.it/vocabolario/sensore/.

[3] "Eurostat, ufficio statistico dell'Unione europea." https://ec.europa.eu/eurostat/.

[4] "Webtribunal, indipendent review site." https://webtribunal.net/.

[5] A. Albano, G. Ghelli, and R. Orsini, *Fondamenti di basi di dati*, vol. 2. Zanichelli, 2005.

[6] "Gravity: Analog Electrical Conductivity Sensor DFR0300." https://www.dfrobot.com/.

[7] "Waterproof DS18B20 Digital Temperature Sensor for Arduino." https://www.robot-italy.com/it/.

[8] "Gravity: Analog Dissolved Oxygen Sensor SEN0237-A / Meter Kit for A." https://www.dfrobot.com/.

[9] "Sensore di pressione BPS110-AG01P0-2DG." https://www.digikey.it/.

[10] "Gravity: Analog pH Sensor SEN0161 / Meter Pro Kit for Arduino." https://www.dfrobot.com/.

[11] "CQRobot Ocean: TDS (Total Dissolved Solids) Meter Sensor CQRSENTDS01." https://www.amazon.it/CQRobot-Ocean-Compatible-Scientific-Laboratory/.

[12] E. F. Codd, "A relational model of data for large shared data banks," in *Software pioneers*, pp. 263–294, Springer, 2002.

[13] C. Strauch, U.-L. S. Sites, and W. Kriha, "Nosql databases," *Lecture Notes, Stuttgart Media University*, vol. 20, p. 24, 2011.

[14] "Understanding The Different Types of NoSQL Databases." https://www.mongodb.com/types-of-nosql-databases.

[15] "Cos'è il cloud computing." https://aws.amazon.com/it/what-is-cloud-computing/.

[16] "Tipi di cloud computing." https://aws.amazon.com/it/types-of-cloud-computing/.

[17] “What Is Amazon DynamoDB?.”
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/.

[18] “What Is Amazon IAM?.”
https://docs.aws.amazon.com/IAM/latest/UserGuide/.

[19] “Boto3 Documentation.”
https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

[20] “What is AWS IoT?.”
hhttps://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.