



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Curriculum Software Engineering and IT Management

Realizzazione di una RESTful API per l'esposizione di servizi di Quantum Programming

RELATORE

Prof. Andrea De Lucia

CO-RELATORE

Dott. Manuel De Stefano

CANDIDATO

Salvatore Amideo

Matricola: 0522500964

Anno Accademico 2021-2022

"La mente è come un paracadute, funziona solo se si apre"

Abstract

Negli ultimi anni, i ricercatori stanno investendo sempre più risorse per comprendere in che misura l'informatica quantistica possa migliorare le prestazioni degli algoritmi classici di apprendimento automatico. L'obbiettivo che si vuole raggiungere, è una più efficiente esecuzione di questi algoritmi ad alto costo computazionale utilizzando l'informatica quantistica, traducendo la logica classica originale all'interno della teoria quantistica. Poiché il volume globale dei dati raccolti aumenta continuamente e le architetture degli algoritmi diventano sempre più sofisticate, cresce anche la necessità di trovare approcci innovativi per ottenere prestazioni migliori. A questa premessa si collega il presente lavoro, incentrato sullo sviluppo di un'applicazione cloud che riesca a portare la programmazione quantistica a un livello più semplice e accessibile anche a soggetti che non conoscono perfettamente la materia. A tal fine si è sviluppato un sistema che permetterà allo sviluppatore di caricare e utilizzare il proprio algoritmo quantistico rendendolo però anche disponibile ad altri sviluppatori o utilizzatori. Inoltre saranno già precaricati anche gli algoritmi di programmazione quantistica più noti.

Le conoscenze di base per la comprensione di questo lavoro sono esplicitate nei primi due capitoli che trattano i fondamenti della meccanica quantistica, limitando lo studio solo all'area di interesse ma definendo comunque un quadro matematico generale. A seguire, una panoramica di dispositivi e applicazioni attuali e previsti a breve termine. Nel terzo capitolo, la progettazione del sistema; nel quarto, un esempio di utilizzo reale; e nel quinto, per concludere l'analisi dei risultati ottenuti e le prospettive future di sviluppo.

Indice

Indice	ii
Elenco delle figure	v
Elenco delle tabelle	vii
1 Introduzione	1
1.1 Contesto applicativo	1
1.2 Motivazioni e obiettivi	2
1.3 Risultati ottenuti	3
1.4 Struttura della tesi	4
2 Background e Stato dell'arte	5
2.1 L'era dei quanti	5
2.2 I Fondamenti della computazione quantistica	6
2.2.1 I circuiti quantistici	6
2.2.2 Sovrapposizione di Stati	8
2.2.3 Entanglement	9
2.3 Peculiarità e problematiche del computer quantistico	10
2.4 Informatica quantistica: dall'hardware al software	11
2.4.1 Software quantistico: Qiskit e Cirq	12
2.4.2 Qiskit software	13
2.4.3 Cirq software	13

2.4.4	Azure Quantum	15
2.4.5	Amazon Braket	16
2.4.6	Zapata Computing	17
2.4.7	QuantumPath	17
2.5	Primi passi nell'ingegneria del software quantistico	18
2.5.1	Reingegnerizzazione quantistica del software	21
2.5.2	Quantum service-oriented computing	23
3	Progettazione e implementazione infrastruttura server-side	32
3.1	Architettura di sistema proposta	33
3.1.1	Gateway API	33
3.1.2	Orchestrator	34
3.1.3	Meta-modello	34
3.1.4	Provider Quantistico	34
3.2	Architettura di sistema implementata	35
3.2.1	Servizio REST o Backend	35
3.2.2	Implementazione /CustomAPIquantum	40
3.2.3	Implementazione /quantumAPI	41
3.2.4	Implementazione /insertAlgorithm	44
3.2.5	Implementazione /getInfoAlgorithm	45
3.2.6	Database	45
3.2.7	Codice sorgente classe database	47
3.3	Swimline Diagram	48
3.4	Use cases	50
3.4.1	Use case (utilizzo tramite API)	51
3.5	Esempio utilizzo tramite API	51
3.5.1	Il problema di Simons	51
3.5.2	L'algoritmo di Simons	52
3.5.3	Soluzione classica	52
3.5.4	Soluzione quantistica	53
3.5.5	Implementazione algoritmo Simons	53
3.5.6	Implementazione classe Simons	54
3.5.7	Implementazione endpoint /quantumAPI	56
3.5.8	Programma semplice per l'utilizzo dell'algoritmo di Simons	57

4 Progettazione e implementazione applicazione web	59
4.1 Requisiti funzionali	59
4.2 Frontend	60
4.2.1 Progettazione Frontend	60
4.3 Use case (applicazione web)	61
4.4 Inserimento e testing dell'algoritmo	62
4.4.1 Pagina di Benvenuto	62
4.4.2 Pagina di Scelta Algoritmo	63
4.4.3 Inserimento Dati per Esecuzione Algoritmo	63
4.4.4 Risultato Esecuzione Algoritmo	64
4.4.5 Pagina di Caricamento Algoritmo	65
4.4.6 Inserimento Algoritmo	66
4.4.7 Risultato Inserimento Algoritmo	66
5 Conclusioni e sviluppi futuri	68
5.1 Conclusioni	68
5.2 Sviluppi Futuri	70
6 Ringraziamenti	71
Bibliografia	73
A Tecnologie utilizzate	76
A.1 HTML, CSS, JavaScript	76
A.2 Python	77
A.3 Docker	79

Elenco delle figure

2.1 Esempio circuito	7
2.2 Differenza tra Bit e Qubit	8
2.3 La struttura di funzionamento di Qiskit	13
2.4 La struttura di un circuito di Cirq	14
2.5 La struttura di funzionamento di Cirq	15
2.6 La struttura di funzionamento di Azure Quantum	16
2.7 Domande poste nel sondaggio	20
2.8 Processo di reingegnerizzazione quantistica	22
2.9 Architettura quantistica orientata ai servizi	24
2.10 Metodologia di progettazione dell'applicazione QSOA basata su modelli	25
2.11 Versione classica dell'implementazione di un servizio	26
2.12 Algoritmo quantistico avvolto da un servizio classico	28
2.13 Versione ibrida architettura classica-quantistica di un'implementazione di un servizio	30
3.1 Diagramma attività	33
3.2 Architettura di sistema	35
3.3 Modulo Servizio REST	36
3.4 Swimline Diagram	49
3.5 Formula uno-a-uno (Simons)	52
3.6 Formula due-a-uno (Simons)	52
3.7 Formula (Simons)	52

3.8 Circuito implementazione algoritmo Simons	53
4.1 Modulo Frontend	61
4.2 Schermata di Benvenuto	62
4.3 Schermata di Scelta dell'Algoritmo	63
4.4 Schermata di Inserimento Input	64
4.5 Schermata Esito Esecuzione Algoritmo	65
4.6 Schermata Caricamento Algoritmo	65
4.7 Schermata Inserimento Algoritmo	66
4.8 Schermata Risultato Inserimento Algoritmo	67
A.1 I loghi di HTML, JavaScript, CSS	77
A.2 Il logo di Python	77
A.3 Il logo di Docker	79

Elenco delle tabelle

3.1 Tabella Endpoint	39
3.2 Tabella Algorithm	47
4.1 Requisiti Funzionali	59

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Fin da quando l’umanità ha cominciato a interrogarsi sull’essenza di ciò che la circonda, ponendosi quesiti sui fenomeni naturali e sul proprio ruolo nel cosmo, ha avuto bisogno di strumenti in grado di supportarla nel raggiungimento di tale obiettivo. Uno di questi è lo sfruttamento della natura; l’utilizzo delle risorse fisiche come materiali, forze e sorgenti di energia, ha fatto sì che si riuscisse a ottenere progressi tecnologici sempre più importanti.

L’avanzamento scientifico e tecnologico procede, da sempre, in parallelo, a un processo che si ripete ciclicamente: il primo porta alla luce nuove possibilità mentre il secondo le sfrutta.

La più importante arma di conoscenza è il cervello. Non c’è nulla che possa essere paragonato allo strabiliante organo che è alla base di ogni invenzione. In quanto componente umana, però ha un difetto: è suscettibile al trascorrere del tempo. L’utilizzo della tecnologia infatti è principalmente mirata a rendere l’elaborazione dei dati più semplice ed efficiente possibile. Una volta osservati, i dati devono essere elaborati attraverso calcoli e processi che portino a una conclusione ragionevole e coerente con gli esperimenti effettuati e ciò diventa possibile per un umano (o per un gruppo di umani) solo se queste informazioni rimangono al di sotto di una certa soglia di complessità o quantità. Per tale motivo, nel tempo, sono stati sviluppati strumenti di calcolo che supportassero la specie umana nella propria “fame” di risposte. Partendo dagli astrolabi e dai congegni per la stima delle orbite dei pianeti e

delle posizioni degli astri e passando per le prime calcolatrici o macchine programmabili, si è infine arrivati alla costruzione del computer.

Nel corso del Ventesimo secolo, l'informazione è entrata a far parte della tecnologia, a partire dal momento in cui l'invenzione dei computer ha permesso di processare informazioni complesse all'esterno dei cervelli umani. Del resto, la stessa storia dei calcolatori è intessuta di una sequenza di cambiamenti da un tipo di implementazione fisica a un altro.

Dalle prime valvole ai circuiti integrati di oggi, alle odierne tecniche più avanzate che consentono di costruire componenti di dimensioni inferiori al micron; e presto avremo chip ancora più ridotti, sino a raggiungere il punto in cui le porte logiche necessarie al funzionamento dei PC saranno costituite di pochi atomi ciascuna.

Sul piano degli atomi, la materia obbedisce alle regole della meccanica quantistica, molto differenti da quelle della fisica classica, che invece determinano le proprietà delle porte logiche convenzionali. Per questo motivo, se i computer in futuro dovranno diventare di dimensioni sempre più ridotte, una tecnologia completamente nuova dovrà sostituire ciò che utilizziamo ora.

Da questa premessa nasce l'informatica quantistica, ovvero un campo tra fisica, informatica e matematica che sfrutta le capacità di un nuovo tipo di computer basato sulla meccanica quantistica.

I computer quantistici rivoluzionano le basi stesse del modo in cui le macchine eseguono i calcoli. Mentre quelli classici si basano sul concetto di bit, ossia valori logici che possono essere 0 o 1 e sono implementati da dispositivi che esistono in uno dei due stati possibili, i computer quantistici nascono sull'idea di bit quantistici, o qubit, un sistema quantomeccanico a due stati che si trova contemporaneamente, in entrambi. Ciò conduce alla necessità di un nuovo approccio al calcolo e all'elaborazione delle informazioni, con l'aspettativa che, sfruttando le proprietà della meccanica quantistica, sia possibile in compiti molto specifici, ottenere una notevole accelerazione e un aumento delle prestazioni.

È importante sottolineare che i computer quantistici non sono destinati a superare quelli classici in ogni compito. Il loro scopo, infatti, è piuttosto quello di risolvere problemi computazionalmente difficili.

1.2 Motivazioni e obiettivi

Negli ultimi anni l'attenzione per l'informatica quantistica è aumentata notevolmente, sia da parte dei ricercatori che delle aziende tecnologiche, dal momento che può essere un

valore aggiunto specifico in settori industriali come la sanità, i servizi finanziari e le catene di approvvigionamento. Ad esempio, potrebbe contribuire ad aumentare significativamente la velocità e l'efficacia del rilevamento in tempo reale di potenziali frodi e anomalie di transazioni finanziarie su scala nazionale o addirittura globale.

Tuttavia, lo sviluppo di programmi quantistici non è un compito facile. Studi recenti hanno infatti evidenziato come questa nuova pratica di programmazione abbia le sue peculiarità, mettendo in difficoltà gli sviluppatori di applicazioni quantistiche che spesso lottano con problemi relativi all'infrastruttura software, come l'integrazione tra componenti quantistici e tradizionali. Tali problemi potrebbero essere causati da diversi fattori, come i diversi paradigmi e linguaggi di programmazione, la mappatura dell'input tra le parti classiche e quantistiche e l'altissimo vendor lock-in che caratterizza le applicazioni quantistiche. Inoltre, è stato anche dimostrato come le API dei framework quantistici siano in costante cambiamento, rendendo così gli sviluppatori costretti ad aggiornare spesso il loro codice.

Proprio da questi scenari nasce l'obbiettivo principale del presente studio che mira infatti, a sviluppare un sistema che consentirà agli sviluppatori di scrivere i programmi quantistici e integrarli nei loro sistemi senza preoccuparsi di alcun dettaglio tecnico, delle piattaforme e dei provider quantistici. Grazie a questo sistema, gli sviluppatori potranno comporre, configurare e gestire applicazioni senza una conoscenza approfondita dell'infrastruttura quantistica sottostante-cioè ambiente di esecuzione e provider.

L'architettura del sistema proposto dovrà essere formata da quattro componenti principali: il gateway API, l'orchestratore, il traduttore di meta-modelli e il provider quantistico. L'API Gateway, è la componente più semplice che darà la possibilità di comunicare con il sistema tramite API. L'orchestratore avrà invece il compito di orchestrare le richieste che arriveranno all'endpoint e verificare quale macchina quantistica riesca a eseguire l'algoritmo nel modo più efficiente. Prima però di eseguire l'algoritmo, l'orchestratore dovrà interagire con il traduttore di meta-modelli per ottenere una rappresentazione dell'algoritmo compatibile con il provider selezionato. Pertanto, l'algoritmo tradotto, verrà inviato al provider quantistico responsabile dell'esecuzione. Completato il calcolo, l'output risultante sarà restituito all'API Gateway che soddisferà la richiesta.

1.3 Risultati ottenuti

Dal progetto proposto è stata implementata una parte cloud che funge da middleware tra la componente classica e quella quantistica. Il sistema sviluppato infatti, implementa

il gateway API che comunica direttamente con il provider quantistico. Tutto ciò permette l'elaborazione di algoritmi di programmazione quantistica con input personalizzati dall'utente oltre che la possibilità di incrementare il numero di algoritmi quantistici a disposizione, condividendo i propri grazie alla capacità della piattaforma di acquisirli e renderli disponibili agli sviluppatori. Per tale scopo si è utilizzato il modulo di Qiskit per Python che ha reso molto più semplice l'interfacciamento con il sistema quantistico di IBM, mentre, tutta l'architettura Cloud, è stata implementata *ex novo*, così come l'interfaccia grafica. Quest'ultima serve per semplificare il caricamento degli algoritmi e per verificarne la corretta esecuzione. Inoltre sono stati precaricati, all'interno del sistema, gli algoritmi quantistici più famosi come: Algoritmo di Bernstein-Vazirani, Algoritmo di Deutsch-Jozsa e Algoritmo di ricerca di Grover solo per citarne alcuni.

Si è scelto di implementare il sistema utilizzando Qiskit, non solo perché sta divenendo sempre più di ampio utilizzo, ma anche per capacità di utilizzare computer quantistici in modo semplice anche dai non addetti ai lavori.

1.4 Struttura della tesi

Di seguito vengono elencati i capitoli in cui si snoda l'argomentazione della tesi. Per ognuno vengono elencati i temi trattati:

- **Capitolo 2** - in questo capitolo diviso in due sezioni, verrà prima fatta una panoramica sul quantum computing andando ad affrontare l'argomento sia a livello teorico che pratico, poi nella seconda sezione verrà effettuata una panoramica sui lavori e sulle tecnologie esistenti;
- **Capitolo 3** - spiegazione del sistema progettato;
- **Capitolo 4** - un esempio di utilizzo del sistema;
- **Capitolo 5** - nella parte finale si esprimono i punti di forza del Quantum Computing, del sistema sviluppato e i molteplici vantaggi che caratterizzano l'utilizzo e gli eventuali sviluppi futuri.

CAPITOLO 2

Background e Stato dell'arte

Questo capitolo illustra lo stato dell'arte e il lavoro presente in letteratura sugli aspetti di ricerca interessati dal nostro studio.

2.1 L'era dei quanti

Il XXI secolo sarà ricordato come "*l'era dei quanti*" [1]. Questo grazie all'utilizzo dei principi di meccanica quantistica per la risoluzione di problemi NP-completi [2, 3]. Se questo evento dovesse accadere si parlerà di "*supremazia quantistica*" [4], ovvero il momento in cui un dispositivo quantistico programmabile sarà in grado di risolvere problemi che nessun computer classico può risolvere in un tempo fattibile. Prima di addentrarci nel vivo della computazione quantistica, si deve descrivere su quale modello si basa quella classica per meglio capirne le differenze. La computazione classica si basa sul modello astratto della Macchina di Turing, definito nel 1936 dal matematico inglese A. Turing e successivamente rielaborato da John von Neumann negli anni '40; fonda i suoi principi sul concetto di bit: unità di misura con il quale i calcolatori informatici codificano l'informazione. Matematicamente, il bit è una cifra binaria che assume i valori 0-1 che costituiscono il dominio di realizzazione con il quale si classifica uno stato logico, caratterizzato da due azioni opposte equiprobabili.

R. Feynman successivamente, dimostrò che nessuna Macchina di Turing classica potesse simulare certi fenomeni fisici senza incorrere in un inevitabile rallentamento esponenziale delle sue prestazioni. Un "*simulatore quantistico universale*" invece, avrebbe potuto effettuare

la simulazione in maniera più efficiente. Nel 1985 infine, D.Deutsch materializzò queste idee nella Macchina di Turing Quantistica Universale.

Successivamente diversi team di ricercatori si sono impegnati per definire applicazioni della programmazione quantistica a problemi attualmente molto complessi. Sono state elaborate diverse promettenti applicazioni della programmazione quantistica come ad esempio la risoluzione di vari problemi nei campi dell'apprendimento automatico, della crittografia e della chimica. Lo sviluppo di software quantistico su larga scala invece sembra essere purtroppo ancora lontano. A questo proposito, ricercatori come Piattini et al. [1, 5, 6] Miquel et al. [7] e Zhao [8] hanno sostenuto la necessità di una nuova disciplina scientifica in grado di rielaborare ed estendere l'ingegneria del software classica al dominio quantistico. Questo nuovo campo, che dovrebbe consentire agli sviluppatori di progettare e sviluppare programmi quantistici con la stessa sicurezza dei programmi classici, è quello che chiamiamo *ingegneria del software quantistico (QSE)*[8]

2.2 I Fondamenti della computazione quantistica

Per entrare nel vivo del Quantum Computing bisogna conoscere alcune importanti definizioni come, i circuiti quantistici e il fulcro del quantum computing ovvero il qubit.

2.2.1 I circuiti quantistici

Definizione 2.0.1

Un circuito quantistico è un processo computazionale che riceve in input uno stato iniziale e fornisce in output il risultato di una misurazione. Esso è composto da un insieme di corde, che fanno riferimento ai qubits del sistema, e da una serie consecutiva di gates, attraverso cui è possibile compiere opportune trasformazioni lineari del vettore di stato in superposition[9]

Le parti di cui è composto un circuito, hanno un preciso ordine sequenziale, di cui si fornisce l'elenco nel seguito esposto:

1. **Inizializzazione:** in un primo momento si definisce il numero di qubits che viene utilizzato per la simulazione dell'esperimento di interesse. Ogni qubits è rappresentato da una linea orizzontale e la convenzione esige che esso sia inizializzato a zero. Se necessario, può essere fatto uso di ulteriori bits classici o qubits chiamati ancillas, che servono alla memorizzazione intermedia di alcune grandezze fisiche, o per l'esecu-

zione di particolari algoritmi classici. Inoltre, alla fine di ogni circuito c'è una fase di measurement nella quale viene conservato l'output binario finale di ogni misurazione;

2. **Trasformazione:** segue una fase di manipolazione dello stato, in cui intervengono i gates quantistici che compiono opportune trasformazioni sul sistema generale. Tutto ciò in base al fine richiesto dal circuito quantistico. Di solito intervengono successioni di gates atti ad approssimare note funzioni chiamate oracles. Questa pratica consente di ricreare lo stato quantistico richiesto dal problema, nella fase precedente alle misurazioni;
3. **Misurazione:** la fase finale è quella in cui viene eseguito il processo del measurement, il quale possiede natura probabilistica. Esso collassa lo stato del sistema su uno dei possibili stati della base computazionale, fornendo così l'output finale. Tale risultato viene conservato sulle corde del registro riservato ai bit classici, che di norma si trovano nella parte inferiore del circuito.

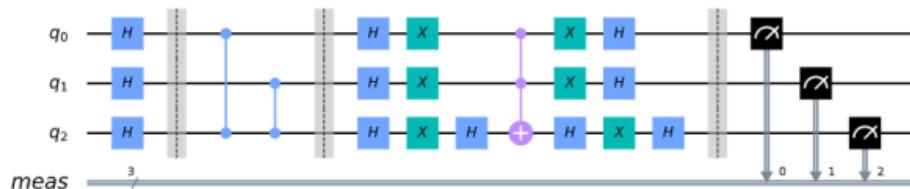
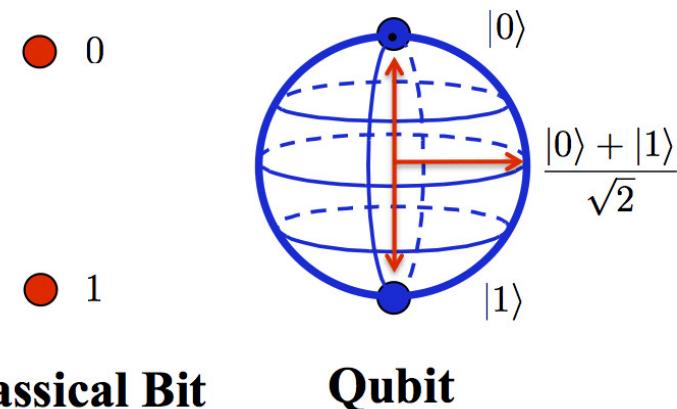


Figura 2.1: Esempio circuito

Definizione 2.0.2

Un qubit usa i fenomeni meccanici quantistici della sovrapposizione per ottenere una combinazione lineare di due stati. Un bit binario classico può rappresentare solo un singolo valore binario, ad esempio 0 o 1, ovvero può trovarsi solo in uno di due stati possibili. Un qubit, tuttavia, può rappresentare uno 0, un 1 o qualsiasi proporzione di 0 e 1 nella sovrapposizione di entrambi gli stati, con una determinata probabilità che si tratti di uno 0 e che si tratti di un 1.[10]

Nella figura 2.2 è visibile una rappresentazione grafica della differenza succitata.

**Figura 2.2:** Differenza tra Bit e Qubit

Dopo la definizione formale di qubit analizziamo le sue proprietà fondamentali:

- **Sovrapposizione di stati**
- **Entanglement**

2.2.2 Sovrapposizione di Stati

Il Qubit presenta una importante caratteristica che lo distingue dal bit normale ovvero la possibilità di avere diversi stati infatti può essere:

- **Vero**
- **Falso**
- **In parte Vero e in parte Falso** (Andando a misurare il Qubit si può leggere Vero oppure Falso con probabilità dipendente da quanta parte è Vero e quanta parte è Falso)

Tutto questo è possibile poichè si basa sul primo postulato della meccanica quantistica che afferma che due o più stati quantistici possono essere sovrapposti. Il risultato di questa somma sarà un altro stato quantistico valido. Allo stesso modo, uno stato quantistico può essere rappresentato dalla somma di due o più stati.

Si tratta di un principio applicabile a tutti quei sistemi che ammettano stati fondamentali che generano stati possibili. Il moto di un qualunque corpo, ad esempio, può essere scomposto nella somma di un termine di moto libero e di un moto forzato. In informatica viene usato per scomporre un problema in altri sotto-problemi più semplici; in elettronica nel risolvere un circuito calcolando tensioni e correnti separatamente. Spesso vengono usate definizioni come “due o più funzioni d’onda che descrivono lo stesso stato quantistico”. Una funzione d’onda non è altro che una funzione matematica complessa delle coordinate spaziali e temporali [11]

2.2.3 Entanglement

"*Entanglement*" (in inglese, "groviglio", "intreccio") è un termine coniato da **Erwin Schrödinger** nel **1935** ed in meccanica quantistica, indica un legame fra particelle. È definito da una funzione, chiamata funzione d'onda di un sistema, che descrive le proprietà delle stesse come fossero un unico oggetto, anche se a enorme distanza. Questa correlazione permette alla prima particella di influenzare la seconda istantaneamente, e viceversa.

Ma non tutte sono "*entangled*", ovvero aggrovigliate. Affinché questa correlazione abbia luogo, cioè per far sì che le due particelle abbiano stati quantici correlati, devono essere prodotte simultaneamente da un'interazione fisica. Un tipico esempio di stato quantico è lo spin di una particella. Lo spin è una grandezza associata alla rotazione delle particelle sul proprio asse ed è stato introdotto matematicamente da W.Pauli per distinguere le particelle che non possono trovarsi in stati energetici affini. Esso può assumere valore positivo o negativo. Quando si ha a che fare con particelle "*entangled*", quindi unite nel legame, la somma degli spin delle due particelle è pari a zero. Pertanto, se si misura lo spin di una delle due, automaticamente e istantaneamente si conoscerà anche lo spin dell'altra. È come prendere un paio di guanti e chiuderli separatamente in due scatole diverse. Se aprodo la prima scatola si trova il guanto destro, si saprà immediatamente che nella seconda c'è il sinistro.

Il solo entanglement però non è sufficiente poichè nel momento della lettura di un qubit ad esempio, non si ha la certezza di quale sarà l'informazione trasmessa. Se da un lato ciò è possibile, bisogna tenere conto che il trasferimento dell'informazione con fedeltà assoluta, implica la distruzione dell'informazione originale.

L'entanglement permette quindi non solo la creazione di reti quantistiche di trasmissione dell'informazione, ma fa sì che gli stessi dati all'interno di un computer quantistico possono essere scambiati in questo modo: questo fenomeno è possibile anche tra moltissime particelle in forme che attualmente sono oggetto di ricerca. Sebbene per molti anni gli scienziati abbiano rifiutato di crederci, questo fenomeno è stato più volte dimostrato, osservato e ricreato in laboratorio. La sua importanza è collegata per di più a un ulteriore principio quantistico: la non località.

L'entanglement non dipende infatti dalla distanza tra le particelle che possono essere lontane migliaia di chilometri l'una dall'altra, rendendo ancora più vasto il numero dei possibili modi in cui l'informatica potrebbe sfruttare questo fenomeno.

Tra il 2012 e il 2013 è stato inoltre dimostrato in alcuni esperimenti, che la non località

si estende anche al dominio del tempo. Infine per avere una visione completa è opportuno fornire una definizione generale di Quantum Computer.

Definizione 2.0.3

Un computer quantistico o calcolatore quantistico è un computer che sfrutta le leggi della fisica e della meccanica quantistica per l'elaborazione dei dati sfruttando come unità fondamentale il qubit.[12]

2.3 Peculiarità e problematiche del computer quantistico

Dopo aver illustrato i fondamenti della computazione quantistica, spiegheremo le problematiche e le peculiarità di un computer quantistico. Come descritto precedentemente, il fulcro della programmazione quantistica è il qubit; molto potente ma allo stesso tempo delicato, tutto ciò perché perde rapidamente le sue speciali qualità quantiche (in circa 100 microsecondi) a causa di molteplici fattori quali le vibrazioni, le fluttuazioni della temperatura dell'ambiente e le onde elettromagnetiche. Questo perchè le particelle sono volatili e fragili, proprio a causa del loro cambiamento di stato, e potrebbero determinare la perdita di dati ed informazioni utili al processo di calcolo.

Per il funzionamento della tecnologia quantistica inoltre, sono necessarie temperature molto basse, vicino allo zero assoluto (circa -273° C). Per ottenerle finora il metodo più comune è ricorrere ai gas liquefatti (come l'isotopo elio-3), sistema però molto costoso. Un team di ricercatori della TUM (Technical University of Munich) [13] ha sviluppato un sistema di raffreddamento magnetico per temperature estremamente basse, adatto per l'elettronica quantistica, che evita il ricorso all'elio-3 ed è già commercializzato dalla startup Kiutra [14].

Secondo un report della McKinsey & Company, multinazionale di consulenza strategica, l'hardware rappresenta ancora il vero “collo di bottiglia” per il calcolo quantistico e le sue applicazioni. Bisogna aumentare il numero di qubit di un computer quantico, garantendone allo stesso tempo qualità adeguata. L'hardware ha inoltre un'importante barriera d'ingresso, poiché richiede un mix raro di capitale, esperienza nella fisica quantistica sperimentale e teorica nonché di competenza approfondita. Specialmente conoscenza degli ambiti specifici per le singole opzioni di implementazione, si legge nel report.

2.4 Informatica quantistica: dall’hardware al software

In questo paragrafo si descrive sommariamente come venga realizzato un computer quantistico.

A tal scopo si focalizza l’attenzione sull’hardware e sul software dei computer quantistici sviluppati dalle aziende IBM e Google, illustrando la tecnologia dei “*qubit superconduttori*”, su cui si basa l’hardware di IBM e Google.

È opportuno sottolineare che questo capitolo è solo una breve introduzione all’hardware quantistico e che un’analisi approfondita va oltre lo scopo di questa tesi.

Si analizza poi il software necessario come interfaccia tra l’apparato hardware quantistico e gli utenti finali. In particolare, i framework software quantistici forniti da IBM e Google, sono rispettivamente Qiskit e Cirq. Questa sezione fornisce gli strumenti per creare un circuito quantistico su ciascun software, eseguirlo su un simulatore o su un dispositivo quantistico reale, quando previsto, e visualizzare i risultati dei dati. Entrambi i software utilizzano il linguaggio di programmazione Python.

L’implementazione di un vero hardware quantistico deve soddisfare cinque condizioni, chiamate criteri di Di Vincenzo: [15]

1. **Scalabilità:** un sistema fisico scalabile con qubit ben caratterizzati.
2. **Reset:** la capacità di inizializzare lo stato dei qubit a un semplice stato fiduciale, come $|000\dots0\rangle$.
3. **Tempi di decorrenza lunghi:** tempi di decorrenza lunghi, molto più lunghi del tempo di funzionamento del gate.
4. **Porta "universale":** un insieme "universale" di porte quantistiche.
5. **Lettura efficiente:** una capacità di misura specifica del qubit.

Una delle sfide principali dell’implementazione di un hardware quantistico efficiente è quella di mantenere il sistema disaccoppiato dalle influenze esterne, tranne che durante le fasi di scrittura, controllo e misurazione. Uno dei principali approcci per l’implementazione di un computer quantistico, si basa su “*circuiti integrati quantistici*”, in cui i qubit sono costruiti da modalità elettrodinamiche collettive di elementi elettrici macroscopici. Il vantaggio è che i qubit possono essere accoppiati tra loro tramite elementi elettrici lineari non dissipativi come condensatori, induttori e linee di trasmissione. La parte difficile è isolare questi qubit dal rumore esterno. Il primo requisito di un circuito integrato, per comportarsi in modo

quantomeccanico, è l'assenza di dissipazione per tutte le parti metalliche che costituiscono il circuito alla temperatura di funzionamento del qubit. Pertanto, i superconduttori a bassa temperatura, sono la scelta migliore per questo compito e tale implementazione del circuito integrato quantistico è chiamata "qubit superconduttore". La temperatura del circuito integrato deve essere inferiore all'energia associata alla transizione tra lo stato fondamentale, $|0\rangle$, e lo stato eccitato $|1\rangle$. Anche la temperatura dei fili delle porte di controllo e di lettura collegate al chip deve essere bassa. L'elaborazione dei segnali quantistici viene eseguita nei circuiti integrati da elementi dissipativi non lineari che consistono in giunzioni tunnel superconduttrici, chiamate anche giunzioni Josephson.

2.4.1 Software quantistico: Qiskit e Cirq

Oggi è disponibile un'ampia gamma di software di calcolo quantistico. La maggior parte delle aziende di calcolo quantistico, come IBM, garantisce l'accesso ad un vero computer quantistico grazie alla piattaforma cloud e l'interazione tra utenti e computer quantistici, è mediata da una piattaforma software con accesso API (Application Programming Interface).

Altre aziende, come Google, non hanno ancora la possibilità di connettersi ad un computer quantistico e forniscono solo il simulatore. Quando si utilizza un'interfaccia software per comunicare con un computer quantistico reale o simulato, emergono molte considerazioni pratiche. Ad esempio, possiamo chiederci quanto sia facile creare, lavorare e manipolare i circuiti quantistici o quanto sia facile parametrizzare gli algoritmi per il calcolo quantistico a breve termine.

Un altro quesito può essere quello di conoscere se il simulatore di computer quantistici possa essere usato per testare gli algoritmi e quanti qubit possono essere simulati in quel preciso momento. Se la piattaforma dà accesso a dispositivi quantistici reali, possiamo chiedere quali siano le loro caratteristiche principali, come il numero di qubit o le operazioni di gate nativamente integrate. È anche importante capire come si possano visualizzare i risultati dei dati.

In questa sezione proviamo a rispondere a queste domande per i framework software quantistici di IBM e Google e Microsoft, rispettivamente Qiskit, Cirq e Azure Quantum. Il primo fornisce l'accesso a dispositivi quantistici reali, il secondo rende disponibili solo simulatori mentre il terzo è l'ambiente di sviluppo perfetto poiché offre un set eterogeneo di soluzioni e tecnologie quantistiche.

2.4.2 Qiskit software

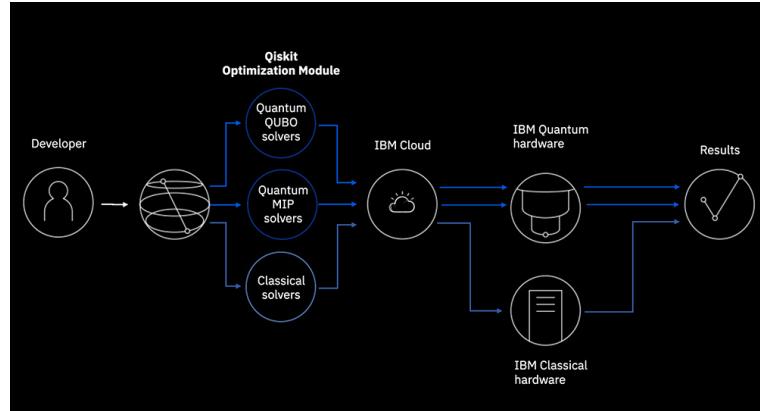


Figura 2.3: La struttura di funzionamento di Qiskit

Qiskit è un framework open-source per lavorare con il linguaggio quantistico OpenQASM e i processori quantistici dell’esperienza IBM Q. È organizzato in quattro componenti distinti:

- **Terra:** Qiskit Terra fornisce strumenti per la composizione di circuiti quantistici a livello di codice macchina quantistico e gestisce l’esecuzione di tale codice su hardware quantistico. Fornisce, inoltre, strumenti che consentono di ottimizzare i circuiti quantistici per un particolare dispositivo.
- **Aer:** Qiskit Aer fornisce simulatori per lo studio di algoritmi e applicazioni di calcolo quantistico in dispositivi affetti da rumore; permette di costruire modelli di rumore per simulazioni rumorose.
- **Ignis:** Qiskit Ignis include strumenti per la caratterizzazione degli errori attraverso metodi come la tomografia e per la correzione degli errori quantistici.
- **Aqua:** Qiskit Aqua fornisce algoritmi quantistici che possono essere utilizzati direttamente per costruire applicazioni di calcolo quantistico. [16]

2.4.3 Cirq software

Cirq è un framework open-source che fornisce un modello di programmazione semplice che astrae i mattoni fondamentali delle applicazioni quantistiche. La versione iniziale di Cirq è stata implementata in Python ma è ancora in via di sviluppo. Dal punto di vista architettonico, Cirq modella un’applicazione quantistica utilizzando una serie di costrutti di base che sono:

- **Circuits:** Qualsiasi programma quantistico all’inizio di Cirq è estratto da una delle due classi fondamentali: un Circuit o un Schedule. In Cirq, un Circuit rappresenta la forma più elementare di un circuito quantistico. Un circuito Cirq è rappresentato come una raccolta di Momenti che includono operazioni che possono essere eseguite su Qubit durante un’astratta diapositiva del tempo.

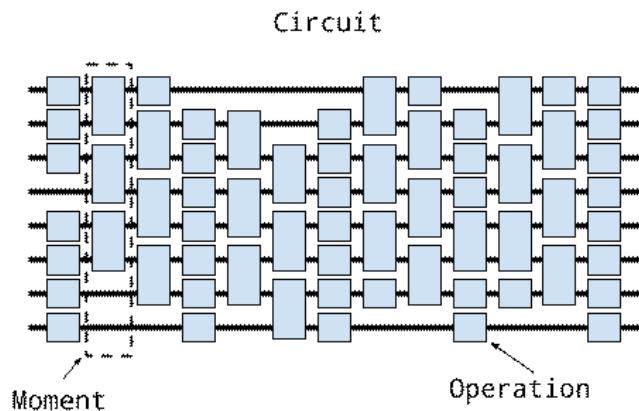


Figura 2.4: La struttura di un circuito di Cirq

- **Devices:** il costrutto Device in Cirq rappresenta i vincoli di un tipo specifico di hardware quantistico. Ad esempio, la maggior parte dell’hardware consente solo l’attivazione di determinati gate sui qubit. Oppure, come altro esempio, alcuni gates potrebbero essere vincolati a non essere in grado di funzionare contemporaneamente ai gates vicini.
- **Gates:** in Cirq, Gates astrae operazioni su raccolte di qubit.
- **Simulators:** Cirq include un simulatore Python che può essere utilizzato per eseguire circuiti e pianificazioni. L’architettura del simulatore può scalare su più thread e CPU, il che gli consente di eseguire circuiti abbastanza sofisticati.[17]

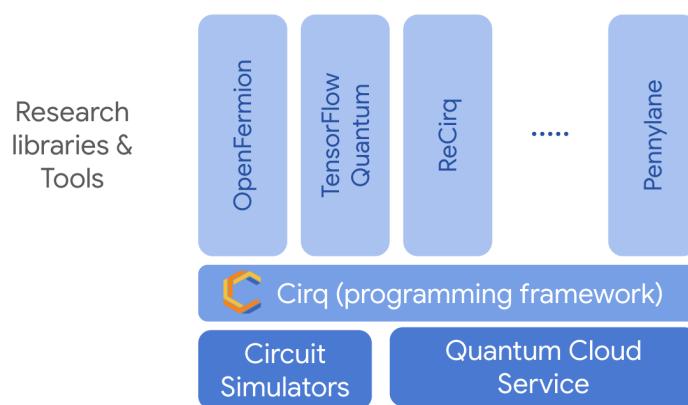


Figura 2.5: La struttura di funzionamento di Cirq

2.4.4 Azure Quantum

Azure Quantum è il servizio per il calcolo quantistico di Azure ed offre un set eterogeneo di soluzioni e tecnologie quantistiche. Azure Quantum garantisce un percorso aperto, flessibile e futuro per il calcolo quantistico che si adatta al proprio modo di lavorare, accelera il progresso e protegge gli investimenti tecnologici.

Offre l’ambiente di sviluppo migliore per creare algoritmi quantistici per più piattaforme contemporaneamente mantenendo la flessibilità per ottimizzare gli stessi algoritmi per sistemi specifici. È possibile scrivere una volta il codice ed eseguirlo con modifiche ridotte o nulle in più destinazioni della stessa famiglia. Ciò consente di concentrarsi sulla programmazione a livello di algoritmo.

- **Un ecosistema aperto**, che consente di accedere a software, soluzioni e hardware quantistici di vario tipo da Microsoft e dai rispettivi partner. È possibile scegliere tra linguaggi di programmazione quantistici, ad esempio Qiskit, Cirq e Q# ed eseguire gli algoritmi in più sistemi quantistici.
- L'impatto quantistico consente di esplorare simultaneamente i sistemi quantistici di oggi e di essere pronti per i sistemi quantistici scalabili del futuro e con soluzioni predefinite eseguite su risorse di calcolo classiche e accelerate (denominate anche soluzioni di ottimizzazione).

Azure Quantum offre due percorsi principali per le soluzioni quantistiche:

- *Calcolo quantistico*: è possibile imparare, sperimentare e creare prototipi con diversi provider di hardware quantistico per prepararsi al futuro dei computer quantistici ridi-

mensionati. A differenza delle altre soluzioni, non si è limitati a una singola tecnologia hardware ed è possibile sfruttare un approccio a stack completo, in modo da proteggere gli investimenti a lungo termine.

- *Ottimizzazione*: è possibile sviluppare soluzioni utili per ridurre i costi operativi in una vasta gamma di campi, tra cui settore finanziario, costi energetici, gestione delle flotte, pianificazione e altro ancora.

Grazie ad Azure Quantum e al set di strumenti kit di sviluppo Microsoft Quantum, sarà possibile programmare gli algoritmi quantistici e le soluzioni di ottimizzazione e quindi applicare tali soluzioni quantistiche nella piattaforma di Azure esistente per ottenere impatti concreti anche prima dello sviluppo di un computer quantistico per utilizzo generico.[18]

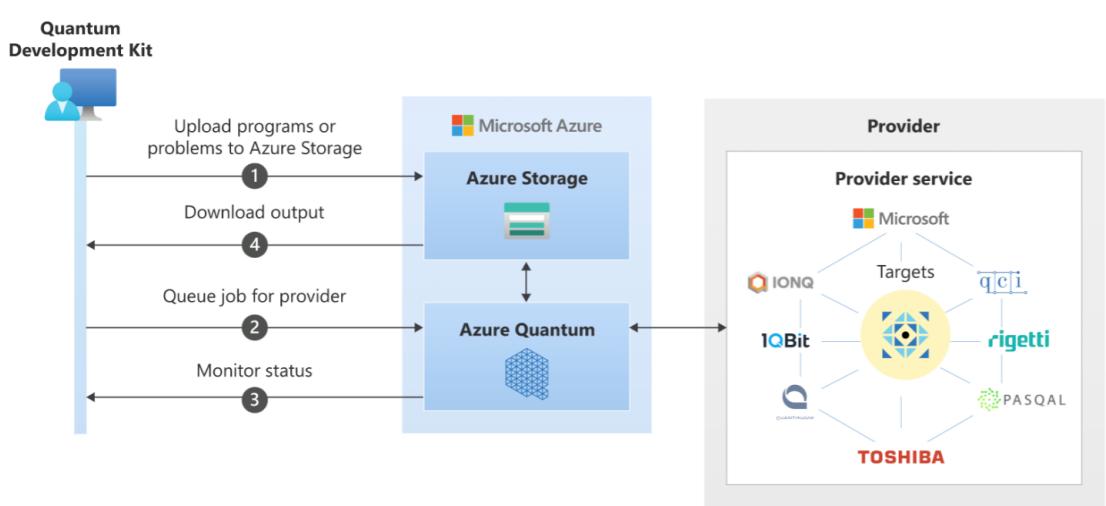


Figura 2.6: La struttura di funzionamento di Azure Quantum

Dopo aver fatto una panoramica delle principali tecnologie utilizzate per la computazione quantistica andremo a descrivere tre progetti a cui ci si è riferiti per il lavoro di tesi.

2.4.5 Amazon Braket

Amazon Braket [19] è un servizio gestito da un unico luogo gestito da Aws che consente a scienziati, ricercatori e sviluppatori di eseguire esperimenti quantistici con i computer di più fornitori di hardware quantistico. Il servizio di Amazon Web Services prende il nome da Bra-Ket, termine comunemente utilizzato in meccanica quantistica per rappresentare gli stati quantistici.

Il servizio va considerato nelle attività che Amazon ha messo in campo da tre anni a questa parte per sviluppare la propria strategia di quantum computing.

Quindi Braket è un servizio di calcolo quantistico gestito progettato per aiutare a velocizzare la ricerca scientifica e lo sviluppo di software per il calcolo quantistico. Un esempio di caso d’uso può essere la ricerca di algoritmi di calcolo quantistico, il test di diversi hardware quantistici, la creazione di software, e la risoluzione di problemi computazionali complicati.

Il servizio è progettato per consentire un’esperienza pratica con qubit e circuiti quantistici; Permette di costruire e testare circuiti in un ambiente simulato e successivamente eseguirli su un vero computer quantistico. Tutto il sistema essendo gestito da Amazon, eredita i servizi di sicurezza e crittografia integrate a ogni livello e per la sua personalizzazione, consente la scrittura di codice in Python, utilizzando l’Amazon Braket SDK. Oltre all’ambiente di simulazione Braket permette anche l’accesso ai computer quantistici di D-Wave, IonQ e Rigetti.

2.4.6 Zapata Computing

Zapata Computing [20] è una startup lanciata con l’intenzione di creare e vendere software per computer quantistici. Il team di Zapata ha in programma di supportare tutti i principali produttori di hardware, compresi IonQ e Rigetti. Hanno una partnership con IBM e stanno cercando di farlo con altri sviluppatori di hardware. Il loro lavoro si focalizza sullo sviluppo di una piattaforma che permetta di lavorare con computer quantistici in modo semplice ed immediato utilizzando diversi ambienti in funzione delle esigenze.

2.4.7 QuantumPath

QPath (QuantumPath) [21] è la prima piattaforma al mondo che mira ad accelerare l’accesso delle aziende all’informatica quantistica. Le caratteristiche principali della piattaforma sono le seguenti:

- Creazione di risorse per applicazioni quantistiche e possibilità di impostare i requisiti dell’ambiente;
- Progettazione visiva degli algoritmi quantistici;
- Ciclo di vita completamente automatizzato;
- Possibilità di scelta di computer quantistici e ambienti di simulazione;
- Integrazione delle applicazioni personali quantistiche nel sistema;
- Esplorazione dei risultati utilizzando uno schema unificato;

- Gestione di tutti i processi e analisi della telemetria archiviata,

2.5 Primi passi nell’ingegneria del software quantistico

Il lavoro svolto da **El Aoun et al.**[22] si basa su uno studio empirico delle domande poste dagli sviluppatori quantistici sui forum di Stack Exchange e sui problemi segnalati su GitHub. Gli autori hanno eseguito analisi di codifica qualitativa e sulla modellazione automatica degli argomenti, per individuare gli argomenti dei post e delle segnalazioni di problemi dell’ingegneria del software quantistico. Tutto ciò ha permesso di ottenere un quadro generale potendo quindi iniziare a risolvere i problemi riscontrati. Secondo i risultati riportati, la conoscenza della teoria quantistica e la comprensione dei programmi quantistici rappresentano aspetti ingegneristici chiave che ostacolano le capacità degli sviluppatori di sviluppare programmi quantistici. In parole semplici, se non si ha una corretta conoscenza della teoria quantistica con tutti i suoi derivati, si riscontra una notevole difficoltà a comprendere e quindi sfruttare, tramite programmi, la potenza del calcolo quantistico.

Un’altro interessante lavoro è quello esposto nel primo *International Workshop on Quantum Software Engineering*, dove ricercatori e professionisti hanno proposto un manifesto QSE, noto come "Talavera Manifesto", che definisce l’insieme dei principi fondamentali di questa nuova disciplina [5, 6]. Alcuni di questi principi includono i seguenti punti:

- Agnosticismo verso specifiche tecnologie quantistiche
- Coesistenza di programmazione classica e quantistica
- Supporto per lo sviluppo e la manutenzione di software quantistico

Da allora, sono stati presentati diversi studi che trattano le sfide e le potenziali direzioni della ricerca sull’ESS in varie prospettive. **Piattini et al.** [1] hanno definito una serie di argomenti su cui i ricercatori dovrebbero prestare attenzione. In particolare, dopo una digressione sulle "*Golden Erases of Software Engineering*", sono stati analizzati alcuni settori prioritari per l’ingegneria del software quantistico. Per essere più precisi, quattro aree dell’ingegneria che necessitano di grande attenzione da parte degli sviluppatori:

1. La progettazione del software di sistemi ibridi quantistici;
2. Le tecniche di test per la programmazione quantistica;
3. La qualità dei programmi quantistici;

4. La reingegnerizzazione e la modifica verso sistemi informativi classico-quantistici

Oltre che definire le aree su elencate, forniscono alcuni spunti utili per ricercatori, professionisti ed università, descritti di seguito. I ricercatori dovrebbero considerare che molti informatici quantistici non hanno una conoscenza sufficiente delle tecniche di ingegneria del software, per cui, molti errori, potrebbero essere ripetuti. Inoltre, ritengono che i ricercatori non debbano aspettare che i linguaggi di programmazione quantistica siano "stabili" o "raffinati" per proporre, adattare o sviluppare tecniche di ingegneria del software quantistico, ma svilupparle parallelamente all’evoluzione dei linguaggi quantistici. Infine, sostengono che i ricercatori dovrebbero fare tesoro degli errori del passato ed affidarsi alla convalida empirica quando propongono nuove tecniche di ingegneria del software quantistico.

Un’altro lavoro di rilevanza è quello di Zhao et al.[8], nel quale si effettua una panoramica completa dell’ingegneria del software quantico, includendo tutti gli aspetti del ciclo di vita del software quantico (analisi dei requisiti, progettazione, implementazione, test) e l’argomento critico del riuso dello stesso. Lo studio ha anche esplorato alcune delle sfide e delle opportunità del settore. Si riconosce infatti quanto sia importante che emerga una linea guida completa per lo sviluppo di software quantistico. Inoltre, questo lavoro definisce il ciclo di vita del software quantistico e lo sfrutta come base per un’indagine completa degli attuali sforzi di ricerca nel settore. Presenta infine anche una panoramica sui test e sulla manutenzione del software quantistico.

Piattini et al. [1] ha invece condotto un lavoro che non sfrutta uno studio empirico per scoprire le sfide del campo, ma si basa sulle domande aperte poste ed individuate dallo studio effettuato da **El Aoun et al.** già descritto precedentemente.

Fino ad ora si è evidenziato come i principali lavori si siano incentrati sul creare una guida allo sviluppo di applicazioni quantistiche ed inoltre come molti lavori si sono occupati di raccogliere tutte le principali problematiche che gli sviluppatori incontrano giorno dopo giorno lavorando con un’ambiente quantistico.

In relazione ai lavori finalizzati alla raccolta di informazioni, non possiamo non descrivere lo studio empirico portato avanti da **Manuel De Stefano et al.** [23]. L’obiettivo del lavoro, è stato indagare l’attuale utilizzo delle tecnologie di programmazione quantistica e di esplorare le sfide che gli sviluppatori quantistici si trovano ad affrontare al giorno d’oggi, con lo scopo di valutare dove i metodi e le pratiche di ingegneria del software possono essere applicati. Inoltre si è incentrato sull’analisi di tutti i repository GitHub che fanno uso dei framework di programmazione quantistica più utilizzati attualmente sul mercato. In seguito sono state

condotte sessioni di analisi della codifica per produrre una tassonomia degli scopi per i quali vengono utilizzate le tecnologie quantistiche. Infine, un'indagine che ha coinvolto i collaboratori dei repository considerati, con l'obiettivo di raccogliere le opinioni degli sviluppatori sull'attuale adozione e sulle sfide della programmazione quantistica.

Le domande poste agli sviluppatori sono raggruppate nella Figura 2.7:

Testo della domanda	Tipo di risposta	Risposte possibili
Parte 1 - - - Contesto		
Qual è il suo attuale stato occupazionale?	Scelta multipla	Studente di laurea; studente di master; studente di dottorato; ricercatore; sviluppatore open source; sviluppatore industriale; altro
Qual è il suo percorso formativo?	Scelta singola	Informatica; Chimica; Fisica; Altro
Qual è la vostra fascia d'età?	Scelta singola	18-24; 25-34; 35-44; 45-54; 55 +
Qual è il tuo genere?	Testo libero	-
Indicare la propria esperienza (in anni) nello sviluppo di software.	Scelta singola	Nessuno; 0-3; 3-5; 5-10; 10+
Indicare la propria esperienza (in anni) nello sviluppo industriale.	Scelta singola	Nessuna; 0-3; 3-5; 5-10; 10+
Indicare la propria esperienza (in anni) nella programmazione quantistica.	Scelta singola	Nessuna; 0-3; 3-5; 5-10; 10+
Qual è il suo Paese?	Testo libero	-
Parte 2 - - - Adozione attuale		
Con quale tecnologia quantistica vi sentite più sicuri?	Scelta singola	Qiskit; Cirq; Q#; Altro
Quale altra tecnologia quantistica utilizzate?	Scelta multipla	Qiskit; Cirq; Q#; Altro
In quale contesto si utilizza l'informatica quantistica?	Scelta multipla	Studio accademico; Hackaton; Industria; OSS; Studio personale; Ricerca; Altro
Potrebbe illustrarci meglio i compiti che svolgete con l'informatica quantistica?	Testo libero lungo	-
Parte 3 - - Potenziale adozione e sfide		
Considerate la tecnologia con cui avete maggiore confidenza. Quali sono state le 3 principali sfide che avete affrontato?	Testo libero multiplo	-
In base alla vostra esperienza, avete mai risolto (o cercato di risolvere) un problema con la programmazione quantistica che non ha una soluzione "tradizionale" (o la soluzione è intrattabile)?	Scelta singola	Sì; No
In caso affermativo, potrebbe spiegare meglio il problema e il motivo per cui è necessario utilizzare l'informatica quantistica?	Testo libero lungo	-
In base alla vostra esperienza, avete mai risolto (o cercato di risolvere) un problema che ha una soluzione "tradizionale" utilizzando la programmazione quantistica?	Scelta singola	Sì; No
In caso affermativo, potrebbe spiegare meglio di cosa si trattava e perché ha scelto di utilizzare l'informatica quantistica?	Testo libero lungo	-

Figura 2.7: Domande poste nel sondaggio

Questo studio è stato fondamentale poichè ha rivelato che la programmazione quantistica viene utilizzata principalmente per scopi didattici o per la curiosità di sperimentare le tecnologie quantistiche. Inoltre, l'indagine ha fatto luce su diverse sfide che non sono solo legate agli aspetti tecnici dello sviluppo quantistico, ma anche a quelli socio-tecnici, come le sfide di comprensione, il grado di realismo e la comunità. Gli autori inoltre, affermano che i risultati ottenuti dallo studio, hanno posto le basi per altri approfondimenti con l'obiettivo di affrontare le sfide identificate, fornendo un supporto (semi)automatico agli sviluppatori in termini di comprensione, analisi, manipolazione e test dei programmi quantistici.

Una notevole difficoltà è anche l'integrazione della componente quantistica nelle attuali architetture IT. Molte aziende stanno indirizzano i loro investimenti nella modernizzazione delle proprie infrastrutture, puntando ad'integrazione quantistica. Tutto ciò perchè, avere nei processi aziendali una potenza di calcolo nettamente superiore, consentirebbe di raggiungere alti valori qualitativi e, conseguentemente, conquistare nuove opportunità di mercato. Tale modernizzazione non significa però, scartare completamente i sistemi informativi classici. Alcuni dei processi aziendali supportati, infatti, non saranno probabilmente gestibili dall'informatica quantistica e, anche se quantizzati, il loro valore economico sarà basso rispetto

all’investimento.

La soluzione dell’evoluzione dei sistemi attuali verso il calcolo quantistico potrebbe essere nello sviluppo di sistemi classici-quantistici, denominati infatti, sistemi informativi ibridi. Sono costituiti da un sistema tradizionale principale che invia richieste ai computer quantistici (tipicamente nel cloud) per calcolare algoritmi quantistici specifici. Tuttavia, l’evoluzione dai sistemi classici all’ibrido non è un processo semplice. Molti lavori hanno affrontato questa integrazione; uno di esso è quello svolto da **Luis Jiménez-Navajasa et al.** che ha proposto una soluzione che si basa sulla reingegnerizzazione, in particolare sulla modernizzazione basata sull’architettura [24]. E’ infatti un’evoluzione della riprogettazione tradizionale che segue l’approccio Model-Driven Engineering (MDE), metodologia di sviluppo software che si focalizza sulla creazione di modelli o astrazioni più vicina a particolari concetti di dominio piuttosto che a concetti di computazione o di algoritmica. Tale approccio sostiene l’uso di KDM (Metamodello di Knowledge Discovery) [25]. KDM garantisce la rappresentazione, in modo indipendente dalla tecnologia, dell’architettura del sistema comprese le sue interrelazioni. L’astrazione ottenuta tramite KDM, garantisce l’interoperabilità tra gli strumenti di reverse engineering e altri strumenti di modernizzazione del software. Nel paragrafo successivo verrà descritto nei particolari la soluzione proposta da **Luis Jiménez-Navajasa et al.**

2.5.1 Reingegnerizzazione quantistica del software

La reingegnerizzazione del software quantistico potrebbe essere la soluzione per affrontare le sfide che l’evoluzione verso i sistemi ibridi comporta. Sulla base della modernizzazione del software (la sua evoluzione aggiungendo un approccio MDE), l’ingegneria del software quantistico potrebbe essere utilizzata in tre scenari complementari:

1. Migrare algoritmi quantistici esistenti ed isolati nonchè integrarli nei sistemi informativi ibridi.
2. Migrare i sistemi informativi legacy classici verso architetture ibride che supportano l’integrazione dei sistemi informativi classici-quantistici.
3. Trasformare o aggiungere nuove operazioni aziendali supportate da software quantistico che verranno integrate nei sistemi ibridi target.

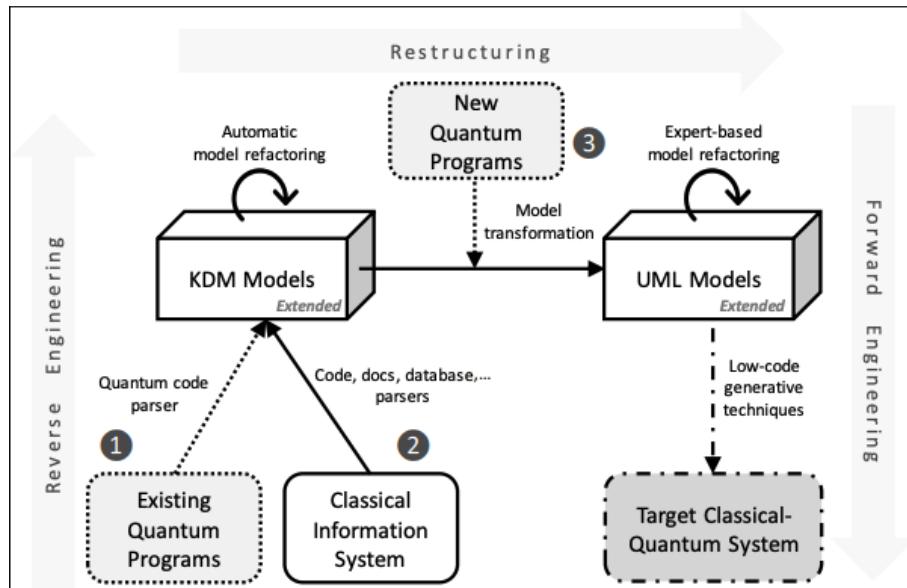


Figura 2.8: Processo di reingegnerizzazione quantistica

Nella Figura 2.7 è illustrato il processo complessivo di reingegnerizzazione quantistica, dove si può notare che, la soluzione proposta all’evoluzione verso i sistemi ibridi, utilizza standard come KDM e UML come core principali. La prima fase della reingegnerizzazione quantistica è il reverse engineering che analizza gli artefatti dei sistemi informativi esistenti come il codice sorgente, gli schemi di database, ecc. Non solo i sistemi informativi classici (scenario 1) potrebbero essere ispezionati ma anche i programmi quantistici (scenario 2). L’output di questa fase è un insieme di file KDM.

La seconda fase è la ristrutturazione (vedi Figura 2.7), in cui i modelli KDM vengono trasformati in modelli ad alto livello di astrazione che rappresentano sia gli aspetti di analisi che di progettazione del sistema ibrido target con UML. Una volta eseguite le rappresentazioni, gli ingegneri del software potrebbero modellare il target del sistema ibrido.

L’ultima fase consiste nel forward engineering (vedi Figura 2.7), composto da un insieme di tecniche che dalla rappresentazione fatta nella fase precedente, genera il codice sorgente per il sistema ibrido di destinazione. Al giorno d’oggi, esiste un vasto numero di generatori di codice sorgente da modelli UML. Tuttavia, non esistono però, generatori di sorgenti quantistiche da modelli di astrazione di alto livello, e tutto ciò rimane ancora un problema da risolvere.

2.5.2 Quantum service-oriented computing

In questo capitolo si analizzeranno due importanti lavori di progettazione di architetture ibride.

Il primo è quello effettuato da **Kumara et al.** [26]. Esso ha proposto un’architettura di riferimento a strati, per quantum service-oriented computing, la QSOA (Quantum Service-Oriented Architecture). Essa fornisce un quadro architettonico per la costruzione di applicazioni che sfruttano una collezione di unità funzionali riutilizzabili (servizi) con interfacce ben definite ed implementate utilizzando un mix ben bilanciato di approcci di calcolo classici e quantistici (ibridi). La Figura 2.8 illustra l’architettura complessiva di QSOA stratificata che si compone dei seguenti sei livelli, a partire dal dominio concettuale del problema fino all’infrastruttura fisica di calcolo:

1. **Domini aziendali e dati:** è il livello superiore di QSOA che cattura i risultati dell’analisi del dominio del problema.
2. **Processi aziendali e pipeline di dati:** rappresentano i processi aziendali che producono e consumano dati operativi.
3. **Servizi middleware:** offrono le capacità di ospitare, eseguire, monitorare e gestire processi aziendali, pipeline di dati e servizi di componenti. L’implementazione di questi servizi middleware può anche essere un ottimo candidato per sfruttare l’informatica quantistica.
4. **Servizi per i componenti:** corrisponde al dominio dell’applicativo da sviluppare.
5. **Infrastrutture:** livello importante in quanto le applicazioni e i middleware QSOC vengono distribuiti su più infrastrutture tra cui cloud, edge, HPC e quantum. La portabilità e l’adattabilità saranno fondamentali per supportare il passaggio continuo e dinamico tra i diversi tipi di infrastrutture man mano che queste o i loro contesti di utilizzo, si evolvono.

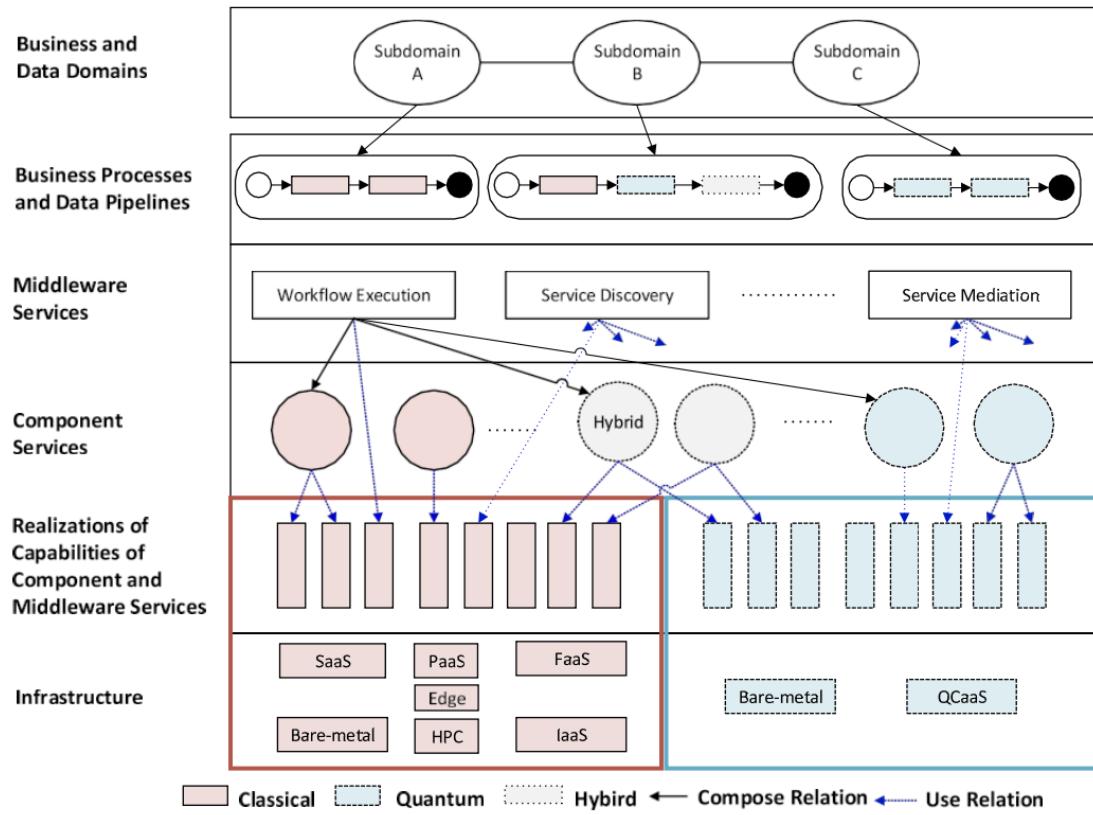


Figura 2.9: Architettura quantistica orientata ai servizi

Nello studio è anche descritta una metodologia QSOA guidata da modelli (MD-QSOA) che facilita gli sviluppatori di applicazioni aziendali e gli esperti quantistici, a progettare, comporre, distribuire, eseguire e gestire applicazioni QSOA eterogenee in modo portatile, condividendo e riutilizzando gli articoli di progettazione e implementazione. La metodologia è mirata al fine di guidare i team DevOps a implementare ed evolvere la prossima generazione di applicazioni aziendali ibride in modo ben strutturato, ripetibile e trasparente. L’approccio è basato sull’ingegneria guidata dai modelli [27] ed è ispirata dai precedenti lavori degli autori sull’ingegneria dei servizi classici e cloud [28, 29]. In particolare, la metodologia QSOA ha tratto vantaggio dai lavori in corso sull’ingegneria delle applicazioni eterogenee [30, 31]. La Figura 2.9 fornisce una panoramica dell’approccio proposto per l’ingegnerizzazione delle applicazioni QSOA. È stata progettata partendo dal presupposto che le applicazioni quantistiche debbano essere confezionate come servizi containerizzati che abbracciano il modello Quantum Algorithm / Application as a Service (QaaS) [32].

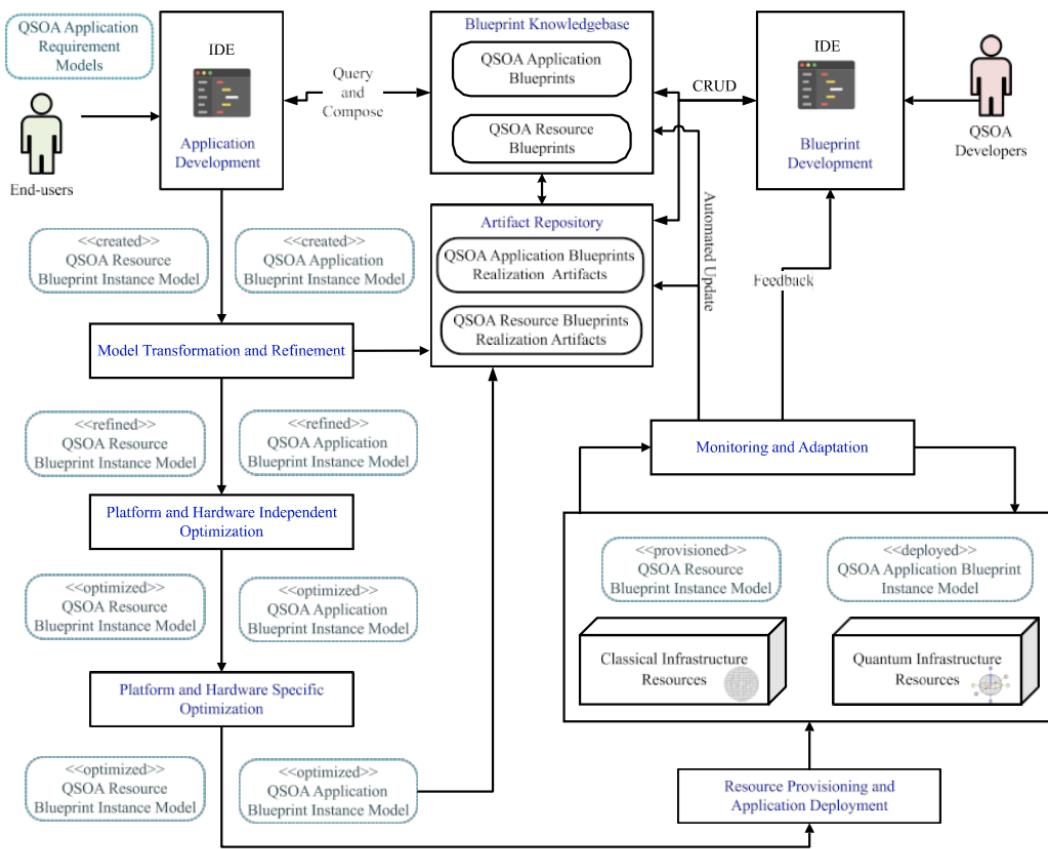


Figura 2.10: Metodologia di progettazione dell'applicazione QSOA basata su modelli

L'elemento critico della metodologia descritta nel lavoro è il model-driven, costituito dai modelli di conoscenza riutilizzabili, blueprints. Identifichiamo due categorie principali di blueprints:

- **Blueprints di applicazioni:** simili a modelli indipendenti dal calcolo (CIM) e dalla piattaforma che compartmentano semanticamente i meta-dati su vari aspetti di un'applicazione QSOA, ad esempio requisiti, contratti di servizio, modelli di processo di alto livello e modelli di architettura di distribuzione.
- **Blueprints di risorse:** catturano i meta-dati sui requisiti operativi, prestazionali e di capacità delle risorse infrastrutturali (ad esempio, cloud, edge, HPC e nodi quantistici) e dei servizi esterni.

Riassumendo, in questo lavoro è stato coniato il concetto di quantum service-oriented computing (QSOC). In particolare, è stata introdotta un'architettura di riferimento per il modello SOA quantistico adottando il collaudato modello SOA stratificato del settore. Inoltre,

è stata presentata una nuova metodologia model-driven per la realizzazione di applicazioni QSOA che facilita le imprese nella costruzione di applicazioni QSOA portabili e adattabili.

Questo studio è collegabile a uno simile, mirato a evidenziare le asperità e le limitazioni che si presentano nell'integrazione di sistemi ibridi classici-quantistici, utilizzando l'informatica orientata ai servizi. Il lavoro che descriveremo è quello effettuato da **Enrique Moguel et al.** [33]. Per sottolineare differenze e difficoltà nello sviluppo di servizi quantistici di qualità, il suddetto lavoro, prende in considerazione un problema ben noto a cui può essere fornita una soluzione quantistica, ovvero la fattorizzazione degli interi, facendo uso della piattaforma di servizi quantistici Amazon Braket. Pertanto, l'obiettivo del lavoro svolto, è stato quello di riuscire a definire la buona prassi per la progettazione di un sistema orientato ai servizi quantistici.

Prima di entrare nel dettaglio, è importante sottolineare come anche se concettualmente l'invocazione di un servizio software quantistico è simile a quella di un servizio classico, tecnicamente esistono differenze e limitazioni tecnologiche che si riferiscono all'indipendenza dalla piattaforma, al disaccoppiamento, alla scalabilità, ecc. Successivamente si descriverà uno dei possibili modi per gestire l'intero ciclo di vita di un servizio classico, a partire dall'implementazione, seguita dalla distribuzione e dal successivo monitoraggio e manutenzione.

La figura 2.11 mostra come potrebbe apparire una buona versione classica dell'implementazione, del deployment e della manutenzione di un servizio.

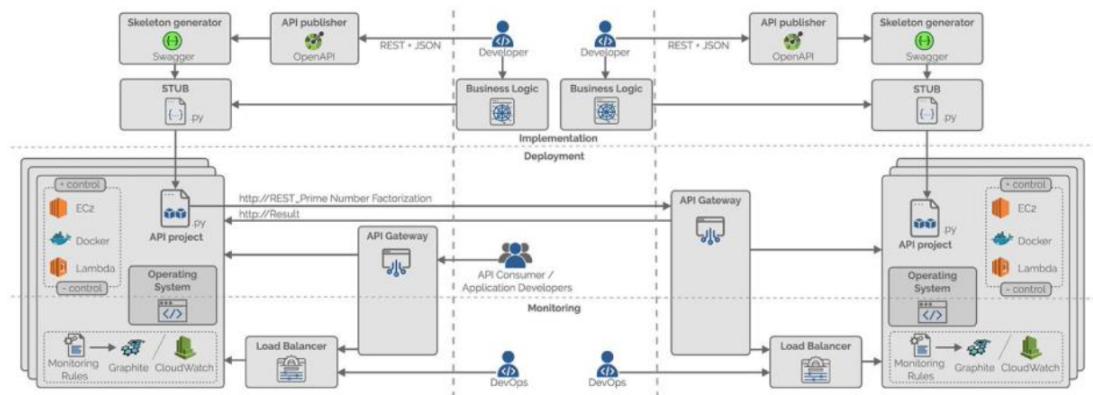


Figura 2.11: Versione classica dell'implementazione di un servizio

La figura 2.10 è divisa verticalmente in due. Sul lato sinistro è visibile il client del servizio (che può essere un servizio stesso). Nello studio in essere, questo servizio si occupa di decodifica crittografica. Sul lato destro invece, il servizio invocato, che nel nostro caso sarebbe

l’algoritmo per il calcolo della fattorizzazione dei numeri interi. In orizzontale, infine, tre livelli che rappresentano le diverse fasi del ciclo di vita del servizio:

- In alto la fase di implementazione del servizio sviluppato;
- Al centro la fase di distribuzione del servizio, che viene pubblicato per poter essere invocato;
- In basso, la fase di monitoraggio in cui il servizio viene utilizzato.

Tale descrizione ha l’obiettivo di permettere di comprendere, genericamente, come possa essere implementato un servizio ma, come già riportato, non si occupa di progettare un servizio classico ma semplicemente descrivere alcuni problemi che non possono essere risolti in modo efficiente con il calcolo classico, proponendone una soluzione.

Un chiaro esempio, che rappresenta anche il focus del lavoro, è quello della fattorizzazione dei numeri primi. Proprio in quest’ambito l’informatica quantistica può offrire i maggiori vantaggi, senza però dimenticare tutte le conoscenze acquisite nell’informatica classica orientata ai servizi.

Per affrontare meglio lo stato attuale dei servizi quantistici, è stato utilizzato Amazon Braket, piattaforma precedentemente descritta. La scelta è ricaduta su di esso poichè è noto quanto Amazon sia il leader mondiale del cloud computing e le tecnologie dei servizi attraverso AWS, pertanto Braket è una buona alternativa per sviluppare servizi quantistici.

Ritornando al lavoro, l’elemento di base del service-oriented computing, è un servizio, definito come un elemento computazionale autodescrittivo e indipendente dalla piattaforma che supporta la composizione rapida e a basso costo di applicazioni distribuite. Tuttavia, Braket non è direttamente predisposto per offrire gli algoritmi quantistici sviluppati come servizi che possono essere invocati attraverso un endpoint per comporre un’applicazione più complessa.

Questa lacuna può essere affrontata avvolgendo l’algoritmo quantistico in un servizio classico. Ciò implica l’inclusione di un computer classico per eseguire il servizio classico che, a sua volta, invoca il computer quantistico. Per quanto ne è a conoscenza degli autori, attualmente non esiste un modo per invocare direttamente un algoritmo quantistico come servizio.

La Figura 2.12 mostra un esempio di tale approccio. Uno dei circuiti quantistici più semplici e noti, quello utilizzato per creare stati di Bell tra due qubit, è avvolto da un servizio Flask, che può essere implementato in un computer classico fornendo un modo semplice per includere algoritmi quantistici in una soluzione complessa orientata ai servizi.

```

from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
import matplotlib.pyplot as plt

from braket.circuits import Circuit
from braket.devices import LocalSimulator

app = Flask(__name__)
CORS(app)

@app.route('/execute', methods=["get"])
def execute_quantum_task():

    bell = Circuit().h(0).cnot(control=0, target=1)
    device = LocalSimulator()
    result = device.run(bell, shots=1000).result()
    counts = result.measurement_counts

    plt.bar(counts.keys(), counts.values())
    plt.xlabel('bitstrings')
    plt.ylabel('counts')
    plt.savefig("result.png")

    return send_file("result.png", mimetype='image/png')

if __name__ == '__main__':
    app.run(host="localhost", port=33888)

```

Braket libraries for quantum computing

Classical wrapping service

Quantum algorithm

Figura 2.12: Algoritmo quantistico avvolto da un servizio classico

L’esempio però, non risolve le incompatibilità e integrazione tra sistemi. Infatti lo studio si è incentrato nel mostrare in particolare, come un problema computazionalmente difficile per i sistemi classici con un approccio quantistico, possa essere facilmente risolto. Si è scelto come esempio la fattorizzazione degli interi, ovvero, più precisamente, una particolare applicazione della successiva fattorizzazione dei primi. Come è noto, sebbene questo problema fondamentale della teoria dei numeri sia computazionalmente difficile, non si ritiene che appartenga alla classe dei problemi NP- difficili. Ciononostante è un problema che ricordiamo è stato utilizzato come presupposto di affidabilità per gli algoritmi crittografici, come il famoso algoritmo RSA.

Esistono molteplici proposte e algoritmi per la soluzione di questo problema, tra cui il più famoso è l’algoritmo di Shor di Nielsen et al. [34]. Tale algoritmo è normalmente descritto in termini di porte e circuiti quantistici, adatti allo sviluppo e all’esecuzione su macchine come il chip Q computing di IBM. Ma se si considerano altri approcci all’informatica quantistica, come l’Adiabatic Quantum Computing basato su concetti come il quantum annealing, non è possibile implementare direttamente l’algoritmo di Shor.

In questo contesto, si sottolinea la problematica di dover sempre adattare l’algoritmo sviluppato all’hardware della macchina sottostante, stia producendo problemi simili a quelli

della crisi del software degli anni Sessanta, in cui ogni algoritmo era progettato per ogni particolare hardware di calcolo, dovendo spesso ricreare gli algoritmi per ogni nuova macchina o addirittura per ogni nuovo incremento del problema. Un esempio di ciò è visibile quando si deve generare un nuovo circuito nell’algoritmo di Shor per la fattorizzazione dei primi. Anche se questo potrebbe essere fatto attraverso l’uso di algoritmi per generare automaticamente detti circuiti, per la grande maggioranza dei possibili utenti dell’informatica quantistica, la capacità di creare questi tipi di "meta-algoritmi" va al di là delle loro capacità, complicando l’espansione dell’uso dell’informatica quantistica al di fuori del campo specialistico. Pertanto, è necessario offrire soluzioni agli utenti non specializzati per l’utilizzo del calcolo quantistico, come il caso della distribuzione di servizi quantistici che consentono di nascondere la complessità agli utenti, fornendo solo gli endpoint di ingresso e restituendo i risultati dell’esecuzione.

Lo studio infine, termina con un’analisi dove viene chiarito che esistono alcune asperità, limitazioni e problemi che sorgono quando si prevede che un software quantistico venga fornito come servizio. Le limitazioni non riguardano l’impossibilità di costruire servizi quantistici, ma il fatto che, implementando questi servizi con le attuali tecnologie, si perdono i potenziali vantaggi del Service-Oriented Computing. Pertanto, nel caso d’uso descritto, ovvero il servizio di fattorizzazione dei numeri primi, non è possibile ricorrere agli algoritmi tradizionali, ma sfruttare i vantaggi offerti dal calcolo quantistico. Nell’informatica tradizionale orientata ai servizi, la sostituzione di un servizio con un altro, anche se distribuito su un’architettura hardware diversa, è perlopiù banale.

Tuttavia, cosa succede se è necessario sostituire un servizio classico con un servizio quantistico?

Seguendo un approccio simile a quello descritto nella definizione dei servizi classici, si mostra di seguito un modo per progettare l’architettura classica trasferendola nel mondo quantistico, al fine di mantenere gli attributi di qualità forniti dall’informatica orientata ai servizi.

Nella Figura 2.13 è illustrata una versione ibrida di un’implementazione di un servizio. Per implementare un livello di servizio quantistico, è necessario progettare la logica di business del servizio, come abbiamo fatto per quello classico. Ma successivamente, non abbiamo alcun supporto o meccanismo di standardizzazione per la definizione di API per gli endpoint quantistici. Allo stesso modo, non esistono strumenti per la generazione di codice a livello di servizi quantistici. In quest’ultimo, seguendo un’implementazione tradizionale, i servizi devono essere indipendenti dalla piattaforma. A tal fine, viene utilizzata un’API REST

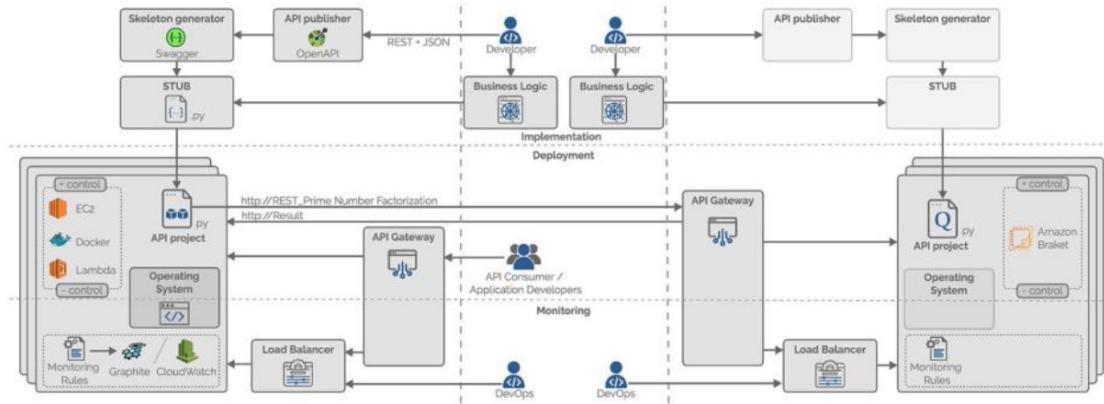


Figura 2.13: Versione ibrida architettura classica-quantistica di un’implementazione di un servizio

per garantire che gli stessi possano comunicare tra loro.

Allo stesso modo, viene utilizzato JSON come metodo per trasferire i dati tra i servizi, indipendentemente dai linguaggi di implementazione. Tuttavia, per un’implementazione quantistica, sono assenti sia protocolli di comunicazione, sia modi testati per comunicare servizi classici con servizi quantistici e inoltre, non esistono specifiche per definire un’API al fine di ottenere l’indipendenza dei servizi. Tutto ciò anche se altri ricercatori hanno iniziato a considerare questi problemi, come Cuomo et al. [35] che forniscono una panoramica delle principali sfide e dei problemi aperti che sorgono nella progettazione di un ecosistema di calcolo quantistico distribuito dal punto di vista dell’ingegneria del software.

Anche Rojo et al. [36] ha analizzato la possibilità di fornire un’implementazione sotto forma di microservizio quantistico ad un problema ben noto, come quello del commesso viaggiatore.

Un’altra caratteristica importante dell’informatica orientata ai servizi, è il disaccoppiamento. A livello di implementazione, il disaccoppiamento deve avvenire tra i servizi ed il linguaggio di programmazione in cui sono scritti. Nello sviluppo di un servizio classico, esso è fornito dalle specifiche OpenAPI e da strumenti di generazione del codice come Swagger, in grado di generare lo stub del servizio in diversi linguaggi di programmazione.

Tuttavia, nella progettazione di un servizio quantistico, non esistono specifiche e strumenti di generazione del codice per realizzare la struttura dello stub di un progetto API. Anche se si trovano alcuni lavori come quello di Dreher [37] che hanno sviluppato un prototipo di sistema basato su container che consente, allo sviluppatore, di prototipare, testare e implementare algoritmi quantistici con maggiore agilità e flessibilità.

Uno studio simile è quello svolto da **Enrique Moguel et al.**[33] e quello di **Manuel De**

Stefano et al. [38] preso come riferimento per la redazione del presente studio i cui dettagli saranno in seguito descritti nel capitolo progettazione.

Per concludere, è stato descritto come sia possibile creare un’architettura ibrida tenendo però in considerazione, tutte le limitazioni descritte precedentemente.

CAPITOLO 3

Progettazione e implementazione infrastruttura server-side

L'obiettivo del capitolo è ottenere una descrizione accurata di tutte le porzioni del sistema che si desidera sviluppare. La fase di progettazione è di fatto quella più importante nello sviluppo di un'applicazione in quanto può essere vista come la creazione di una soluzione a un determinato problema. È proprio per questo motivo che durante la progettazione bisogna tenere in considerazione i fattori, le potenzialità e i limiti relativi alle tecnologie che verranno adoperate per la realizzazione. In virtù di ciò, tutte queste scelte devono essere eseguite in modo corretto e accurato, per non incorrere in problemi durante la fase di sviluppo.

Prima di procedere alla descrizione dell'architettura che si vuole implementare, è importante sottolineare quanto lo scopo principale della nostra applicazione è quello di permettere a un utente di utilizzare gli algoritmi di programmazione quantistica senza incorrere nei problemi che sono stati descritti nel capitolo precedente. Essi riguardano soprattutto l'incompatibilità sia hardware che software tra i due ambienti, quello classico e quello quantistico. Il sistema pertanto, deve saper eseguire l'algoritmo richiesto, scegliendo sempre la migliore macchina quantistica disponibile in quel preciso momento, in base alle caratteristiche dell'algoritmo. Inoltre, il sistema dovrebbe essere implementato in modo tale che l'integrazione di altri provider quantistici possa essere eseguita senza troppe difficoltà.

3.1 Architettura di sistema proposta

Com'è stato anche accennato nel capitolo relativo ai related work, in cui abbiamo descritto i lavori più importanti che si sono occupati della progettazione di un sistema ibrido, il lavoro portato avanti da *De Stefano et al.* [38] è stato preso come punto di partenza per definire l'architettura del sistema attuale. Per questo in questo paragrafo si illustrerà l'idea di progetto descritta nel lavoro di *De Stefano et al.* [38] nel dettaglio e successivamente partendo da questo lavoro che cosa è stato effettivamente implementato.

La figura 3.1, mostra il diagramma di attività che illustra il processo ad alto livello svolto dal sistema. Sono descritti quattro componenti importanti: Il gateway API, L'orchestratore, il meta-modello e infine il provider quantistico per eseguire il calcolo. Di seguito il dettaglio di ogni componente.

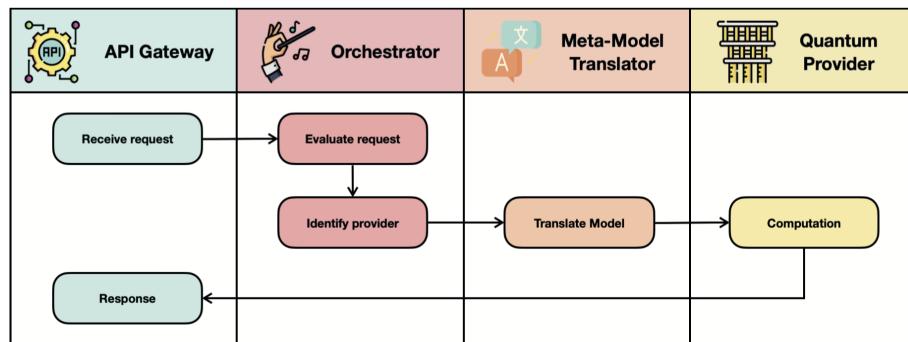


Figura 3.1: Diagramma attività

3.1.1 Gateway API

Il primo componente è il gateway API; è una classica facciata API accessibile tramite le tradizionali tecnologie API, come REST, GraphQL o RPC. Tramite questa interfaccia i componenti esterni possono interagire con il gateway API per richiedere l'esecuzione di un algoritmo quantistico. Il gateway API comunica con l'orchestrator successivamente analizzato.

Nella fase di analisi delle varie tecnologie per l'implementazione di un gateway API, sono stati considerate due tecnologie: REST e GraphQL. Da un certo punto di vista, GraphQL potrebbe essere considerato lo strumento più potente tra i due, dal momento che restituisce una descrizione completa e immediatamente comprensibile dei dati interni a una specifica API. Inoltre consente agli sviluppatori, di individuare più facilmente i dati richiesti per eseguire un'azione, permettendo inoltre di consolidarli da molteplici sorgenti all'interno

di un'unica richiesta. E' proprio quest'ultima la caratteristica fondamentale per il nostro progetto, poichè, riuscire a gestire diverse sorgenti come i vari provider quantistici in un'unica richiesta, rappresenta un fattore molto importante.

Inoltre, attraverso un meccanismo chiaro di segnalazione degli errori e un'interfaccia semplice e intuitiva, GraphQL si dimostra come giusta scelta per i nostri scopi.

3.1.2 Orchestrator

Il secondo componente è l'orchestrator come accennato precedentemente. Esso ha il compito di inoltrare la richiesta a un provider quantistico specifico secondo determinati parametri. L'orchestrator verifica la disponibilità del provider quantistico e effettua anche una verifica su quale provider quantistico è più efficiente far eseguire l'algoritmo. L'orchestrator comunica con il meta-modello, anche esso verrà descritto dettagliatamente in seguito.

Per quanto riguarda la seguente componente durante la fase di progettazione e analisi, si è pensato di utilizzare Kubernetes come orchestratore dal momento che l'intenzione è quella di creare l'intero sistema rendendolo disponibile a qualsiasi ambiente utilizzando Docker. Infatti Kubernetes è un meccanismo progettato in modo efficace che gestisce il ciclo di vita delle applicazioni containerizzate. Può essere definito come un sistema che distribuisce una funzione importante e ottimizza le modalità di lavoro delle applicazioni.

3.1.3 Meta-modello

Questa componente è utile poichè tutti gli algoritmi che sono resi disponibili dal sistema, sono memorizzati in esso sotto forma di un meta-modello. Ad esempio un modello indipendente dal framework che descrive l'algoritmo quantistico da eseguire. Pertanto, in base alle caratteristiche specifiche del meta-modello e alle capacità dei provider da eseguire, l'orchestrator inoltrerà la richiesta a un provider quantistico specifico. A tal fine, interagirà con il traduttore di meta-modelli per ottenere una rappresentazione dell'algoritmo compatibile con il provider selezionato.

3.1.4 Provider Quantistico

Infine l'ultima componente è il provider quantistico, responsabile dell'esecuzione. Completato il calcolo, l'output risultante verrà restituito all'API Gateway che soddisferà la richiesta.

I provider quantistici disponibili sono: Qiskit di IBM, Cirq di Google e Azure Quantum di Microsoft.

3.2 Architettura di sistema implementata

In questo paragrafo invece, si approfondirà l’architettura del sistema che è stato implementato. Quest’ultimo, avrà differenze rispetto a quello proposto; infatti è stato implementato solo il Gateway API che comunica direttamente con il provider quantistico di IBM. Alcune scelte architettoniche descritte precedentemente sono state comunque rispettate.

Nel dettaglio, la piattaforma sviluppata si basa sull’utilizzo della tecnologia di virtualizzazione Docker che permette di ospitare diverse applicazioni in grado di comunicare tra di loro. Si possono individuare, come mostrato in Figura 3.2, i quattro principali moduli costitutivi del sistema:

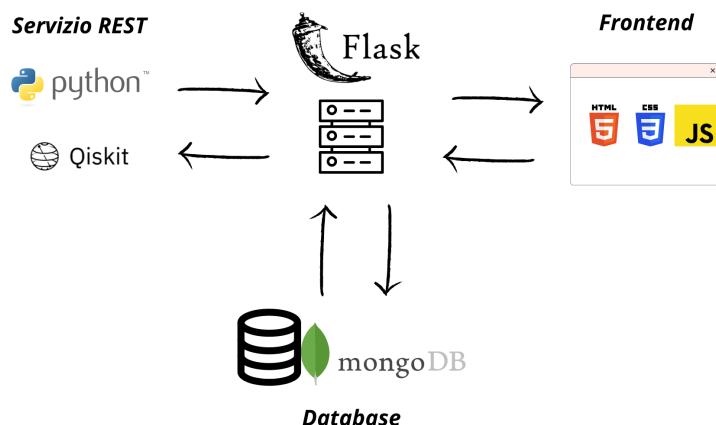


Figura 3.2: Architettura di sistema

Di seguito una descrizione più approfondita dei moduli che fanno riferimento all’infrastruttura *server-side*, il modulo del frontend invece sarà analizzato nel dettaglio nel capitolo 4 dove sarà descritta l’implementazione del prototipo web.

3.2.1 Servizio REST o Backend

Il primo modulo descritto è il servizio REST (Representational State Transfer) anche chiamato backend. REST è uno stile architettonico per sistemi distribuiti che semplifica l’implementazione del componente, riduce la complessità della semantica del connettore, migliora l’efficacia dell’ottimizzazione delle prestazioni e aumenta la scalabilità di componenti

server puri. Nel nostro sistema, l'intero modulo è sviluppato basandosi sul micro-framework Flask, che espone diverse API al frontend comunicando direttamente con il Database per effettuare operazioni di lettura e scrittura dei dati.

Il sistema è stato diviso in quattro classi fondamentali, come mostrato in Figura 3.3:

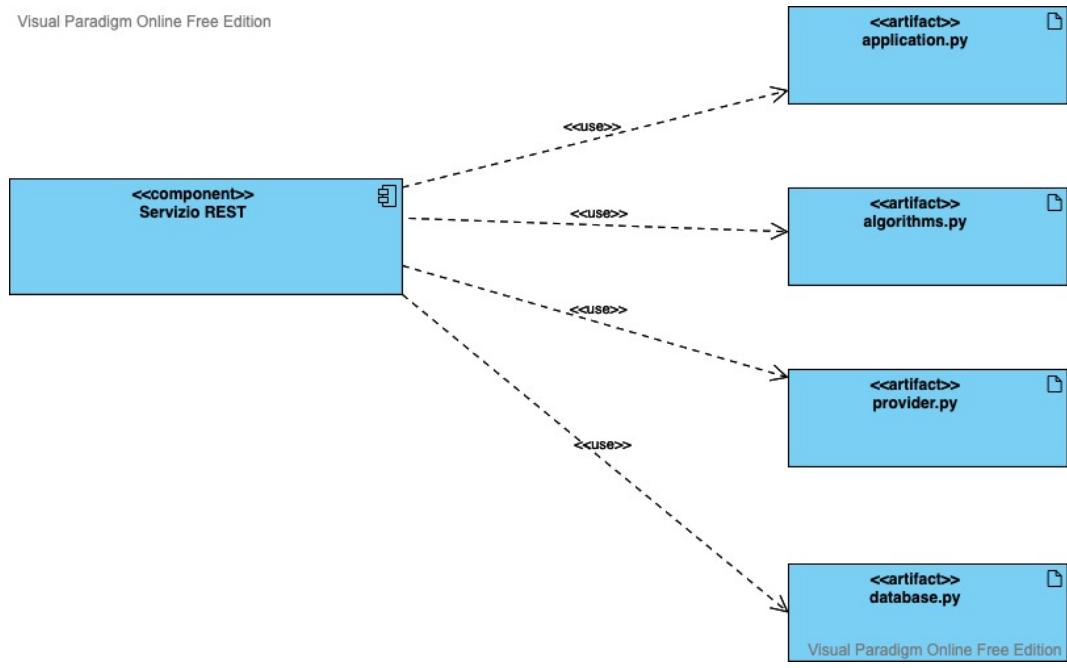


Figura 3.3: Modulo Servizio REST

- **Provider:** In questa classe viene gestita tutta la logica e le varie connessioni con l'SDK del provider scelto.
- **Algorithm:** In questa classe vengono gestiti tutti gli algoritmi, compresa l'esecuzione e l'inserimento.
- **Database:** In questa classe è definita la connessione con il database
- **Application:** Il core del nostro sistema. In esso sono definiti tutti gli endpoint resi disponibili dal sistema.

Come precedentemente sottolineato quando si è descritta la classe Provider, la comunicazione tra il backend e il provider avviene tramite QISKit (acronimo di Quantum Information Software Kit). È un Software Development Kit (SDK) per usufruire della IBM Q Experience e facilitare la produzione di codice OpenQASM. QISKit viene anche usato per creare programmi di computazione quantistica, compilare ed eseguirli su uno dei possibili backend. I backend si suddividono in dispositivi reali, simulatori online e simulatori locali.

Il modulo qiskit

Come già accennato, gli scopi principali di QISKit sono la produzione di codice OpenQASM e la connessione ad uno dei backend disponibili: entrambi i compiti vengono realizzati in Python, tramite chiamate a funzioni su opportuni oggetti resi disponibili dall'importazione del modulo qiskit.

Di seguito un esempio di come avviene la creazione di un circuito all'interno del sistema, e la sua esecuzione su di una macchina quantistica reale.

Un circuito di esempio

In questa sezione si illustreranno i semplici passaggi da includere nella realizzazione di un qualsiasi circuito in QISKit: la creazione del circuito e dei registri, la selezione del backend, la compilazione ed esecuzione del programma e la stampa dei risultati. Il circuito di esempio, si limita all'accensione del qubit meno significativo tramite l'applicazione di un gate opportuno:

```
# Importazione del modulo
from qiskit import QuantumProgram

# Creazione di QuantumProgram()
qp = QuantumProgram()

#Creazione di un registro quantistico e uno classico a n bit
n = 3
qr = qp.create_quantum_register("qr", n)
cr = qp.create_classical_register("cr", n)

#Creazione di un circuito che sfutta i registri appena creati
qc = qp.create_circuit("circuito_zero", [qr], [cr])
qc.x(qr[0])

# Misurazione del registro quantistico
qc.measure(qr, cr)

#Esecuzione del circuito sul simulatore locale
```

```

result = qp.execute(["circuito_zero"] ,
                    backend ="local_qasm_simulator",
shots=1024)

#Stampa del codice QASM
print(qp.get_qasm("circuito_zero"))
#Stampa dei risultati su stdout
print(result)
print(result.get_data("circuito_zero"))

```

Tutti i passaggi sono spiegati direttamente nel codice sotto forma di commenti.

L'output del programma contiene una prima parte in cui viene riportata la traduzione in linguaggio QASM, e una riga finale dove è possibile visualizzare il risultato di ciascuna delle 1024 iterazioni:

```

OPENQASM 2.0;
include "qelib1.inc";
qreg qr [3];
creg qc [3];
x qr [0];
measure qr [0] -> qc [0];
measure qr [1] -> qc [1];
measure qr [2] -> qc [2];
COMPLETED
{"counts": { '001': 1024}}

```

Esecuzione del codice su un processore quantistico reale

Per poter eseguire i codici sui processori quantistici reali forniti da IBM, occorre innanzitutto creare un account IBM Q experience. Successivamente, sarà necessario fare il login e generare un API Token, ovvero una stringa di caratteri esadecimali da riportare in un file con la seguente procedura:

```

APItoken = "inserire qua il token"

config = {

```

```
"url": " https://quantumexperience.ng.bluemix.net/api" ,  
}
```

Successivamente sarà possibile effettuare la ricerca della macchina quantistica disponibile per l'esecuzione del circuito. Di seguito, come avviene questo controllo all'interno del sistema.

```
#Viene caricato l'account attualmente in uso sul dispositivo  
IBMQ.load_account()  
  
#Selezine del provider  
provider = IBMQ.get_provider(hub='ibm-q')  
  
#Scelta backend libero  
per il numero di qubit indicati  
backend = least_busy(provider.backends(filters=lambda x:  
x.configuration().n_qubits >= num_qbit + 1 \  
and not x.configuration().simulator \  
and x.status().operational == True))  
device = provider.get_backend(str(backend))  
  
return device
```

Endpoint

Per soddisfare i requisiti descritti e analizzati precedentemente, è stata implementata una classe chiamata *application.py* dove, al suo interno, sono presenti gli endpoint impiegati dal sistema per esporre le proprie funzionalità rendendole utilizzabili.

Nella tabella sottostante sono mostrati gli endpoint disponibili.

Tabella 3.1: Tabella Endpoint

Type	Path	Response
POST	/CustomAPIquantum/(name_algorithm,input,num_qbit)	200,400
POST	/quantumAPI/(type_selected,log_size,case_selected,/NumQubitsMaschine, num_qubits,bynary_string,num_qubits,num)	200,400
POST	/insertAlgorithm/(name_selected,algorithm,description)	200,400
GET	/getInfoAlgorithm/(name_algorithm)	200,400

La tabella 3.3 riporta gli endpoint che sono esposti dall'API per gestire l'interazione con gli eventi. Successivamente sarà mostrata la loro implementazione.

1. */CustomAPIquantum*: si occupa dell'esecuzione degli algoritmi personali presenti nel sistema, i tre parametri che devono essere trasmessi insieme, sono il nome dell'algoritmo, il numero di qubit che si vogliono utilizzare per l'esecuzione dell'algoritmo e gli input utili all'esecuzione dell'algoritmo;
2. */quantumAPI*: si occupa dell'esecuzione degli algoritmi più noti che sono preventivamente caricati nel sistema. I parametri sono diversi e variano a seconda dell'algoritmo scelto;
3. */insertAlgorithm*: si occupa dell'inserimento di un nuovo algoritmo all'interno del sistema. I due parametri trasmessi sono: il nome dell'algoritmo, l'algoritmo in formato QASM e la descrizione del suo funzionamento oltre che alla definizione degli input richiesti;
4. */getInfoAlgorithm*: restituisce tutte le informazioni dell'algoritmo come il risultato, il circuito generato e gli input trasmessi in fase di esecuzione. Il parametro richiesto è il nome dell'algoritmo.

3.2.2 Implementazione /CustomAPIquantum

Di seguito l'implementazione dell'endpoint */CustomAPIquantum*. Nella prima parte è prevista la raccolta degli input dell'utente al fine di creare un dizionario chiave-valore. Questi valori verranno poi gestiti dalla classe *PersonalAlgorithms* tramite il metodo *execution*. Dopo l'esecuzione sarà salvato inizialmente il risultato nella variabile *algoritmopersonal* per poi essere restituita tramite il *return*.

```
@app.route('/CustomAPIquantum', methods=['POST'])
def personaQuantumAPI():
    # Costruzione dizionario con i valori inseriti dall'utente
    value_form = {
        "algo": request.form.get('name_algorithm', type=str),
        "input": request.form.get('input_algorithm')
    }
    """numero qbit per verificare quale macchina quantistica libera"""
    num_qbit = request.form.get('num_qbit', type=int)
```

```

strategia = PersonalAlgorithms()
algoritmopersonal = PersonalAlgorithms.execution(strategia,
                                                    value_form, num_qbit, db, qiskit)

return algoritmopersonal

```

3.2.3 Implementazione /quantumAPI

Anche in questo caso sarà illustrata l'implementazione dell'endpoint */quantumAPI*. Nella prima parte è prevista la raccolta degli input dell'utente al fine di creare un dizionario chiave-valore per poi essere gestito dalla classe specifica dell'algoritmo richiesto. Da segnalare la presenza di un controllo sulla variabile *switch*, stanziata con il nome dell'algoritmo scelto dall'utente. Dopo l'esecuzione, la pagina *result.html* verrà renderizzata con l'esito dell'operazione.

```

@app.route('/quantumAPI', methods=['POST'])
def quantumAPI():
    # Nome dell'algoritmo
    switch = request.form.get('type_selected', type=str)
    # Numero di qubits della macchina reale
    # per l'esecuzione dell'algoritmo
    num_qbitMaschine = request.form.get('NumQubitsMaschine', type=int)
    # Costruzione dizionario con i valori inseriti dall'utente
    value_form = {
        "log_size": request.form.get('log_size', type=int),
        "case_selected": request.form.get('case_selected', type=str),
        "num_qubits": request.form.get('num_qubits', type=int),
        "bynary_string": request.form.get('bynary_string', type=str),
        "n_count": request.form.get('num_qubits', type=int),
        "a": request.form.get('num', type=int)
    }

    if switch == 'DeutschJozsa':

```



```
elif switch == 'Shors':  
  
    # Esecuzione algoritmo tramite la sua classe specifica  
    strategia = Shors()  
    shors = Shors.execution(strategia,  
                            value_form, num_qbitMaschine, db, qiskit)  
  
    # Renderizzazione pagina result.html per esito operazione  
    return render_template('result.html',  
                          message="Shor's algorithm executed correctly",  
                          circuit=shors, page='execution')  
  
elif switch == 'Grovers':  
  
    # Esecuzione algoritmo tramite la sua classe specifica  
    strategia = Grovers()  
    grovers = Grovers.execution(strategia,  
                                value_form, num_qbitMaschine, db, qiskit)  
  
    # Renderizzazione pagina result.html per esito operazione  
    return render_template('result.html',  
                          message="Grover's algorithm executed correctly",  
                          circuit=grovers, page='execution')  
  
elif switch == 'WalkSerachHypercube':  
  
    # Esecuzione algoritmo tramite la sua classe specifica  
    strategia = WalkSerachHypercube()  
    walkHyper = WalkSerachHypercube.execution(strategia,  
                                              value_form, num_qbitMaschine, db, qiskit)  
  
    # Renderizzazione pagina result.html per esito operazione  
    return render_template('result.html',  
                          message='Walk Serach Hypercube  
algorithm executed correctly',
```

```

    circuit=walkHyper, page='execution')

else :

    # In caso di errore renderizzazione
    pagina result.html per esito operazione
    return render_template('result.html', message='Error!',
                           circuit='error.png', page='execution')

```

3.2.4 Implementazione /insertAlgorithm

Sarà ora descritta l'implementazione dell'endpoint */insertAlgorithm*, dove è prevista la raccolta degli input dell'utente al fine di creare un dizionario chiave-valore. Quest'ultimi saranno poi gestiti dalla classe *PersonalAlgorithms* tramite il metodo *insert*, che si occuperà di salvare l'algoritmo nel database. Al termine dell'inserimento, verrà renderizzata la pagina *result.html* con l'esito dell'operazione.

```

@app.route('/insertAlgorithm', methods=['GET', 'POST'])
def insertAlgorithm():

    #Costruzione dizionario con i valori inseriti dall'utente
    value_form = {
        "name": request.form.get('name_selected', type=str),
        "algoritmo": request.form.get('algorithm', type=str),
        "descrizione": request.form.get('description', type=str)
    }

    # Inserimento algoritmo tramite metodo specifico
    strategia = PersonalAlgorithms()
    insert = PersonalAlgorithms.insert(strategia, value_form, db)

    # Renderizzazione pagina result.html per esito operazione
    return render_template('result.html',
                           message='Algorithm upload successfully!',
                           circuit='Success.png', page='insert')

```

3.2.5 Implementazione /getInfoAlgorithm

Infine l'implementazione dell'endpoint /getInfoAlgorithm, dove, inizialmente si effettua la raccolta degli input dell'utente, creando un dizionario chiave-valore, per poi essere gestito dalla classe PersonalAlgorithms tramite il metodo getInfo(). Il metodo si occupa quindi, di restituire tutte le informazioni presenti nel sistema sull'algoritmo precedentemente richiesto. Dopo l'operazione, le informazioni saranno memorizzate nella variabile info in formato JSON e restituite tramite il *return*.

```
@app.route('/getInfoAlgorithm', methods=[ 'GET' , 'POST' ])
def InfoAlgorithm():

    #Costruzione dizionario con i valori inseriti dall'utente
    value_form = {
        "name": request.form.get('name_selected', type=str),
    }

    # Info algoritmo tramite metodo specifico
    strategia = PersonalAlgorithms()
    info = PersonalAlgorithms.getInfo(strategia, value_form, db)

    return info, 200
```

3.2.6 Database

Il secondo modulo che andremo a descrivere è il database. Il database è una delle componenti più importanti di ogni sistema poichè permette il salvataggio dei dati per poi effettuare qualsiasi tipo di elaborazione. Dopo l'analisi del problema e dei requisiti, sono state analizzate le varie alternative su quale tipo di database potesse essere idoneo al nostro progetto. La scelta è ricaduta su **MongoDB** database non relazionale in quanto il nostro sistema avrà l'obiettivo di raggiungere performance elevate. I database non relazionali presentano infatti le performance migliori per i tempi di risposta grazie alla possibilità di contenere tutte le informazioni necessarie in un'unica struttura chiamata JSON. Altro vantaggio deriva dalla loro semplicità: è proprio questa caratteristica che permette di scalare in orizzontale in maniera efficiente. Infine, scegliendo un database adatto alla mappatura più diretta alle classi d'oggetti del proprio applicativo, si possono ridurre notevolmente i tempi dedicati

allo sviluppo del metodo di scambio dati tra il database e l'applicativo stesso (il cosiddetto object-relational mapping, invece necessario in presenza di database relazionali). La loro semplicità comporta però anche degli svantaggi: non essendoci i controlli fondamentali sull'integrità dei dati, il compito ricade totalmente sull'applicativo che dialoga col database. Altra pecca è la mancanza di uno standard universale (come può essere l'SQL). Ogni database ha, infatti, le proprie API e il suo metodo di storing e di accesso ai dati. Nonostante ciò, i database non relazionali, pur non essendo la soluzione definitiva al problema del salvataggio e recupero dei dati, visto l'avvento di Social Network e del cloud, risultano la miglior soluzione in contesti in cui ogni giorno bisogna salvare un numero elevatissimo di dati e solo grazie alla scalabilità orizzontale dei database NoSQL e quindi all'aggiunta di server facili da gestire, è possibile ottenere delle performance soddisfacenti, di gran lunga migliori rispetto ai soliti database relazionali. Esempi di applicazioni che utilizzano questo modello sono infatti Twitter, Facebook e Amazon.

Progettazione Database

Dopo aver illustrato i pro ed i contro dei database non relazionali, scegliendo come migliore soluzione ai nostri problemi, l'utilizzo di **MongoDB**. In questo paragrafo invece, ci concentreremo sulla progettazione del database, che sia in grado di aver performance elevate a livello di caricamento e reperimento dati. Oltre alla natura prestazionale, vi è un motivo prettamente strutturale per aver scelto un database non relazionale ovvero l'entità che viene salvata sul db è solo una, cioè l'algoritmo con i relativi input inseriti dall'utente, pertanto non è necessario un database relazionale che metta in relazione più entità. La suddetta entità è strutturata come da tabella 3.3.

Tabella 3.2: Tabella Algorithm

Nome Campo	Tipo	Descrizione
_id	ObjectID	Id univoco che differenzia ogni entità del db
nome	String	Nome dell'algoritmo
log_size	Int	Input dell'utente utile all'esecuzione dell'algoritmo
data	Datetime	Data di inserimento nel db dell'algoritmo
case	String	Input dell'utente utile all'esecuzione dell'algoritmo
binary_string	String	Input dell'utente utile all'esecuzione dell'algoritmo
num_qubits	Int	Input dell'utente utile all'esecuzione dell'algoritmo
description	String	Input dell'utente che descrive l'algoritmo inserito
file	Bindata	In questo file viene memorizzato l'algoritmo
insert	Bool	Questo booleano viene posto a true quando l'algoritmo che è stato salvato è stato inserito da un'utente
result	Int	Risultato dell'esecuzione dell'algoritmo

Di seguito, l'implementazione nel dettaglio della classe database con il compito di instaurare la connessione con il database restituendo l'istanza da utilizzare per le varie operazioni di lettura e scrittura.

3.2.7 Codice sorgente classe database

Nel codice descritto successivamente, viene instaurata la connessione con il database. Per raggiungere tale risultato, si utilizza la libreria esterna *pymongo*. Nel metodo *connect()* viene creata una connessione con il database tramite **URL** e in seguito, viene stanziata la variabile *mydb* con il database **QuantumAPI**, la variabile *mycol* con la collection di nostro interesse ovvero **algorithm**. Il tutto solo dopo una stampa informativa sulla corretta connessione. Infine viene restituita l'istanza di questa collection per effettuare le operazioni di scrittura e lettura.

```

class Database:
    connection = pymongo.MongoClient()

    @staticmethod
    def connect():
        try:

```

```
conn = MongoClient('mongodb://localhost:27017/')

print('Connected successfully !!!')

mydb = conn['QuantumAPI']

mycol = mydb['algorithm']

return mycol

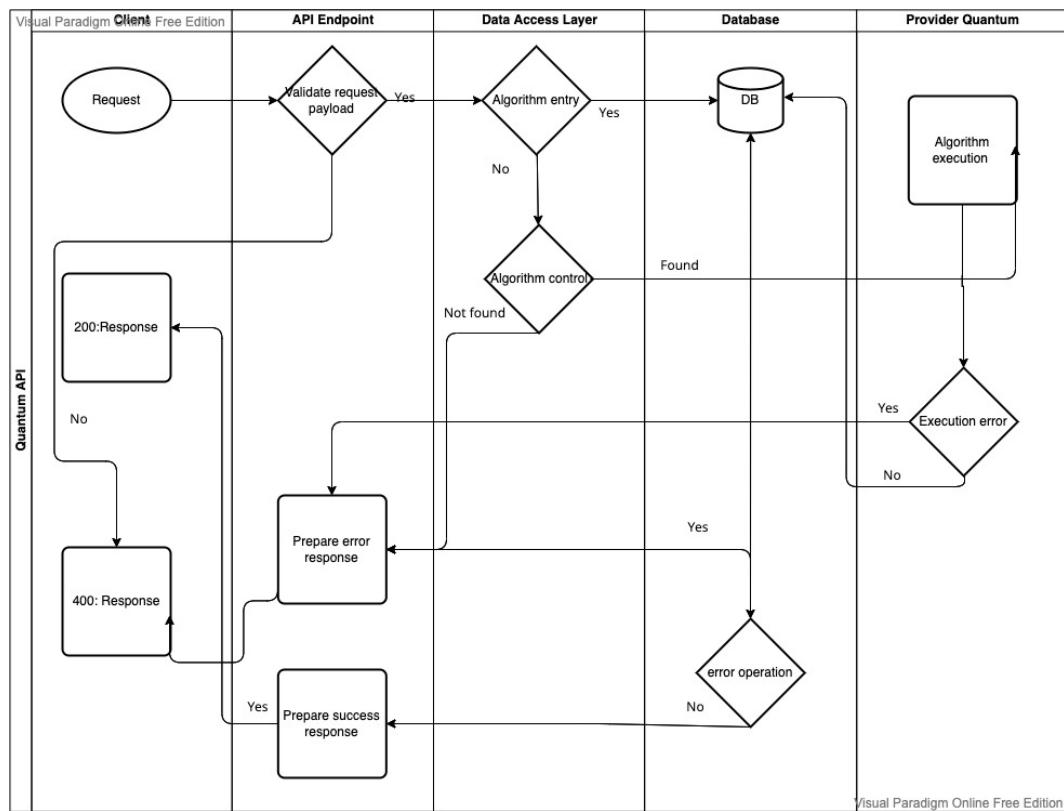
except:

    print('Could not connect to MongoDB')
```

3.3 **Swimline Diagram**

In questa sezione si vuole porre l'attenzione sull'interazione di tutte le componenti del sistema. A tal fine si è deciso di sfruttare lo swimline, tipologia di diagramma di flusso che delinea chi fa cosa in un processo. Utilizzando la metafora delle corsie in una piscina, un diagramma swimlane (che in inglese significa appunto corsia) fornisce chiarezza e responsabilità, inserendo le fasi del processo all'interno delle "swimlane" orizzontali o verticali di un sistema. Mostra connessioni, comunicazioni e passaggi tra queste corsie e può servire ad evidenziare sprechi, ridondanze ed inefficienze all'interno di un processo.

Di seguito il diagramma del sistema in Figura 3.4:

**Figura 3.4:** Swimline Diagram

Si evince, dall'analisi del diagramma, che sono cinque le swimline e rappresentano le componenti fondamentali del nostro sistema.

Da sinistra a destra le componenti sono:

1. **Client**: interagisce direttamente con l'utilizzatore del sistema, infatti può inviare una richiesta (caricare algoritmo oppure esecuzione di un algoritmo) e ricevere anche due tipologie di risposte. La risposta di tipo 200 vuole comunicare che le operazioni richieste sono state eseguite correttamente, mentre quella di tipo 400 segnala un eventuale problema. Questa componente comunica direttamente con API Endpoint.
2. **API Endpoint**: ha un ruolo fondamentale nel sistema perchè si occupa di ricevere le richieste, elaborarle e restituire una risposta. Il primo passo, quando si riceve una richiesta, è la validazione, ovvero comprendere se la richiesta è corretta e sono stati rispettati tutti i vincoli (come ad esempio se sono stati trasmessi tutti i valori utili per l'elaborazione). Dopo la validazione, API si occupa di interagire con le altre componenti per poi preparare una risposta e inviarla al client.
3. **Data Access Layer**: fornisce un accesso semplificato ai dati archiviati in un database.

4. *Database*: è il database vero e proprio, quindi si occupa della memorizzazione degli algoritmi.
5. *Provider Quantum*: rappresenta il provider quantistico scelto per l'esecuzione dell'algoritmo.

Dopo aver descritto le componenti in dettaglio cerchiamo di capire come avviene il flusso degli eventi senza entrare troppo nel dettaglio. La prima fase è la generazione della richiesta da parte del client, che successivamente viene ricevuta dall'API che, dopo averla validata, comprende quale tipo di attività debba essere eseguita. Se la validazione ha esito positivo, la richiesta passa al data access layer, che si occupa di instradarla in funzione della tipologia di attività richiesta dal client.

Se quest'ultima comprende l'inserimento di un algoritmo, i dati saranno elaborati e in seguito salvati nel database, se invece la richiesta comprende l'esecuzione di un algoritmo, i dati saranno preparati e inviati al provider quantistico che eseguirà l'algoritmo.

A esito positivo, i risultati verranno prima salvati nel database e in seguito restituiti al client. A esito negativo invece, i dati non saranno salvati nel database e verrà restituito un errore.

3.4 Use cases

Il caso d'uso, in informatica, è una tecnica utilizzata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua la raccolta dei requisiti al fine di produrre software di qualità.

Essa consiste nel valutare ogni requisito focalizzandosi sugli attori che interagiscono col sistema, valutandone le varie interazioni. In UML sono rappresentati dagli Use Case Diagram.

Il documento dei casi d'uso individua e descrive gli scenari elementari di utilizzo del sistema da parte degli attori che si interfacciano con esso.

Di seguito saranno descritti due casi d'uso per ogni funzionalità offerta dal sistema.

Il primo, esaminato in questo capitolo, tratta l'utilizzo del sistema sfruttando l'API; il secondo, nel capitolo 4, invece l'interfaccia utente, che renderà più semplice l'inserimento e il testing degli algoritmi. In seguito inoltre, sarà descritto un'esempio pratico di tutte le azioni riportate nel caso d'uso.

3.4.1 Use case (utilizzo tramite API)

Nel seguente caso d'uso, un esempio di utilizzo del sistema sfruttando le API messe a disposizione. Un'ingegnere del software ha necessità di utilizzare l'Algoritmo di Grover per un progetto aziendale. Esso è noto nella programmazione quantistica perchè risolve il problema di ricerca in un database indifferenziato di N elementi in $O(N^{1/2})$ tempo usando $O(\log N)$ come spazio di memorizzazione.

Il sistema proposto, oltre ad un'interfaccia grafica, presenta alcuni endpoint che da utilizzare per varie operazioni, e uno di questi endpoint consente di utilizzare gli algoritmi quantistici più noti. Quindi, dopo aver letto la documentazione disponibile del sistema, si apprende che per i suoi scopi, deve effettuare una chiamata al seguente endpoint */quantumAPI*.

Successivamente, nel codice sorgente effettua una chiamata all'endpoint sopra citato, utilizzando come parametri, il nome dell'algoritmo (in questo caso Grover) e ulteriori input di cui lo stesso ha bisogno. Fatto ciò, si utilizzerà il risultato dell'esecuzione dell'algoritmo per il progetto aziendale.

3.5 Esempio utilizzo tramite API

In questa paragrafo verrà illustrato un utilizzo del sistema tramite API. Per far ciò prenderemo come esempio l'algoritmo di Simons che è uno degli algoritmi pre-caricati all'interno del sistema, poichè si tratta di un algoritmo noto nella programmazione quantistica. Inizialmente si descriverà l'algoritmo e in seguito, sarà mostrata la sua implementazione. L'algoritmo di Simon, è stato il primo quantistico a mostrare una velocità esponenziale rispetto al miglior algoritmo classico nella risoluzione di un problema specifico. È stato di ispirazione per gli altri algoritmi quantistici basati sulla trasformata di Fourier quantistica, ovvero una trasformazione lineare su qubit. Questa caratteristica viene anche utilizzata nell'algoritmo quantistico più famoso: L'algoritmo di fattorizzazione di Shor.

3.5.1 Il problema di Simons

Viene assegnata una funzione blackbox sconosciuta f , che gode delle proprietà uno-a-uno (1:1) o due-a-uno (2:1), di seguito analizzate singolarmente:

1. **uno-a-uno**: mappa esattamente un output univoco per ogni input. Un esempio di funzione che accetta quattro input:

$$f(1) \rightarrow 1, \quad f(2) \rightarrow 2, \quad f(3) \rightarrow 3, \quad f(4) \rightarrow 4$$

Figura 3.5: Formula uno-a-uno (Simons)

2. **due-a-uno**: mappa esattamente due input su ogni output univoco. Un esempio di funzione che accetta quattro input:

$$f(1) \rightarrow 1, \quad f(2) \rightarrow 2, \quad f(3) \rightarrow 1, \quad f(4) \rightarrow 2$$

Figura 3.6: Formula due-a-uno (Simons)

Questa mappatura due-a-uno si basa su di una stringa di bit nascosta, b , dove:

$$\begin{aligned} \text{dato } X_1, X_2 : \quad & f(X_1) = f(X_2) \\ \text{è garantito : } & X_1 \oplus X_2 = b \end{aligned}$$

Figura 3.7: Formula (Simons)

Data questa scatola nera f , quanto velocemente possiamo determinare se f è uno-a-uno o due-a-uno?

Se f risulta essere due-a-uno, quanto velocemente possiamo determinare b ?

A quanto pare, entrambi i casi si riducono allo stesso problema, ovvero trovare b , dove una stringa di bit di $b=000\dots$ rappresenta l'uno-a-uno f .

3.5.2 L'algoritmo di Simons

3.5.3 Soluzione classica

La soluzione classica per sapere cosa sia b con certezza per una determinata f , è quella di controllare

$$2^{n-1} + 1$$

ingressi, dove n è il numero di bit nell'ingresso. Ciò significa controllare poco più della metà di tutti i possibili input fino a trovare due casi dello stesso output. Naturalmente ci saranno casi in cui è possibile risolvere il problema con soli due tentativi, anche se, solitamente, la complessità cresce in maniera esponenziale con n . Per questo motivo l'algoritmo presenta un'implementazione quantistica che riduce la complessità.

3.5.4 Soluzione quantistica

La soluzione quantistica che implementa l'algoritmo di Simon è rappresentata dal circuito mostrato in Figura 3.8:

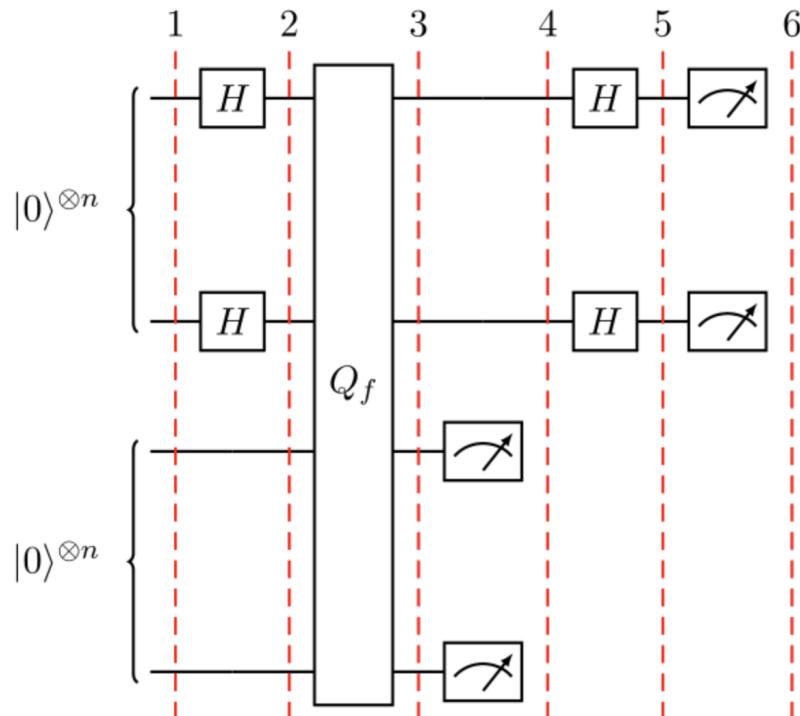


Figura 3.8: Circuito implementazione algoritmo Simons

Di seguito sarà mostrata l'implementazione quantistica del seguente algoritmo.

3.5.5 Implementazione algoritmo Simons

```
from qiskit import QuantumCircuit
from qiskit_textbook.tools import simon_oracle

def s_algorithm(b, n):
```

```

n = len(b)
simon_circuit = QuantumCircuit(n * 2, n)

# Apply Hadamard gates before querying the oracle
simon_circuit.h(range(n))

# Apply barrier for visual separation
simon_circuit.barrier()

simon_circuit += simon_oracle(b)

# Apply barrier for visual separation
simon_circuit.barrier()

# Apply Hadamard gates to the input register
simon_circuit.h(range(n))

# Measure qubits
simon_circuit.measure(range(n), range(n))

return simon_circuit

```

Com'è possibile notare dal codice sopra descritto, la funzione principale è la *s_algorithm* che accetta due valori, *b* rappresenta la stringa binaria, mentre *n* il numero di qubits.

La seguente funzione, viene richiamata nella classe Simon, dove successivamente sarà mostrata la sua implementazione.

3.5.6 Implementazione classe Simons

```

class Simons(AlgorithmsStrategy):

    def execution(self, dict_value, num_qbit: int, db, provider):
        # Getting the current date and time

        num_qubits = dict_value["num_qubits"]

```

```
bynary_string = dict_value["bynary_string"]

S_circuit = s_algorithm(bynary_string, num_qubits)

"""verifica macchina quantistica"""
macchina_free = provider.connect(num_qbit)

"""esecuzione circuito"""
result = execute(S_circuit, backend=macchina_free).result()
final = result.get_counts(S_circuit)

# Getting the current date and time

dt = datetime.now()
cir = S_circuit.draw(output='mpl')
pathImg = './ static/AlgoritmoSimons_ ' \
          + dt.strftime(' %m-%d-%Y_%H:%M%S') + '.png'
cir.savefig(pathImg)
S_circuit.qasm(formatted=True, filename='AlgoritmoSimons.qasm')

with open('AlgoritmoSimons.qasm', 'rb') as f:
    encoded = Binary(f.read())

db.insert_one({
    'nome': 'Algoritmo Simons',
    'num_qubits': num_qubits,
    'data': dt,
    'bynary_string': bynary_string,
    'insert': False,
    'file': encoded,
    'result': final,
})
os.remove('AlgoritmoSimons.qasm')
pathImgNew = pathImg.replace('./ static/', '')
```

```
return pathImgNew
```

Com'è possibile notare, alla sesta riga viene richiamata la funzione che abbiamo mostrato precedentemente. La funzione *s_algorithm* restituisce un circuito, in seguito verrà identificata una macchina quantistica reale libera in quel momento, una volta identificata viene eseguito il circuito e il risultato viene salvato nella variabile *final*. In seguito, ci sono altre operazioni come la rappresentazione grafica del circuito generato e il salvataggio sul database di tutte le informazioni riguardanti la suddetta esecuzione dell'algoritmo, quindi risultato, input, immagine del circuito e la data di esecuzione.

Detto ciò se un utente volesse utilizzare il seguente algoritmo, deve effettuare una chiamata all'endpoint */quantumAPI* che si occupa di gestire tutti gli algoritmi noti presenti nel sistema. Di seguito sarà mostrata l'implementazione del seguente endpoint con un focus sull'algoritmo di Simons.

3.5.7 Implementazione endpoint /quantumAPI

```
@app.route('/quantumAPI', methods=['POST'])
def quantumAPI():

    # Nome dell'algoritmo
    switch = request.form.get('type_selected', type=str)

    # Numero di qubits della macchina reale per
    # l'esecuzione dell'algoritmo
    num_qbitMaschine = request.form.get('NumQubitsMaschine',
                                         type=int)

    # Costruzione dizionario con i valori inseriti dall'utente
    value_form = {
        "log_size": request.form.get('log_size', type=int),
        "case_selected": request.form.get('case_selected', type=str),
        "num_qubits": request.form.get('num_qubits', type=int),
        "bynary_string": request.form.get('bynary_string', type=str),
        "n_count": request.form.get('num_qubits', type=int),
        "a": request.form.get('num', type=int)
    }

    .
```

```

.
.

elif switch == 'Simons':

    # Esecuzione algoritmo tramite la sua classe specifica
    strategia = Simons()
    simons = Simons.execution(strategia,
                               value_form, num_qbitMaschine, db, qiskit)

    # Renderizzazione pagina result.html per esito operazione
    return render_template('result.html',
                           message='Simons algorithm executed correctly',
                           circuit=simons, page='execution')

.
.
```

Nel succitato codice, è possibile notare come nella prima parte ci sia la gestione degli input inseriti dall’utente come, ad esempio: il nome dell’algoritmo selezionato, il numero di qubits che deve possedere la macchina quantistica per l’esecuzione, nonchè gli input utili all’algoritmo. Nel caso in esame, gli input di nostro interesse sono *num_qubits* e *binary_string*. Successivamente, dopo il controllo sul nome, viene richiamata la funzione *execution* mostrata precedentemente.

Sarà ora illustrato un semplice programma che, dopo aver inserito i due valori necessari all’algoritmo, per effettuare il calcolo, chiamerà l’endpoint descritto precedentemente.

3.5.8 Programma semplice per l’utilizzo dell’algoritmo di Simons

```

import requests

if __name__ == "__main__":
    # stringa binaria
    binary = "110"
    # numero qubits macchina quantistica
    num_qubitsMaschine = 6

```

```
# numero qubits per algoritmo
num_qubits = 3

input = {
    "num_qubits" : num_qubits,
    "bynary_string": binary,
    "type_selected": "Simons".
    "NumQubitsMaschine": num_qubitsMaschine
}

result = requests.post("https://localhost:5002//quantumAPI",
                      data=input)

print("Il risultato dopo l'esecuzione dell'algoritmo di Simons:")
print(result)
```

Per quanto precedentemente illustrato, si è dimostrato quanto sia semplice lavorare con il sistema proposto al fine di utilizzare qualsiasi algoritmo noto nella programmazione quantistica.

CAPITOLO 4

Progettazione e implementazione applicazione web

Nella presente capitolo si descriverà l'implementazione del prototipo web iniziando con la definizione dei requisiti funzionali che dovrà avere. Sarà inoltre descritto il modulo del frontend con la sua struttura, lo use case relativo all'inserimento e il testing dell'algoritmo. Infine un esempio pratico delle azioni descritte nello use case, evidenziando il flusso di navigazione da una schermata all'altra dell'applicazione.

4.1 Requisiti funzionali

Un requisito funzionale, nell'ambito dell'Ingegneria del Software, è quello che definisce una funzione di un sistema di uno o più dei suoi componenti, specificandone la tipologia degli ingressi e delle uscite, nonché il comportamento.

Tabella 4.1: Requisiti Funzionali

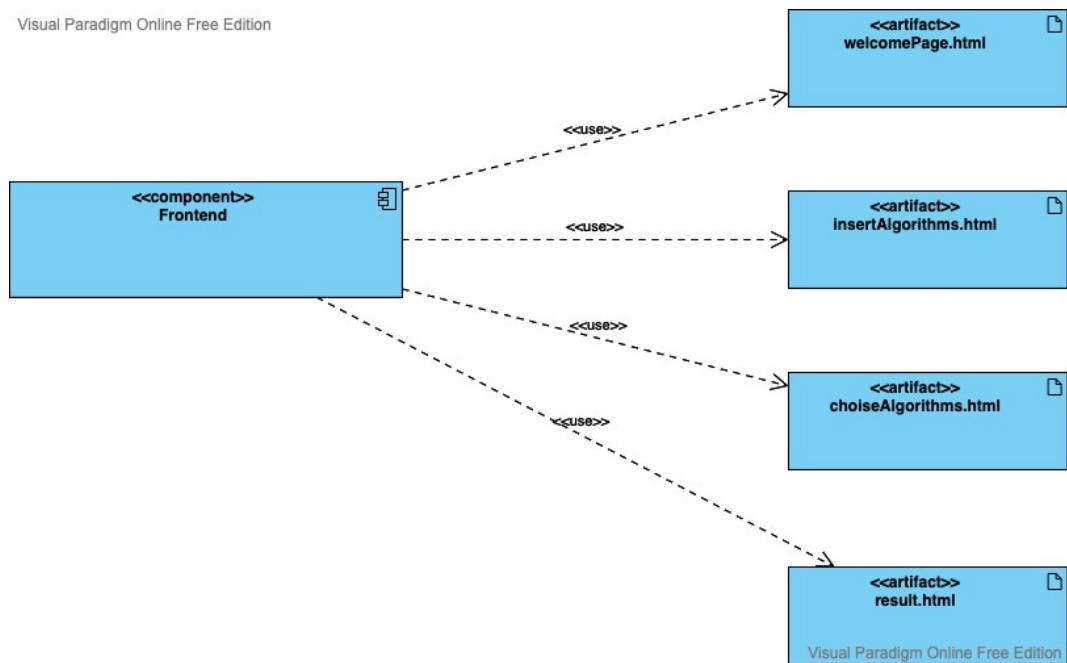
ID	Descrizione
RF1	Il sistema deve poter visualizzare il circuito generato dall'esecuzione dell'algoritmo per testare la sua corretta costruzione
RF2	Il sistema deve prevedere un'interfaccia nella quale sia possibile inserire i propri algoritmi personalizzati
RF3	Il sistema deve poter visualizzare il risultato dell'esecuzione del circuito.

4.2 Frontend

Nel capitolo 3, sono state illustrate le componenti *server-side*. In questo paragrafo invece, si descriverà la componente mancante ovvero il frontend. Il frontend è costituito da una Web Application sotto forma di Multi Page Application che consente l'interazione tra lo sviluppatore ed il sistema. Un'applicazione Web è fruibile attraverso internet da un browser Web. In appena un decennio, il Web si è evoluto dall'essere un repository di pagine utilizzate principalmente per l'accesso a informazioni statiche, per lo più scientifiche, a una potente piattaforma per lo sviluppo e la distribuzione di applicazioni. Per quanto riguarda la scelta del linguaggio di programmazione da utilizzare non è stato scelto uno specifico, si sono costruite delle pagine html in seguito renderizzate da Flask. La prima fase è stata quella di definire un template per l'interfaccia semplice da comprendere dall'utente agevolandogli l'operatività. Ricapitolando, si è utilizzato HTML5 per avere un design responsive. Tale scelta permette la visualizzazione su qualunque dispositivo con un browser, sviluppando un unico sito Web per computer, tablet, smartphone con visualizzazione sempre ottimale. Il modulo di frontend è stato implementato al solo scopo di poter semplificare il caricamento di algoritmi personalizzati e inoltre è possibile verificare la corretta generazione del circuito. Questa precisazione è fondamentale poichè il sistema è utilizzabile anche senza l'utilizzo dell'interfaccia grafica.

4.2.1 Progettazione Frontend

Nella Figura 4.1 è possibile osservare il modulo frontend con le relative pagine html di cui fanno parte.

**Figura 4.1:** Modulo Frontend

Le pagine html presenti sono:

- *welcomePage.html*: questa è la pagina di benvenuto dove l’utente può scegliere quale azione intraprendere;
- *insertAlgorithms.html*: è la pagina che si occupa dell’inserimento di un’algoritmo personalizzato;
- *chooseAlgorithms.html*: si occupa di far scegliere all’utente quale algoritmo utilizzare dando la possibilità anche di inserire gli input e infine eseguire l’algoritmo;
- *result.html*: in questa pagina viene visualizzato l’esito dell’operazione sia per quanto riguarda l’inserimento e sia per quanto riguarda l’esecuzione.

4.3 Use case (applicazione web)

Un’ingegnere del software è riuscito a sviluppare un algoritmo quantistico personale risolvere il Problema di Simon in maniera esponenzialmente più veloce su un computer quantistico rispetto a un computer classico.

Dopo aver terminato di sviluppare l’algoritmo e, dopo una fase di test, vuole renderlo disponibile a tutti gli sviluppatori. Per far ciò utilizza il sistema. Si accede alla home page, si utilizza il tasto “*upload algorithms*” ed il sistema gli mostrerà una pagina in cui sarà presente

un form nel quale inserire il nome dell’algoritmo, la descrizione del suo funzionamento e l’algoritmo in formato *QASM*.

Dopo aver inserito le suddette informazioni, tramite il tasto “*submit*” caricherà l’algoritmo all’interno del sistema.

Successivamente, per capire se l’algoritmo è stato correttamente inserito e verificare il suo corretto funzionamento, sempre dalla home page, si piglia il tasto “*algorithms list*” per ottenere una select bar dalla quale si potrà scegliere l’algoritmo. Dopo averlo selezionato ed inserito gli input richiesti, il sistema elaborerà il risultato mostrando a video l’esito dell’operazione. Grazie a ciò si testa il corretto funzionamento dell’algoritmo sul sistema.

4.4 Inserimento e testing dell’algoritmo

Di seguito saranno descritte tutte le interfacce utente che si incontrano durante il processo di inserimento o testing dell’algoritmo.

4.4.1 Pagina di Benvenuto

La schermata di login è la prima che appare all’avvio dell’applicazione. Essenziale e intuitiva, contiene il nome dell’applicazione e permette la scelta delle operazioni da eseguire utilizzando i due bottoni presenti sotto il nome del sistema come mostrato nella Figura 4.2.

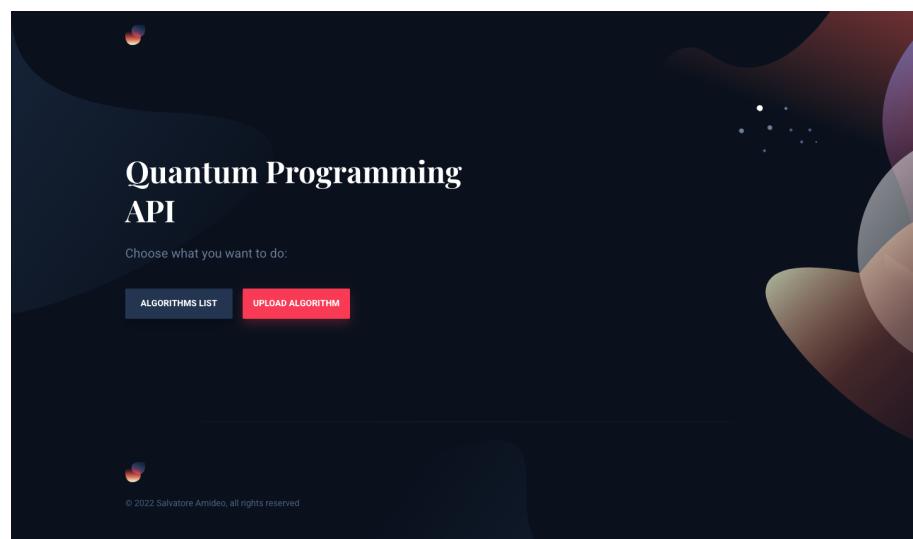


Figura 4.2: Schermata di Benvenuto

4.4.2 Pagina di Scelta Algoritmo

Se l’utente sceglie di voler utilizzare un algoritmo, dopo aver cliccato sul bottone *Algorithms List* si aprirà questa nuova schermata dove sarà possibile scegliere, grazie a un menù a tendina, gli algoritmi che possono essere utilizzati come mostrato in Figura 4.3.

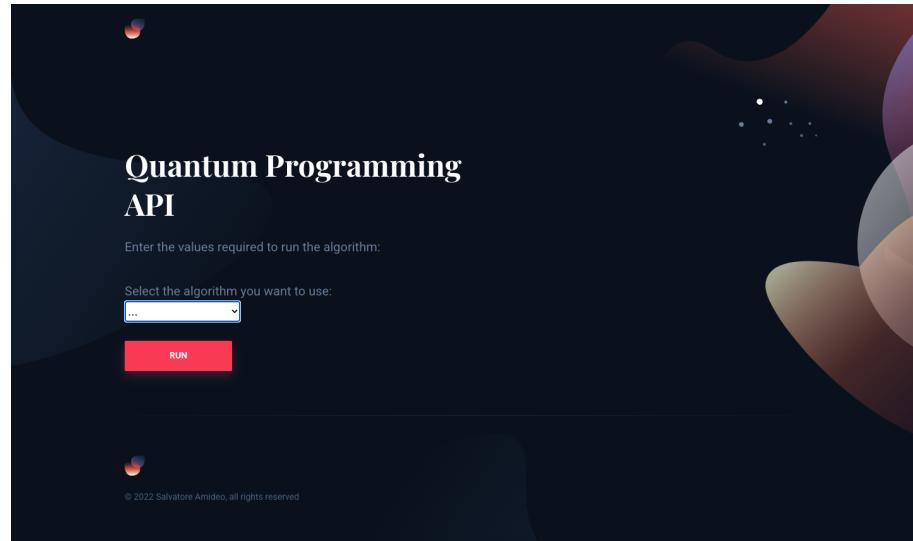


Figura 4.3: Schermata di Scelta dell’Algoritmo

4.4.3 Inserimento Dati per Esecuzione Algoritmo

Dopo la scelta dell’algoritmo da utilizzare da parte dell’utente, appariranno gli eventuali input necessari per la sua esecuzione oltre che una descrizione accurata del funzionamento dell’algoritmo e degli input.

Nella Figura 4.4 viene invece mostrato un’esempio di utilizzo dell’algoritmo **Deutsch-Jozsa**. Quest’ultimo necessita di due valori: la scelta del tipo di oracolo da utilizzare (“balanced” oppure “constant” ed il numero di input del registro. A questi, l’inserimento nel numero dei qubit che deve avere la macchina quantistica reale per l’esecuzione. Naturalmente gli input e la descrizione tra gli algoritmi, sono diversi, pertanto il sistema richiederà l’input corretto e ne mostrerà la descrizione appropriata.

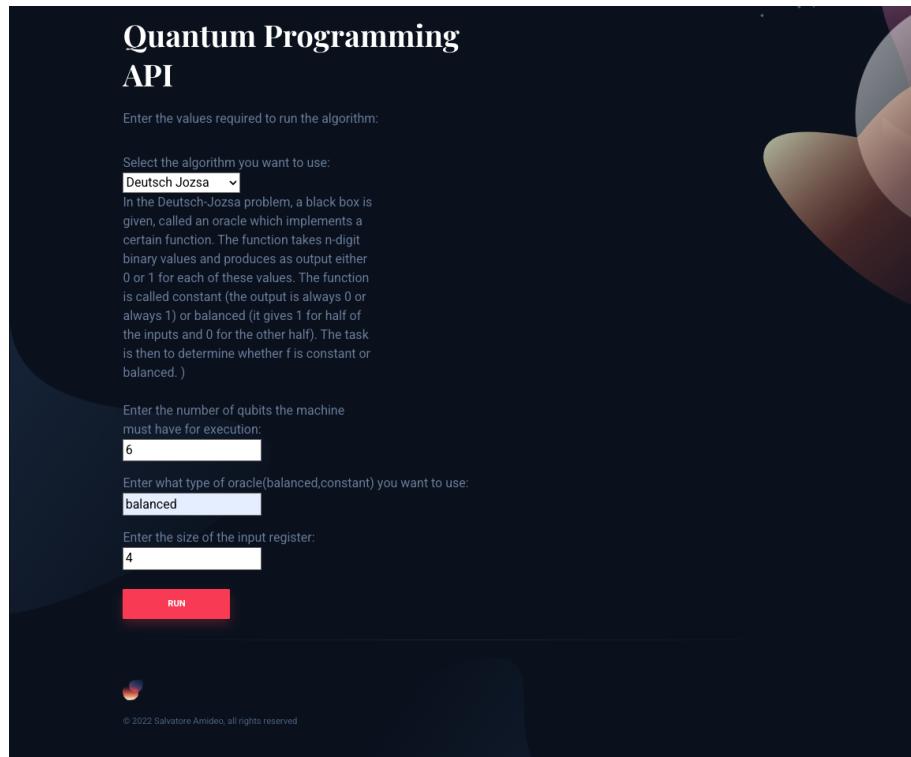
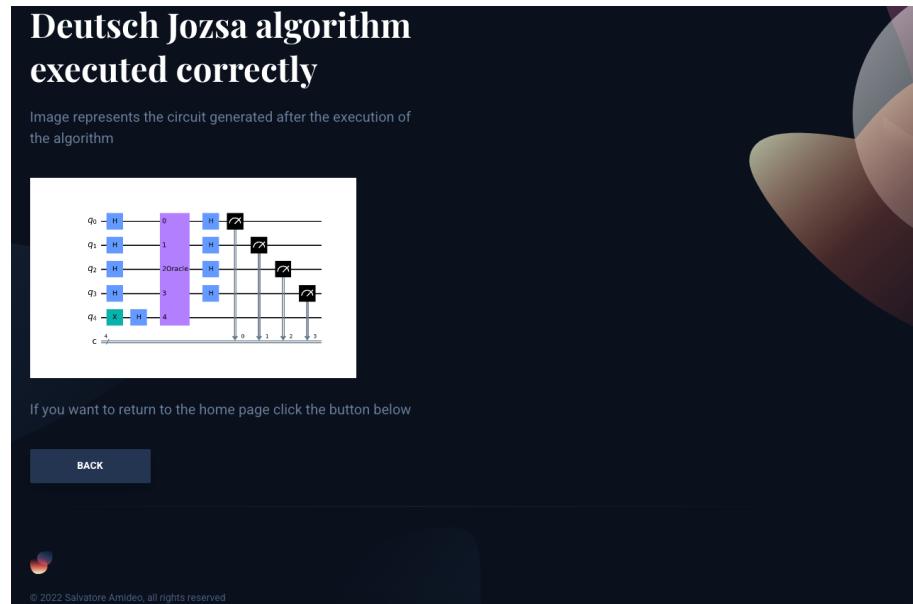


Figura 4.4: Schermata di Inserimento Input

4.4.4 Risultato Esecuzione Algoritmo

Dopo aver inserito gli input richiesti, l’utente dovrà cliccare sul tasto run per eseguire l’algoritmo. Dopo l’esecuzione dell’algoritmo, il sistema mostrerà un messaggio che comunicherà l’esito dell’operazione positiva o negativa mostrando anche l’output che è un circuito. L’utente in questo caso potrà salvare output e continuare con l’utilizzo del sistema ritornando alla pagina di benvenuto cliccando sul tasto “back”. La Figura 4.5 mostra quanto detto.

**Figura 4.5:** Schermata Esito Esecuzione Algoritmo

4.4.5 Pagina di Caricamento Algoritmo

Se l'utente invece di cliccare su *Algorithms List*, clicca su *Upload Algorithm*, verrà mostrata la seguente pagina come mostrato in Figura 4.6

Quantum Programming API

In the box below enter the algorithm you want to load:

Enter a name that identifies the algorithm:
Enter Name

Enter a description on how the algorithm works:

Enter the algorithm:

SUBMIT

Figura 4.6: Schermata Caricamento Algoritmo

4.4.6 Inserimento Algoritmo

In questa schermata l’utente ha possibilità di inserire il suo algoritmo di programmazione quantistica che poi, potrà essere utilizzato anche da altri utenti del sistema. Oltre il codice dell’algoritmo, l’utente dovrà anche inserire un nome e una descrizione sul funzionamento di esso. Nella Figura 4.7 c’è un esempio di inserimento.

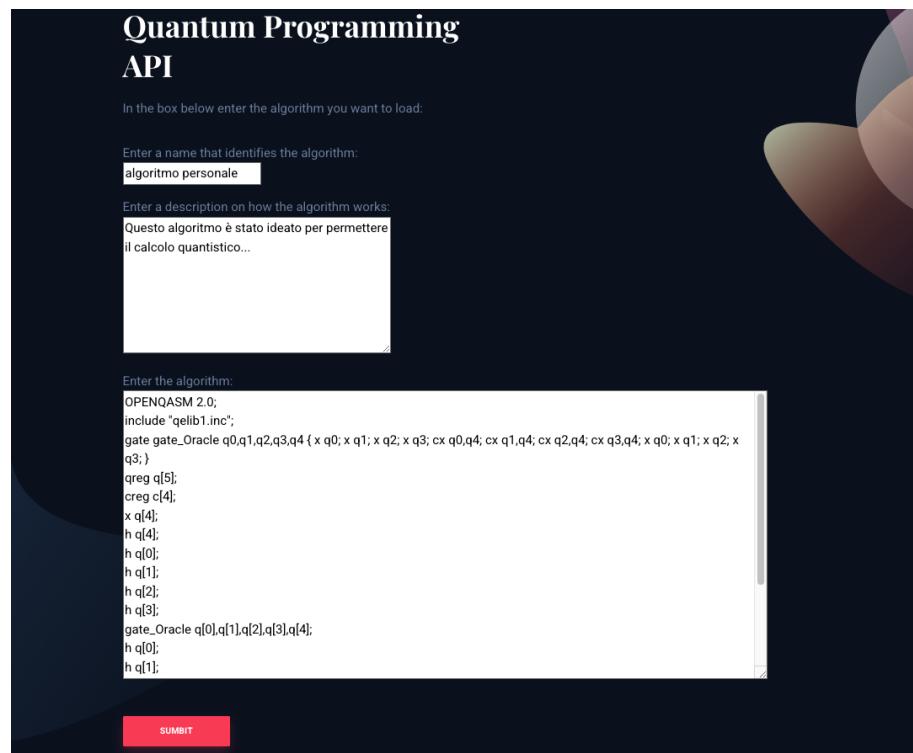


Figura 4.7: Schermata Inserimento Algoritmo

4.4.7 Risultato Inserimento Algoritmo

Dopo che l’utente ha inserito tutti i dati richiesti e cliccato su *Submit*, il sistema mostrerà un messaggio positivo se l’algoritmo è stato caricato in modo corretto, altrimenti esito negativo. Nella Figura 4.8, un esempio di messaggio positivo.

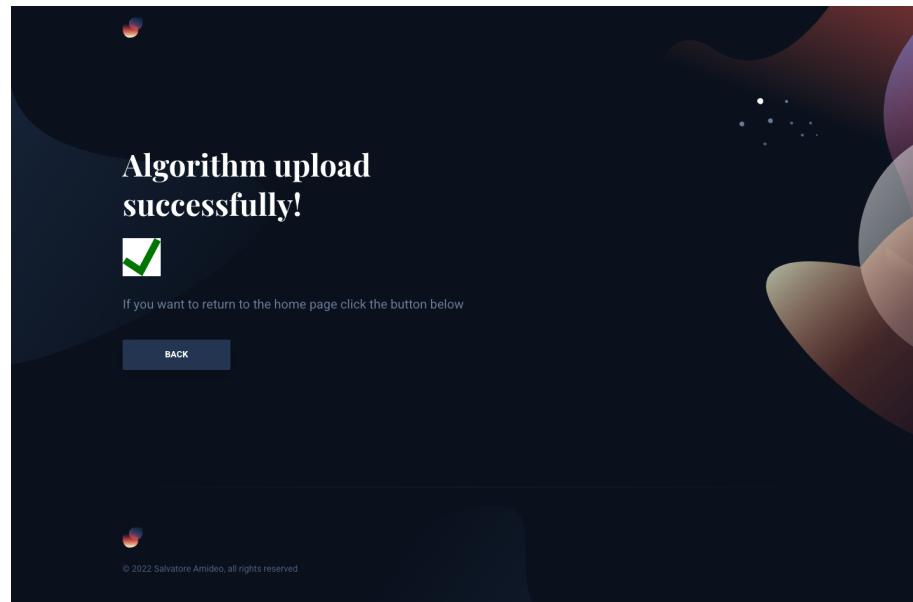


Figura 4.8: Schermata Risultato Inserimento Algoritmo

CAPITOLO 5

Conclusioni e sviluppi futuri

5.1 Conclusioni

In questa tesi si è esplorata la possibilità di realizzare un sistema che permetta di fornire un'astrazione all'esecuzione di algoritmi quantistici, tralasciando i dettagli hardware. Nella prima parte, è stato analizzato l'innovativo metodo computazionale del Quantum Computing. Innanzitutto, è importante definirne l'unità di informazione, ossia il qubit, con cui si codificano i dati reali più semplici.

Dopodiché, è stata illustrata la teoria relativa alla fisica meccanico-quantistica e il modo in cui essa viene sfruttata dall'hardware reale per il suo funzionamento. A questo punto, dopo una panoramica teorica, è stato possibile definire gli obiettivi del lavoro seguiti da una fase di progettazione e infine un esempio pratico di utilizzo.

Dopo un'analisi a caldo, si può affermare che il computer quantistico si è fatto strada tra le nuove tecnologie diventando una realtà tangibile. Gradualmente gli studiosi hanno smesso di chiedersi se la macchina quantistica fosse effettivamente realizzabile, iniziando a domandarsi invece, quando e come questo strumento potesse diventare di uso comune.

L'eccitazione per la scoperta non deve però essere fuorviante ai fini di una valutazione oggettiva del fenomeno: è difficile che i computer quantistici rimpiazzino quelli classici, quantomeno nel futuro più prossimo. Di contro è inevitabile che, esprimendo prestazioni migliori in diversi campi, questi mezzi non vengano sfruttati appieno. Almeno in una prima fase, l'ipotesi più verosimile sarebbe quella di utilizzare le macchine quantistiche come

avveniva in passato per i centri di calcolo, ovvero dando in pasto a esse determinati algoritmi per poi catalogare i risultati con i calcolatori più comuni. Molti degli attuali problemi e limiti dell'informatica sarebbero così abbattuti, permettendo lo spalancarsi di nuove porte.

Non rimane che attendere i progressi tecnologici e le scoperte che porteranno alla nascita di nuovi componenti chiave per la progettazione delle macchine quantiche, fino a raggiungere la potenza e l'efficienza necessarie per un loro utilizzo costante. Le basi della teoria dell'informazione sono state già stabilite molto tempo fa, lasciando ai posteri l'arduo compito di provarle. Nel corso degli anni queste teorie sono state dimostrate una dopo l'altra, dando il via a un processo di ottimizzazione inherente ai diversi ambiti di applicazione. È quindi indubbio che il computer quantistico sia una risorsa potente, in grado di far avanzare la scienza e l'ingegneria a un nuovo livello. Resta da capire come, quando e se i problemi legati alla effettiva realizzazione di una macchina quantistica universale verranno risolti. I computer quantistici costruiti fino ad oggi sono solo dei prototipi e non raggiungono molti degli obiettivi che il loro ramo di ricerca si prefigge, tuttavia molti scienziati sono d'accordo nel sostenere una nozione fondamentale: se non si dovesse riuscire a realizzare un calcolatore del genere almeno si comprenderebbe il perchè ciò non sia possibile.

In conclusione, non si è in grado di sapere fino a dove l'informatica quantistica si spingerà, ma si può affermare per certo che grazie all'ausilio delle conoscenze e delle tecniche raggiunte nella costruzione delle macchine quantistiche, presto saranno disponibili strumenti in grado di ottimizzare la soluzione di molti problemi di elaborazione dei dati. Un po' alla volta la barriera della decoerenza sarà abbattuta e le salde fondamenta di questo nuovo ambito scientifico saranno gettate, e arriverà il momento in cui i fisici e i ricercatori passeranno il testimone agli esperti di informatica e mentre i primi continueranno a indagare nuove possibilità, i secondi utilizzeranno ciò che fino ad allora sarà stato reso disponibile, scrivendo, bit dopo bit, il codice del futuro della specie umana.

Questa esperienza di tesi è stata molto interessante e formativa in quanto ha consentito di lavorare su un argomento nuovo ancora in fase di studio e sviluppo. Ha consentito di mettere in pratica e ampliare le conoscenze accademiche acquisite durante il percorso di studi, approfondendo concetti di meccanica quantistica al di fuori del percorso formativo effettuato.

5.2 Sviluppi Futuri

L'applicazione sviluppata può essere già utilizzabile, ma si presta all'aggiunta di altre funzionalità e all'integrazione con un altri framework di programmazione quantistica sviluppate da altre aziende come Google e Microsoft. Inoltre, per una migliore progettazione del codice, potrebbe essere utile l'implementazione del *Design Pattern Strategy* per migliorare l'utilizzo e facilitare l'integrazione di altri algoritmi o provider. Detto design pattern infatti, consente di isolare un algoritmo al di fuori di una classe, per far sì che quest'ultima possa variare dinamicamente il suo comportamento, rendendo gli algoritmi intercambiabili a runtime.

Nel nostro caso, potremmo avere più implementazioni dello stesso algoritmo. Ad esempio, l'algoritmo Deutsch Jozsa, del quale è già presente un'implementazione nel sistema, potrebbe avere un'altra più efficiente implementazione caricata successivamente da uno sviluppatore. È proprio in questa situazione che lo Strategy Pattern potrebbe semplificare la gestione delle varie versioni dello stesso algoritmo. Tutto ciò perchè esso rende possibile l'utilizzo di una qualsiasi implementazione (genericamente chiamata Strategy o Strategia) più opportuna in un determinato contesto e per una determinata richiesta.

Inoltre, di seguito si elencheranno alcune funzionalità che si potrebbero aggiungere per rendere il sistema più completo:

- Integrazione di Azure Quantum e di Cirq;
- Possibilità di scaricare gli algoritmi caricati in formato QASM;
- Integrazione di un'orchestratore;
- Integrazione di altri meta-modelli per gli algoritmi.

CAPITOLO 6

Ringraziamenti

Vorrei dedicare questa pagina del presente elaborato alle persone che mi hanno supportato in tutto il percorso universitario.

Desidero ringraziare, in particolar modo i miei genitori, che mi hanno fornito mezzi e supporto per completare questo percorso di studi. Il loro incoraggiamento è stato ciò che mi ha maggiormente sostenuto nel conseguimento di questo obiettivo, dandomi forza e coraggio per superare periodi di difficoltà. Allo stesso modo, spero di essere stato un figlio e uno studente di cui andare fieri.

Un pensiero caloroso va rivolto alla mia fidanzata, Alice, che mi ha sostenuto e sopportato durante tutto il percorso universitario. Grazie amore mio per l'affetto e la forza che ogni giorno mi offri. L'amore di chi ti è vicino si comprende nella sua grandezza, proprio nei momenti di maggiore difficoltà.

Un doveroso ringraziamento va al Prof. Andrea De Lucia, il mio mentore durante l'intero periodo di studio. Una persona e un professionista serio e ammirabile, con il quale sono orgoglioso di aver lavorato.

Un grazie anche al Dott. Manuel De Stefano, che ha saputo magistralmente coordinarmi e ispirarmi nella produzione della tesi.

Ringrazio tutti i miei amici sempre presenti anche durante quest'ultima fase del percorso di studi. Grazie per aver ascoltato i miei sfoghi, grazie per tutti i momenti di spensieratezza che mi avete regalato.

Un abbraccio va riservato anche a tutti i miei familiari.

Spero vivamente di non aver dimenticato nessuno.

Infine, un doveroso pensiero va rivolto ai miei cari defunti nonno Salvatore, nonna Giovanna e nonno Pasquale. Mi piace pensare che abbiate vegliato su di me dall'alto e che siate fieri per l'importante traguardo conseguito.

Bibliografia

- [1] M. Piattini et al. "Toward a Quantum Software Engineering". In: (). URL: <https://ieeexplore.ieee.org/document/9340056>.
- [2] S. Aaronson. "NP-complete Problems and Physical Reality". In: (). URL: <https://www.scottaaronson.com/papers/npcomplete.pdf>.
- [3] M. Ohya et al. "New quantum algorithm for studying NP-complete problems". In: (). URL: <https://arxiv.org/abs/quant-ph/0406216>.
- [4] F. Arute et al. "Quantum supremacy using a programmable superconducting processor". In: (). URL: <https://www.nature.com/articles/s41586-019-1666-5>.
- [5] Mario Piattini et al. "The Talavera Manifesto for Quantum Software Engineering and Programming". In: (). URL: <http://ceur-ws.org/Vol-2561/paper0.pdf>.
- [6] Mario Piattini et al. "Quantum Computing: A New Software Engineering Golden Age". In: (). URL: <https://dl.acm.org/doi/10.1145/3402127.3402131>.
- [7] Enrique Moguel et al. "A Roadmap for Quantum Software Engineering: applying the lessons learned from the classics". In: (). URL: <http://ceur-ws.org/Vol-2705/short1.pdf>.
- [8] Jianjun Zhao et al. "Quantum Software Engineering: Landscapes and Horizons". In: (). URL: <https://arxiv.org/abs/2007.07047>.
- [9] Redhotcyber. "Definizione Circuito Quantistico". In: (). URL: <https://www.redhotcyber.com/post/i-circuiti-quantistici-terza-lezione/>.

- [10] Microsoft. "Che cos'è un qubit?" In: (). URL: <https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-a-qubit/#introduction>.
- [11] Passione Astronomia. "Meccanica quantistica: il principio di sovrapposizione". In: (). URL: <https://www.passioneastronomia.it/meccanica-quantistica-il-principio-di-sovrapposizione/>.
- [12] Wikipedia. "Computer quantistico". In: (). URL: https://it.wikipedia.org/wiki/Computer_quantistico.
- [13] Vittorio Rienzo. "Raffreddamento magnetico permanente elettronica quantistica". In: (). URL: <https://www.tomshw.it/hardware/raffreddamento-magnetico-permanente-per-lelettronica-quantistica>.
- [14] Kiutra. "Kiutra". In: (). URL: <https://kiutra.com/>.
- [15] Wikipedia. "DiVincenzo Criteria". In: (). URL: <https://qc-at-davis.github.io/QCC/How-Quantum-Computing-Works/Divincenzo's-Criteria/Divincenzo's-Criteria.html>.
- [16] IBM. "Qiskit". In: (). URL: <https://qiskit.org/documentation/>.
- [17] Google. "Cirq". In: (). URL: <https://quantumai.google/cirq>.
- [18] Microsoft. "Azure Quantum". In: (). URL: <https://docs.microsoft.com/it-it/azure/quantum/overview-azure-quantum>.
- [19] Amazon. "Amazon Braket". In: (). URL: <https://aws.amazon.com/it/braket/>.
- [20] Zapata. "Zapata Computing". In: (). URL: <https://www.zapatacomputing.com/>.
- [21] QuantumPath. "QuantumPath". In: (). URL: <https://www.quantumpath.es/>.
- [22] El aoun et al. "Understanding Quantum Software Engineering Challenges An Empirical Study on Stack Exchange Forums and GitHub Issues". In: (). URL: <https://ieeexplore.ieee.org/document/9609196>.
- [23] Manuel De Stefano et al. "Software engineering for quantum programming: How far are we?" In: (). URL: <https://www.sciencedirect.com/journal/journal-of-systems-and-software>.
- [24] Pérez-Castillo et al. "Modern Software Engineering Concepts and Practices: Advanced Approaches". In: (). URL: <https://www.igi-global.com/book/modern-software-engineering-concepts-practices/46012>.

- [25] Pérez-Castillo et al. “A standard to modernize legacy systems”. In: (). URL: <https://www.sciencedirect.com/science/article/abs/pii/S0920548911000183>.
- [26] Indika Kumara et al. “QSOC: Quantum Service-Oriented Computing”. In: (). URL: <https://discord.com/channels/770242655807471616/963699257578238022/1024220784765448222>.
- [27] Brambilla M. et al. “Model-Driven Software Engineering in Practice”. In: () .
- [28] Papazoglou et al. “Designing the cloud.” In: (). URL: https://link.springer.com/chapter/10.1007/978-3-319-04090-5_10.
- [29] Papazoglou et al. “Service-oriented design and development methodology”. In: (). URL: <https://research.tilburguniversity.edu/en/publications/service-oriented-design-and-development-methodology>.
- [30] Di Nitto E. et al. “An approach to support automated deployment of applications on heterogeneous cloud-hpc infrastructures.” In: () .
- [31] Kumara I. et al. “Application orchestration on cloud-edge infrastructure.” In: (). URL: <https://www.mdpi.com/1424-8220/22/5/1755/htm>.
- [32] Leymann F. et al. “Quantum in the cloud: application potentials and research opportunities”. In: (). URL: <https://arxiv.org/abs/2003.06256>.
- [33] Enrique Moguel et al. “Quantum service-oriented computing”. In: (). URL: <https://link.springer.com/article/10.1007/s11219-022-09589-y>.
- [34] Nielsen M. et al. “Quantum computation and quantum information”. In: () .
- [35] Cuomo D. et al. “Toward a distributed quantum computing ecosystem”. In: (). URL: <https://doi.org/10.1049/IET-QTC.2020.0002>.
- [36] Rojo J. et al. “Trials and Tribulations of the development of hybrid quantum microservices systems”. In: () .
- [37] Dreher P. et al. “Container-based prototype platform for the development of extreme quantum computing algorithms”. In: (). URL: <https://doi.org/10.1109/HPEC.2019.8916430>.
- [38] Manuel De Stefano et al. “Towards Quantum-Algorithms-as-a-Service”. In: () .

APPENDICE A

Tecnologie utilizzate

A.1 HTML, CSS, JavaScript

Queste tre sono le tecnologie fondamentali che stanno alla base di tutte le web applicazioni moderne. **HTML** (*Hyper Text Markup Language*) è un linguaggio di formattazione utilizzato per creare la struttura di una pagina web ed inserirne i contenuti.

La formattazione consiste nell'inserire nel testo una serie di marcatori (*tag*) che definiscono come questo debba essere visualizzato e disposto all'interno della pagina.

CSS (*Cascading style Sheet*) è un linguaggio usato per definire la formattazione e lo stile di una pagina HTML.

Questa tecnologia è stata introdotta per separare i contenuti di una pagina HTML dalla formattazione; in questo modo strutturando correttamente un documento HTML, è possibile cambiarne lo stile modificando semplicemente alcune proprietà nel file CSS a esso associato.

Infine **JavaScript** è un linguaggio di programmazione orientato agli oggetti. A differenza di altri linguaggi di programmazione, che permettono la scrittura di programmi stand-alone, *JavaScript* viene utilizzato soprattutto come linguaggio di scripting, ovvero viene integrato con altri software per programmarne alcuni aspetti.



Figura A.1: I loghi di HTML, JavaScript, CSS

A.2 Python

Il linguaggio utilizzato è stato Python, in quanto interattivo, dinamico e portatile. Fornisce una libreria standard che gestisce in modo automatico la memoria e consente lo sviluppo di servizi web in modo semplice eseguendo lo stesso codice su molteplici piattaforme.



Figura A.2: Il logo di Python

Le librerie principali utilizzate per lo sviluppo sono:

- **Flask:** è una libreria Python usata come framework WSGI per applicazioni web. Tale libreria permette una facile implementazione di una applicazione web quale una API.
- **Flask-RESTPlus:** è una libreria aggiunta a Flask che facilita l'implementazione di una API in Python, con una sintassi più comprensibile e una generazione automatica della documentazione usando Swagger UI.

- **Numpy:** è una libreria open source per il linguaggio di programmazione Python, che aggiunge supporto a grandi matrici e array multidimensionali insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati. È stata creata nel 2005 da Travis Oliphant basandosi su Numeric di Jim Hugunin
- **Qiskit:** è una libreria che ti consente di progettare e testare i propri circuiti su di un simulatore installato sulla macchina locale o su un computer quantico reale.
- **Matplotlib:** è una libreria per la creazione di grafici per il linguaggio di programmazione Python e la libreria matematica NumPy. Fornisce API orientate agli oggetti che permettono di inserire grafici all'interno di applicativi usando toolkit GUI generici, come WxPython, Qt o GTK. All'inizio la libreria venne fatta principalmente da John Hunter e distribuita sotto licenza di tipo BSD.
- **Pandas:** è una libreria software scritta per il linguaggio di programmazione Python per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali. È un software libero rilasciato sotto la licenza BSD a tre clausole. Il nome deriva dal termine "panel data", termine econometrico per set di dati che include osservazioni su più periodi di tempo per gli stessi individui.
- **Pymongo:** è una libreria che permette la interazione di codice in Python con un database MongoDB.

A.3 Docker

Docker è un progetto open-source che automatizza il deployment di applicazioni all'interno di contenitori software, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. Docker utilizza le funzionalità di isolamento delle risorse del kernel Linux come ad esempio cgroups e namespaces per consentire a "*container*" indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale.

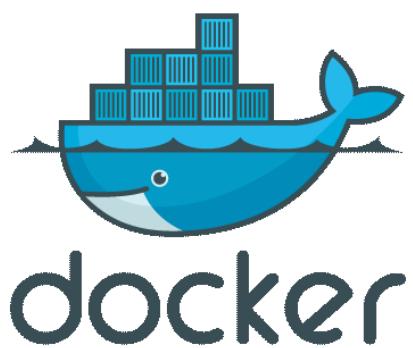


Figura A.3: Il logo di Docker