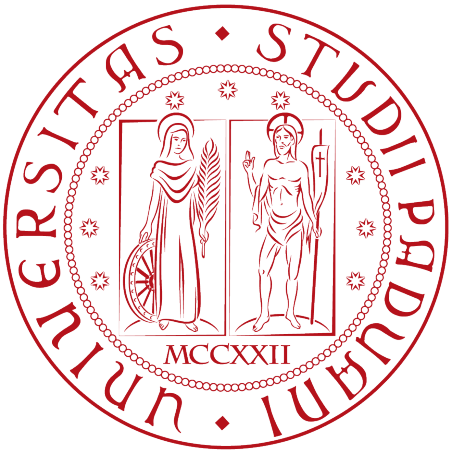
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Un'applicazione per l'interazione con macchinari industriali

Tesi di laurea

Relatore

Prof. Paolo Baldan

Laureando

Luca Biasotto

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Luca Biasotto presso l'azienda Ubware s.r.l. L'obbiettivo da raggiungere era quello di creare un software in grado di connettersi con macchinari industriali di tipo FANUC, leggere i dati trasmessi da questi e inviare loro comandi. Il codice è stato scritto come codice a oggetti di alto livello prevedendo che possa avere un lungo ciclo di vita. Il linguaggio utilizzato è stato Python (versione 3), con l'integrazione di una libreria preesistente scritta in C. In seguito è stata automatizzata la lettura e scrittura di dati tra il macchinario e un database SQL e verso un gestionale terzo tramite l'utilizzo di API REST. È stato infine migliorato il codice back-end della WebApp che consente all'utente finale di consultare i dati salvati nel database e di inviare nuovi comandi al macchinario.

Ringraziamenti

Vorrei esprimere la mia gratitudine al Prof. Paolo Baldan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro, nonché per i preziosi insegnamenti trasmessi durante il percorso di studi.

Desidero inoltre ringraziare con affetto i miei genitori e mia sorella Elena per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio. Desidero inoltre esprimere un ringraziamento speciale alla mia compagna, Chiara, per aver condiviso con me questi annui di vita.

Padova, luglio 2023 Luca Biasotto

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Introduzione al progetto

La presente tesi è il risultato dell'esperienza acquisita durante uno stage coinvolgente e istruttivo presso un'azienda leader nel settore industriale. Durante lo stage, ho avuto l'opportunità di lavorare su un progetto di grande interesse: lo sviluppo di un software innovativo in grado di interagire con i macchinari industriali di tipo FANUC. L'obiettivo principale di questo software è stato quello di leggere i dati trasmessi dai macchinari e inviare comandi ad essi. Il software è stato sviluppato come un codice a oggetti di alto livello, progettato per avere un ciclo di vita duraturo nel tempo.

La sfida principale è stata rappresentata dalla vasta diffusione dei macchinari FANUC sul mercato e dalla loro diversità di modelli e configurazioni. Di conseguenza, la mia prima responsabilità è stata quella di creare un software generico, in grado di funzionare su tutti i macchinari FANUC di tipo standard. Questo ha richiesto una progettazione attenta e accurata, al fine di garantire la massima flessibilità e adattabilità del codice a diverse situazioni. In particolare, ho lavorato per assicurare che il software potesse essere facilmente esteso per gestire casi particolari che richiedessero modifiche rispetto allo standard.

Parallelamente allo sviluppo del software generico, mi è stato richiesto di produrre un codice personalizzato per un'azienda cliente specifica. Ciò ha comportato l'esigenza di apportare modifiche al codice standard per adattarlo alle particolari esigenze del macchinario di tale cliente. È stato un compito impegnativo che mi ha permesso di mettere alla prova le mie competenze di programmazione e adattamento.

Per il linguaggio di programmazione, è stata scelta la versatilità di Python (versione 3.10), che si è dimostrato una scelta ideale per affrontare le complessità del progetto. Tuttavia, è stato necessario interagire con una libreria preesistente scritta in C, richiedendo una stretta collaborazione con i colleghi del reparto di sviluppo.

Una volta implementata con successo la comunicazione tra il software e i macchinari, mi sono dedicato all'automazione della lettura e scrittura dei dati tra i macchinari e un database. Questa fase è stata cruciale per ottimizzare i processi aziendali, consentendo una gestione più efficiente e centralizzata delle informazioni.

L'importanza e l'impatto positivo del mio lavoro durante lo stage hanno aperto la strada a un'ulteriore evoluzione del progetto. Sulla base dei risultati ottenuti, è stata pianificata l'implementazione di una WebApp, che consente un'interazione più intuitiva e semplificata con il software. In questa fase successiva, si sono definiti i framework e i linguaggi di programmazione più adatti per realizzare questa WebApp, tenendo conto

delle esigenze specifiche dell'azienda e dei suoi clienti.

Durante tutto il periodo dello stage, ho avuto il privilegio di lavorare a stretto contatto con il talentuoso team di ricerca e sviluppo dell'azienda. Questa esperienza mi ha fornito una preziosa panoramica sulla realtà lavorativa in questo settore, consentendomi di ampliare le mie conoscenze e competenze in vari ambiti. Oltre al mio progetto principale, ho avuto l'opportunità di collaborare a progetti minori, contribuendo al successo del team in settori come l'intelligenza artificiale, l'automazione dei processi e l'ottimizzazione dei flussi di lavoro.

Attraverso questa tesi, intendo condividere le sfide, le soluzioni e i risultati raggiunti durante lo stage, sottolineando l'importanza di un approccio metodico e flessibile nello sviluppo di software per macchinari industriali. Spero che questa ricerca possa fornire una base solida per futuri progetti simili e contribuire alla crescita e all'innovazione del settore industriale.

1.2 Obiettivi

Lo stage si propone di raggiungere diversi obiettivi chiave nell'ambito dello sviluppo di software per macchinari industriali di tipo FANUC. L'obiettivo principale è la creazione di un software completo, scritto in Python versione 3, che sia in grado di connettersi e dialogare con i macchinari industriali FANUC. Tale software dovrà essere altamente flessibile e facilmente estensibile per permettere future modifiche e personalizzazioni, se necessario.

Un primo obiettivo è l'analisi dei requisiti e lo studio di fattibilità. Durante questa fase, verrà effettuato un approfondito esame delle specifiche del progetto e delle caratteristiche dei macchinari FANUC, al fine di identificare le funzionalità chiave necessarie per garantire un'efficace connessione e comunicazione tra il software e i macchinari stessi.

Successivamente, si procederà alla progettazione del software, che comprenderà la definizione di una gerarchia di tipi appropriata e la selezione dei linguaggi di programmazione e dei framework più adatti per implementare il software. Sarà cruciale anche considerare l'integrazione del software nella suite di prodotti offerti dall'azienda, al fine di garantire una perfetta compatibilità e interazione con gli altri strumenti utilizzati dall'azienda stessa.

Durante la fase di sviluppo, si procederà alla scrittura del codice, implementando la gerarchia dei tipi e le classi di base necessarie per la gestione delle connessioni con i macchinari FANUC. Sarà fondamentale garantire che il codice sia modulare, ben strutturato e in grado di adattarsi a diverse situazioni e requisiti specifici.

Un importante obiettivo sarà anche la realizzazione di un'interfaccia di comunicazione efficiente con una libreria preesistente scritta in linguaggio C. Questo consentirà una comunicazione a basso livello con i macchinari, permettendo al software di interagire direttamente con le funzionalità e i dati forniti dai macchinari FANUC.

Un altro passo significativo sarà il testing del codice su un macchinario reale, al fine di verificarne il corretto funzionamento e identificare eventuali bug o problematiche da risolvere. Sarà necessario anche scrivere il codice per l'interazione con un database, utilizzando gli strumenti selezionati in fase di progettazione, al fine di consentire la lettura e la scrittura dei dati tra i macchinari e il database stesso.

Al termine dello stage, ci si aspetta di ottenere un software completo e funzionante, in grado di connettersi e dialogare con i macchinari industriali FANUC. Questo software sarà facilmente estensibile e modificabile, grazie alla sua struttura modulare

e all'attenzione dedicata alla progettazione di una gerarchia di tipi flessibile. Inoltre, il software dovrà rispondere alle specifiche del progetto e alle esigenze dell'azienda, garantendo un'interazione fluida e affidabile con i macchinari.

Lo stage fornirà l'opportunità di acquisire competenze pratiche nell'ambito dello sviluppo di software per macchinari industriali, oltre a permettere una completa immersione nel contesto di ricerca e sviluppo dell'azienda. Sarà un'occasione per lavorare a progetti minori in diversi ambiti e per sperimentare direttamente la realtà lavorativa nel settore industriale.

1.3 Organizzazione del testo

Si riporta di seguito l'organizzazione del testo all'interno di questa tesi.

Il secondo capitolo approfondisce il contesto in cui lo stage si è svolto, l'azienda proponente e la pianificazione del lavoro.

Il terzo capitolo descrive le tecnologie utilizzate durante lo svolgimento del progetto.

Il quarto capitolo descrive la struttura dell'applicativo realizzato.

Il quinto capitolo approfondisce alcuni aspetti particolarmente significativi del codice.

Il sesto capitolo approfondisce gli aspetti legati alla verifica e al test del software prodotto.

Il settimo capitolo descrive gli obiettivi raggiunti e le possibilità di sviluppo futuro.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parte del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione dello stage

2.1 L'azienda

L'azienda presso la quale lo stage si è svolto si chiama Ubware s.r.l. Si tratta di un'azienda che produce software per diverse aziende clienti, principalmente nelle provincie di Padova e Treviso. Ubware s.r.l. è strettamente legata a Trevigroup s.r.l, di cui, di fatto, costituisce il reparto ricerca e sviluppo. Ubware sviluppa soluzione software per aziende clienti, così come software in ausilio alle altre attività di Trevigroup. I principali prodotti software includono:

- Una *suite* di applicazioni web per la gestione di processi aziendali.
- Software per la connessione a macchinari industriali di vario tipo.
- Software di intelligenza artificiale.
- Personalizzazioni su software gestionali venduti da aziende di cui Trevigroup è partner.

Figura 2.1: Logo di Ubware

Nel corso dello stage l'attività principale ha riguardato lo sviluppo di software per l'interazione con macchinari industriali. Tuttavia, in funzione dell'inserimento nella realtà aziendale sono stati assegnati compiti anche nell'ambito dello sviluppo web e dell'intelligenza artificiale, nonchè l'utilizzo dei vari software per la gestione del workflow aziendale.

2.2 L'idea

Numerose aziende nel territorio pongono come richiesta a Ubware s.r.l. l'implementazione di software che consentano di automatizzare le operazioni aziendali che di lettura e scrittura dati da e verso i macchinari che impiegano, con l'obiettivo di aumentare la produttività e l'efficienza dell'attività economica. Questo genere di prodotti software sono inoltre legati al sistema di incentivi legati al progetto 'Industria 4.0'.

Per rispondere a queste esigenze Ubware ha sviluppato una soluzione software che consiste in un programma scritto in Python e organizzato in classi, che è stato venduto ad alcuni clienti, Dal momento che ogni ditta possiede macchinari di tipo diverso e con diversi protocolli di comunicazione, la struttura generale del programma va adattata di volta in volta sviluppando codice più specifico. Nel caso dei macchinari *FANUC*, era stata compiuta un'analisi e un tentativo iniziale di sviluppo nell'ambito dello studio di fattibilità, che aveva però riscontrato dei problemi a causa del protocollo di comunicazione utilizzato.

L'idea è stata quella di proporre allo stagista la ricerca di una soluzione per questo problema e riuscire a effettuare con successo una comunicazione verso il macchinario. Nel caso la soluzione fosse stata trovata, sarebbe stato poi necessario scrivere il resto del codice necessario a implementare le diverse automazioni richieste dai clienti. Altrimenti, si sarebbe dovuto produrre un'analisi dettagliata per spiegare le ragioni che rendono impossibile risolvere il problema ed eventualmente proporre una soluzione alternativa che venisse incontro quanto più possibile alle richieste dei clienti.

2.3 Lo stage

Una volta accettato lo stage, il tirocinante è stato inserito nell'ambiente di sviluppo aziendale. Sono stati creati i necessari account sui server di sviluppo ed è stata fornita un'analisi più dettagliata di quanto richiesto. È stata anche effettuata una panoramica degli strumenti utilizzati, come l'IDE VSCode

Il primo passo è stato quello di formalizzare in opportuni documenti i requisiti necessari. Sono state poi definite le tecnologie che sarebbero state utilizzate. Un esempio è stata la scelta di utilizzare un database SQL e la versione 3 di Pyhton come linguaggio di programmazione generale. Le motivazioni specifiche dietro queste scelte saranno approfondite nei capitoli successivi. È stata anche definita la struttura di alto livello del codice.

Si è poi pensato a risolvere il problema di comunicazione sopra descritto, che rappresentava il principale ostacolo alla riuscita del progetto. La tecnica utilizzata è spiegata nel dettaglio nel capitolo 5.

Di conseguenza, è stato possibile proseguire con l'implementazione del resto del programma Python, e successivamente con i relativi test.

Nel tempo rimanente, si è iniziato a lavorare ai requisiti non obbligatori, per i quali il lavoro necessario è stato impostato, anche se non portato a termine.

2.4 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si sarebbe potuto andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Raccolta dati dal macchinario
Descrizione: Impossibilità di riuscire a leggere tutti i dati necessari dal macchinario a causa dell'incompatibilità del modello specifico.
Soluzione: È stato coinvolto il responsabile a capo del progetto. Si è determinato che l'impossibilità di leggere tutti i dati richiesti, se dimostrata, è accettabile. Studio di una soluzione alternativa in cui si legge un sottoinsieme dei dati richiesti per i quali la possibilità di lettura è già stata dimostrata in precedenza.

2. Errori nel software prodotto
Descrizione: È possibile che la presenza di bug ed errori nel prodotto finale abbiano un impatto negativo sui macchinari che riceveranno istruzioni tramite il programma stesso.
Soluzione: Coinvolgimento del cliente e preparazione di sessioni di test in contesti controllati per testare il software in maniera sicura prima di farlo entrare in produzione.

2.5 Requisiti

Nel primo giorno di stage si è svolto un incontro con il tutor aziendale per definire in modo dettagliato i requisiti funzionali dell'applicativo, elencati nella tabella 2.1.
Ogni requisito è classificato e identificato univocamente attraverso un codice composto secondo il seguente schema:

`R[Priorità]-[Identificativo]`

Dove:

- **R** indica che si tratta di un requisito;
- **Priorità** può assumere i seguenti valori:
  - **O**: requisito obbligatorio;
  - **D**: requisito desiderabile;
  - **F**: requisito facoltativo.
- **Identificativo**: numero progressivo che identifica il requisito in forma gerarchica, strutturato come segue:

`[codicePadre].[codiceFiglio]`

Codice	Descrizione
`RO-1`	Sviluppo di codice in grado di interagire con il macchinario.
`RO-1.1`	Sviluppo di una funzione in grado di leggere dati dal macchinario.
`RO-1.2`	Sviluppo di una funzione in grado di scrivere dati sul macchinario.
`RO-2`	Sviluppo di codice in grado di interagire con il database.

Codice	Descrizione
RO-2.1	Sviluppo di una funzione in grado di leggere dati dal database.
RO-2.2	Sviluppo di una funzione in grado di scrivere dati sul database.
RO-3	Sviluppo di codice dedicato al test del programma e attività di testing.
RD-1	Sviluppo di API per l'interazione con un software gestionale.
RF-1	Miglioramento delle performance della WebApp utilizzata dai clienti.

Tabella 2.1: Requisiti

2.6 Pianificazione

La pianificazione settimanale per lo stage è stata organizzata come segue:

Prima Settimana (40 ore):

- Si è tenuto un incontro con il proponente per discutere delle prime linee guida del progetto.
- Il proponente ha presentato in modo esaustivo il progetto da sviluppare, fornendo dettagli e indicazioni chiare.
- È stata condotta un'analisi dei requisiti per valutare la fattibilità del progetto.
- È stato assegnato uno spazio di lavoro in azienda e sono state configurate le attrezzature hardware e software necessarie per svolgere il lavoro.
- È iniziata la fase di progettazione del software.

Seconda Settimana (40 ore):

- È stata definita la gerarchia dei tipi da utilizzare nel software.
- Sono stati selezionati i linguaggi di programmazione, le librerie e i framework da impiegare.
- Si è analizzato come il software prodotto potrà essere integrato nella suite di prodotti offerti dall'azienda.
- È stata conclusa la progettazione del codice ad alto livello e sono state definite le modalità di interazione con la libreria preesistente per gestire l'interfaccia a basso livello con i macchinari.
- Si è appreso l'organizzazione del lavoro all'interno dell'azienda e si è partecipato a specifici compiti aziendali a scopo formativo.

Terza Settimana (40 ore):

- È iniziata la scrittura effettiva del codice.
- Sono state implementate le classi di base e la gerarchia a livello elevato.
- È stata sviluppata la classe concreta responsabile della gestione della connessione al macchinario.

Quarta Settimana (40 ore):

- È stato eseguito il testing del codice su un macchinario reale.
- Si è proceduto alla scrittura del codice per l'interazione con il database, utilizzando gli strumenti precedentemente scelti durante la fase di progettazione.

Quinta Settimana (40 ore):

- È stato effettuato un testing completo dell'applicazione utilizzando un macchinario reale.
- È stata prodotta la documentazione riguardante il codice sviluppato.

Sesta Settimana (40 ore):

- È stato scritto il codice per gestire le API.

Settima Settimana (40 ore):

- Sono state effettuate delle migliorie sul codice front-end della WebApp che consente al cliente di consultare il database e inviare ordini al macchinario.

Ottava Settimana (40 ore):

- È stata effettuata una revisione del codice e sono state testate le modifiche apportate alla WebApp.

Capitolo 3

Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

3.1 Python

Python è un linguaggio di programmazione versatile e di alto livello noto per la sua semplicità e leggibilità. È stato creato da Guido van Rossum e rilasciato per la prima volta nel 1991. Python ha guadagnato immensa popolarità nel corso degli anni ed è diventato uno dei linguaggi di programmazione più utilizzati in vari settori, tra cui lo sviluppo web, il calcolo scientifico, l'analisi dei dati, l'intelligenza artificiale e l'automazione.

Uno dei punti di forza di Python è la sua sintassi elegante, che mette l'accento sulla leggibilità del codice e riduce la quantità di codice necessaria per esprimere concetti complessi. Utilizza l'indentazione per definire i blocchi di codice, eliminando la necessità di parentesi graffe o parole chiave esplicite. Questo approccio incoraggia gli sviluppatori a scrivere codice pulito, organizzato e manutenibile.

Python supporta più paradigmi di programmazione, tra cui la programmazione procedurale, orientata agli oggetti e funzionale. Fornisce ampie librerie standard e framework che facilitano compiti comuni e promuovono il riutilizzo del codice. Inoltre, Python ha un vasto ecosistema di pacchetti e moduli di terze parti disponibili attraverso il Python Package Index (PyPI), che consente agli sviluppatori di sfruttare soluzioni predefinite per vari scopi.

La versatilità di Python deriva dalla sua capacità di funzionare su più piattaforme e sistemi operativi. È un linguaggio interpretato, il che significa che il codice sorgente viene eseguito riga per riga al momento dell'esecuzione senza la necessità di compilazione. Ciò lo rende molto accessibile e flessibile per la prototipazione rapida, scripting e sviluppo interattivo.

La ricca serie di librerie e framework del linguaggio lo rende adatto a una vasta gamma di applicazioni. Per lo sviluppo web, Python dispone di framework come Django e Flask, che facilitano la creazione di applicazioni web scalabili e robuste. Per il calcolo scientifico e l'analisi dei dati, librerie come NumPy, Pandas e Matplotlib forniscono potenti strumenti per calcoli numerici, manipolazione dei dati e visualizzazione. La popolarità di Python nel campo dell'intelligenza artificiale e del machine learning è evidente dai framework come TensorFlow, Keras e PyTorch, che offrono implementazioni efficienti di reti neurali e algoritmi di deep learning.

Figura 3.1: Logo di Python

3.1.1 La libreria SQLAlchemy

SQLAlchemy è una libreria Python potente ed estremamente flessibile progettata per semplificare l'interazione con database relazionali. Ciò che distingue SQLAlchemy è la sua capacità di abbinare l'eleganza della programmazione Python con la potenza delle operazioni SQL. Offre una soluzione completa per la gestione dei dati in applicazioni Python, consentendo agli sviluppatori di lavorare con i dati in modo intuitivo e altamente personalizzabile.

Un aspetto chiave di SQLAlchemy è il suo Object-Relational Mapping (ORM), che consente di rappresentare le tabelle di un database come classi Python e le righe come oggetti. Questa astrazione rende la gestione dei dati più naturale e consente di scrivere codice più pulito ed esplicativo. L'ORM semplifica notevolmente le operazioni di lettura e scrittura dei dati, consentendo agli sviluppatori di concentrarsi sulla logica dell'applicazione anziché sui dettagli della query SQL.

SQLAlchemy offre anche un supporto multi-database, il che significa che è possibile utilizzarlo con una vasta gamma di sistemi di gestione di database, tra cui MySQL, PostgreSQL, SQLite e Oracle, tra gli altri. Questa flessibilità lo rende una scelta ideale per progetti che richiedono la compatibilità con diversi database.

Inoltre, SQLAlchemy è altamente personalizzabile ed estendibile. Fornisce strumenti per creare schemi di database in modo dichiarativo, eseguire migrazioni per gestire gli aggiornamenti dello schema e ottimizzare le query per le prestazioni

3.1.2 La libreria FastAPI

FastAPI è un framework web moderno e leggero progettato per la creazione di API RESTful in Python con un'enfasi particolare sulla facilità d'uso, le prestazioni e la documentazione automatica. È stato progettato per semplificare la creazione di servizi web robusti, consentendo agli sviluppatori di concentrarsi sulla logica dell'applicazione anziché sulla gestione dei dettagli di basso livello.

Una delle caratteristiche più distintive di FastAPI è la sua capacità di generare automaticamente documentazione interattiva per le API basata sugli standard OpenAPI. Questo significa che gli sviluppatori possono documentare le loro API semplicemente definendo le tipologie di dati utilizzate nei parametri delle richieste e delle risposte. Tale documentazione risultante è esplorabile tramite un'interfaccia web interattiva.

FastAPI è altamente performante grazie al suo utilizzo di Pydantic per la convalida dei dati e di ASGI (Asynchronous Server Gateway Interface) per la gestione delle richieste asincrone. Ciò lo rende ideale per applicazioni che richiedono una gestione efficiente delle richieste concorrenti.

Un'altra caratteristica di FastAPI è il supporto per i tipi di dati Python 3.6+ e la convalida automatica dei dati in ingresso e in uscita. Questo rende più sicure le API e consente agli sviluppatori di evitare molti errori comuni legati ai dati.

Inoltre, FastAPI offre una serie di funzionalità per semplificare la gestione delle risorse, tra cui la gestione automatica dei percorsi delle risorse, la gestione delle eccezioni personalizzate e la gestione delle autorizzazioni basate su ruoli.

3.2 Git

Git è un sistema di controllo versione distribuito ampiamente utilizzato nel processo di sviluppo del software. È stato creato da Linus Torvalds nel 2005 ed è diventato uno degli strumenti più popolari per la gestione del codice sorgente.

Git consente ai membri del team di collaborare in modo efficiente, tenere traccia delle modifiche apportate al codice e coordinare il lavoro su progetti complessi. Ecco alcune delle sue caratteristiche principali e il suo valore per lo sviluppo in team:

- Controllo delle versioni: Git tiene traccia di ogni modifica apportata al codice sorgente nel tempo. Ciò consente ai membri del team di lavorare in parallelo su diverse funzionalità o rami di sviluppo senza interferire tra loro. Inoltre, consente di visualizzare la cronologia delle modifiche, confrontare versioni e ripristinare facilmente una versione precedente del codice se necessario.
- Branching e merging: Git facilita la creazione di branch, cioè copie isolate del codice sorgente. Ciò consente ai membri del team di sviluppare nuove funzionalità o risolvere bug senza influire sul lavoro principale. Successivamente, i branch possono essere uniti (merged) nel ramo principale (tipicamente chiamato "master" o "main"). Questo processo semplifica la gestione delle modifiche parallele e consente un'integrazione graduale delle nuove funzionalità nel codice principale.
- Collaborazione: Git facilita la collaborazione tra membri del team. Ogni membro può clonare il repository Git sul proprio ambiente di sviluppo locale, lavorare su di esso e successivamente caricare (push) le proprie modifiche nel repository condiviso. Questo consente una modalità di lavoro asincrona, in cui i membri del team possono lavorare su diverse parti del codice in modo indipendente.
- Risoluzione dei conflitti: Quando i membri del team apportano modifiche allo stesso file o alla stessa sezione di codice, potrebbero verificarsi dei conflitti durante il merging dei branch. Git fornisce strumenti per gestire questi conflitti, consentendo ai membri del team di unire manualmente le modifiche in conflitto. Ciò consente di mantenere l'integrità del codice e di assicurare che tutte le modifiche vengano considerate.
- Backup e ripristino: Git funge anche da sistema di backup per il codice sorgente. Tutte le modifiche apportate al repository vengono registrate in modo permanente, consentendo di ripristinare lo stato precedente del codice in qualsiasi momento. Ciò offre una maggiore sicurezza contro la perdita accidentale di dati o la corruzione del codice.

Figura 3.2: Logo di Git

3.2.1 GitLab

L'azienda dispone di un server proprio su quale viene salvato il codice prodotto, che viene gestito grazie al software GitLab

GitLab è una piattaforma completa per la gestione del ciclo di vita del software basato su Git, ideale per progetti di sviluppo software di qualsiasi dimensione. GitLab offre una vasta gamma di funzionalità, tra cui repository Git, issue tracking, integrazione continua e distribuzione continua (CI/CD), gestione delle richieste di pull e molto altro.

Una delle caratteristiche distintive di GitLab è che offre sia una versione self-hosted che una versione basata su cloud, il che significa che puoi scegliere di ospitare il tuo repository e i tuoi strumenti direttamente sul tuo server o utilizzare la versione cloud offerta da GitLab.

Inoltre, GitLab è altamente personalizzabile e offre integrazioni con una vasta gamma di strumenti e servizi di terze parti, rendendo possibile adattare la tua pipeline di sviluppo alle tue esigenze specifiche. In sintesi, GitLab è una potente piattaforma che semplifica il lavoro di collaborazione e gestione dei progetti software, rendendo più efficienti e agili le tue operazioni di sviluppo.

3.3 Angular

Angular è un framework di sviluppo front-end potente e completo che offre una solida base per la creazione di applicazioni web complesse e scalabili. Sviluppato da Google, è diventato uno degli strumenti preferiti dagli sviluppatori per la sua versatilità e le numerose funzionalità che offre.

Una delle caratteristiche distintive di Angular è l'uso di TypeScript come linguaggio di programmazione principale. TypeScript è un superset di JavaScript che introduce il typing statico, le classi e altre funzionalità avanzate. Questo rende il codice più robusto e facilmente manutenibile, fornendo un'esperienza di sviluppo più efficiente e riducendo gli errori.

Il framework si basa sul concetto di componenti, che rappresentano le unità fondamentali dell'interfaccia utente. Ogni componente ha il proprio template HTML, la logica di visualizzazione associata e le proprietà per il binding dei dati. La modularità dei componenti consente una gestione organizzata e riutilizzabile del codice, favorendo la manutenibilità e la scalabilità delle applicazioni.

Un'altra caratteristica potente di Angular è il two-way data binding. Questo meccanismo sincronizza automaticamente i dati tra il modello e la vista, consentendo di riflettere immediatamente le modifiche fatte nell'interfaccia utente nel modello e viceversa. Ciò semplifica notevolmente la gestione dello stato dell'applicazione e migliora l'interattività dell'interfaccia utente.

Angular offre anche un sistema di routing integrato, che consente di gestire la navigazione tra diverse sezioni dell'applicazione senza dover ricaricare completamente la pagina. Questo rende possibile la creazione di applicazioni a pagina singola (SPA) reattive e fluide.

La gestione delle dipendenze è semplificata grazie all'implementazione del design pattern di Dependency Injection (DI). Angular gestisce automaticamente l'iniezione delle dipendenze tra i componenti dell'applicazione, facilitando il riuso del codice, migliorando la modularità e rendendo i test più semplici da scrivere.

Angular è stato progettato tenendo a mente il testing. Fornisce un set completo di strumenti integrati per il testing, come il framework Jasmine per i test unitari

e Protractor per i test end-to-end. Questi strumenti consentono agli sviluppatori di scrivere e eseguire test in modo efficiente, garantendo la qualità e la stabilità dell'applicazione.

Inoltre, Angular offre diverse opzioni per la gestione dello stato dell'applicazione. Con l'utilizzo del servizio RxJS, è possibile gestire in modo efficiente gli eventi asincroni. Inoltre, la libreria di gestione dello stato NgRx offre un approccio reattivo alla gestione dello stato, consentendo di creare applicazioni con uno stato complesso ma facilmente gestibile.

Angular gode di un vasto ecosistema di librerie, strumenti e componenti di terze parti. Ad esempio, Angular Material offre un set di componenti predefiniti e stilizzati che facilitano la creazione di un'interfaccia utente moderna e coerente. Inoltre, la comunità di sviluppatori attiva e dedicata offre una vasta gamma di risorse, tutorial e supporto per aiutare gli sviluppatori a sfruttare al massimo il potenziale di Angular.

Figura 3.3: Logo di Angular

3.4 Postman

Postman è un'applicazione essenziale per gli sviluppatori API, progettata per semplificare il processo di sviluppo, test e documentazione delle API. L'interfaccia utente di Postman è intuitiva e organizzata in modo logico, consentendo agli sviluppatori di creare e testare facilmente richieste API complesse. Grazie alla sua flessibilità, Postman è adatto sia ai neofiti che agli esperti del settore API.

Una delle caratteristiche più potenti di Postman è la sua capacità di automatizzare le richieste API attraverso collezioni di richieste. Questo consente agli sviluppatori di eseguire test completi e ripetibili delle API, migliorando l'efficienza e la precisione del processo di sviluppo.

Inoltre, Postman offre un framework di test integrato che consente agli sviluppatori di definire test per verificare il comportamento dell'API. Questi test possono essere eseguiti automaticamente dopo ogni richiesta, garantendo che l'API funzioni correttamente.

La documentazione automatica generata da Postman è un altro punto di forza. Gli sviluppatori possono generare facilmente documentazione basata sulle richieste effettuate, semplificando la comunicazione e la collaborazione tra team di sviluppo e utenti delle API.

Infine, Postman è un'applicazione altamente collaborativa. Gli sviluppatori possono condividere collezioni di richieste API con altri membri del team, semplificando la collaborazione e migliorando la comunicazione.

Figura 3.4: Logo di Postman

3.5 Swagger

Swagger è una potente suite di strumenti progettata per semplificare il processo di sviluppo, documentazione e testing delle API. Questa suite fornisce una soluzione completa per migliorare il ciclo di vita delle API e favorire una comunicazione efficace tra gli sviluppatori e gli stakeholder.

Una delle caratteristiche chiave di Swagger è il suo editor basato su browser, che rende la progettazione delle API un processo intuitivo e collaborativo. Gli sviluppatori possono definire facilmente endpoint, parametri e schemi dati direttamente dall'editor, riducendo il tempo e lo sforzo necessario per la progettazione delle API.

La generazione automatica di documentazione è un altro punto di forza di Swagger. Questa documentazione è altamente leggibile da parte degli esseri umani e fornisce informazioni dettagliate su come utilizzare l'API. Include esempi di richieste e risposte, contribuendo così a una comprensione chiara e immediata delle funzionalità dell'API.

Swagger offre anche un'interfaccia utente interattiva chiamata "Swagger UI". Questa interfaccia consente agli sviluppatori e agli utenti di testare direttamente le API dal browser, semplificando notevolmente il processo di verifica delle funzionalità dell'API.

3.6 Suite di applicazioni web Ubware

Ubware srl ha sviluppato nel corso degli anni un'applicazione web, scritta usando il framework Angular, che consiste in diversi moduli, alcuni dei quali consentono di gestire e organizzare il lavoro aziendale. L'applicazione è sia usata internamente che venduta ad aziende clienti, che possono richiedere modifiche da applicare ai moduli esistenti o nuovi moduli *custom*

Durante lo stage è stato fatto uso di alcuni di questi moduli, in particolare quelli denominati *Ub4Team* e *Ub4Service*.

- **Ub4Team** È il componente dedicato all'organizzazione del lavoro aziendale. Consente la creazione di eventi e task a calendario e la loro assegnazione a vari dipendenti. Può inoltre tenere traccia del tempo impiegato nello svolgimento di ciascun task assegnato e generare dei report ad esso associati.

- **Ub4Service** È il componente dedicato alla creazione di documenti formali che descrivano la quantità di tempo e risorse impiegate per raggiungere gli obiettivi funzionali stabili in fase di analisi. A differenza di *Ub4Team*, i contenuti così generati non servono per l'analisi interna bensì per essere consegnati al cliente in modo da documentare in maniera trasparente lo svolgimento dei lavori e il relativo costo da sostenere.

In aggiunta a questi strumenti interni, sono stati utilizzati *Microsoft Outlook* e *Microsoft Teams* come supporto alla comunicazione sia interna all'azienda che verso i clienti.

Capitolo 4

Struttura dell'applicativo

4.1 La struttura del codice Python

Ad alto livello, la struttura del codice è determinata da una gerarchia di classi. Le classi base definiscono il comportamento del software in termini astratti, stabilendo una connessione che in linea di principio possa essere applicabile a qualunque tipo di macchinario. Le classi derivate implementano i metodi delle classi superiori in maniera sempre più dettagliata, consentendo allo sviluppatore di scrivere codice appropriato a gestire diversi protocolli e consentendo personalizzazioni con granularità variabile: per tipo di macchinario, per cliente o anche singola macchina.

La classe a base della gerarchia è chiamata *BaseMachine* e ha un metodo principale in cui viene implementato un loop che prosegue nel tempo senza limiti. Questa scelta è dovuta al fatto che in condizioni ordinarie deve ricevere un flusso di dati costante da parte del macchinario. Sono tuttavia presenti meccanismi che consentono al programma di gestire in maniera controllata l'uscita dal loop nel caso si verificassero errori inaspettati, come ad esempio errori di rete che impediscano la connessione fisica tra macchina e server. In tal caso, verrà generato un file log che riporterà informazioni quanto più dettagliate possibile per descrivere la situazione eccezionale verificatasi.

Dal momento che i dati letti vengono generalmente salvati su un database, è legittimo chiedersi come gestire un flusso di dati potenzialmente infinito. Le politiche che regolano il mantenimento dei dati sono decise a livello logico in fase di analisi, in base alle specifiche richieste dal cliente. A livello tecnico, sono implementate con *Trigger* e *Stored Procedure* SQL sul database. Le scelte più semplici e comuni consistono nella definizione di un periodo di tempo oltre il quale i dati possono essere scartati (per esempio, il cliente richiede il mantenimento di tutti i dati ricevuti negli ultimi due anni) o la definizione di un numero massimo di record da mantenere nel database.

Il livello immediatamente inferiore della gerarchia è costituito da una classe chiamata *CustomerMachine*, che nel progetto svolto non sovrascrive alcun metodo di *BaseMachine*. Il ruolo di questa classe sarà discusso ulteriormente nel capitolo successivo.

Dal momento che sul mercato esistono macchinari di diversa produzione che utilizzano protocolli e librerie diverse per comunicare, è utile definire una classe che si occupi di implementare le funzionalità che consentano di gestire le particolarità di ciascuno. Per il progetto assegnato durante lo stage, ci si è concentrati su macchinari di tipo *FANUC*, nome della ditta produttrice.

Se si rendesse necessario adottare ulteriori personalizzazioni per ogni singola macchina da gestire, per esempio su esplicita richiesta della ditta cliente, è possibile espandere

ulteriormente la gerarchia e sovrascrivere i metodi delle classi. Queste nuove classi create ad hoc seguono una nomenclatura codificata in modo da rendere identificabile il macchinario di riferimento così come definito nella documentazione generata in fase di analisi. Spesso il codice identificativo usato per distinguere i macchinari consiste nell'indirizzo IP che identifica la macchina nella sottorete aziendale, e il nome delle classi segue il formato M*IP*.

4.2 La struttura del database

Per quanto riguarda il database, si è scelto di utilizzare un database SQL, data la natura semplice e strutturata dei dati da memorizzare. Il software utilizzato è *MySQL*, in particolare la versione 8. Per interagire con il database direttamente dal codice Python si è fatto uso della libreria *SQLAlchemy*, che si avvale dell' *Object Relational Mapping* per rappresentare la struttura delle tabelle. Di seguito viene elencata la struttura della tabella principale in cui vengono salvati i dati.

Nome	**Descrizione**	**Tipo**
Tiemstamp	Data e ora che identifica il momento in cui il dato è stato ricevuto	DATETIME PRIMARY KEY
ID_Macchina	Un intero utilizzato per identificare i diversi macchinari posseduti dall'azienda cliente che inviano dati	INT NOT NULL
Contapezzi	Il numero di pezzi prodotti dal macchinario	INT NOT NULL
Tempo_Contapezzi	Il tempo trascorso da quando il macchinario ha iniziato la lavorazione del pezzo in corso di preparazione	FLOAT NOT NULL
Programma	I programmi sono file eseguibili salvati nella memoria di un macchinario in una *directory* specifica e identificati da un codice alfanumerico a cinque caratteri rappresentato da questa variabile	VARCHAR(5) NOT NULL
Stato	Una stringa contenente diverse informazioni sullo stato della macchina	VARCHAR(255) DEFAULT NULL

Tabella 4.1: La struttura della tabella '**Dati**' nel database

La scelta della colonna da utilizzare come chiave primaria ricade in maniera naturale sul campo *DATETIME* che rappresenta il *timestamp* di quando una riga di dati è stata ricevuta, dato che è l'unico campo per il quale è garantita l'unicità di ogni riga. Per gli altri campi è stato definito il vincolo *NOT NULL*, poiché si tratta di informazioni essenziali e la mancanza di una di queste equivale, dal punto di vista funzionale, a un errore nella ricezione delle informazioni. L'unica eccezione è costituita dal campo *Stato*, in quanto la sua mancanza non compromette la coerenza degli altri dati. Per ragioni di efficienza, è stato poi creato un indice sulla colonna *ID_Macchina*, che è particolarmente rilevante nelle *query* che vengono inviate al database quando l'utente finale desidera analizzare i dati raccolti.

Di seguito il codice che esegue la mappatura di questa struttura SQL su una classe Python, consentendo di automatizzare le operazioni di lettura e scrittura nel programma.

Listing 4.1: Classe che implementa l'Object Relational Mapping

```
class Data(Base):
        __tablename__ = "Data"
        Timestamp = Column(DateTime, primary_key=True)
        ID_Macchina = Column(Integer, nullable=False)
        Contapezzi = Column(Integer, nullable=False)
        Tempo_Contapezzi = Column(Float, nullable=False)
        Programma = Column(String(5), nullable=False)
        Stato = Column(String(256), default=None )
```

Sono stati poi implementati metodi di classe per gestire la connessione al database, nonchè la lettura e scrittura di una riga nella tabella.

4.3 La struttura delle API

Lo sviluppo previsto per le API prevede l'utilizzo della libreria *FastAPI* e la scrittura di API REST che possano essere chiamate da software di terze parti per accedere al database. Anche in questo caso era già presente un progetto sviluppato precedentemente dall'azienda che mette a disposizione un'ossatura generale basata su una gerarchia di classi per semplificare lo sviluppo di nuovo codice.

È stata prevista l'implementazione di quattro rotte, GET POST PUSH e DELETE, che prendono come parametri i valori da inserire o ricercare all'interno del database. Definite le chiamate, le classi base di alto livello si occupano di gestire le diverse funzioni necessarie a far funzionare l'intera infrastruttura, come la connessione al server e la gestione degli errori.

I software Swagger e Postman vengono utilizzati in fase di test per verificare il corretto funzionamento delle API.

4.4 La WebApp

Ubware sviluppa e mette a disposizione dei clienti una WebApp che consente all'utente finale, anche non esperto a livello tecnico, di leggere i dati memorizzati nel database e scegliere quali programmi fare eseguire ai macchinari. La WebApp è sviluppata tramite il framework Angular, scelto per la sua praticità e flessibilità.

L'interfaccia presentata è semplice e mirata a essere di semplice utilizzo. La schermata principale mostra una tabella che rispecchia quella presente nel database,

consentendo di applicare diversi filtri per meglio selezionare e analizzare i dati. A seconda dei filtri selezionati, le query da sottoporre al database possono variare, lasciando ampio margine per implementare eventuali ottimizzazioni.

Di seguito si riporta un'immagine con un diagramma che descrive la struttura dell'applicativo seguendo il flusso dei dati.

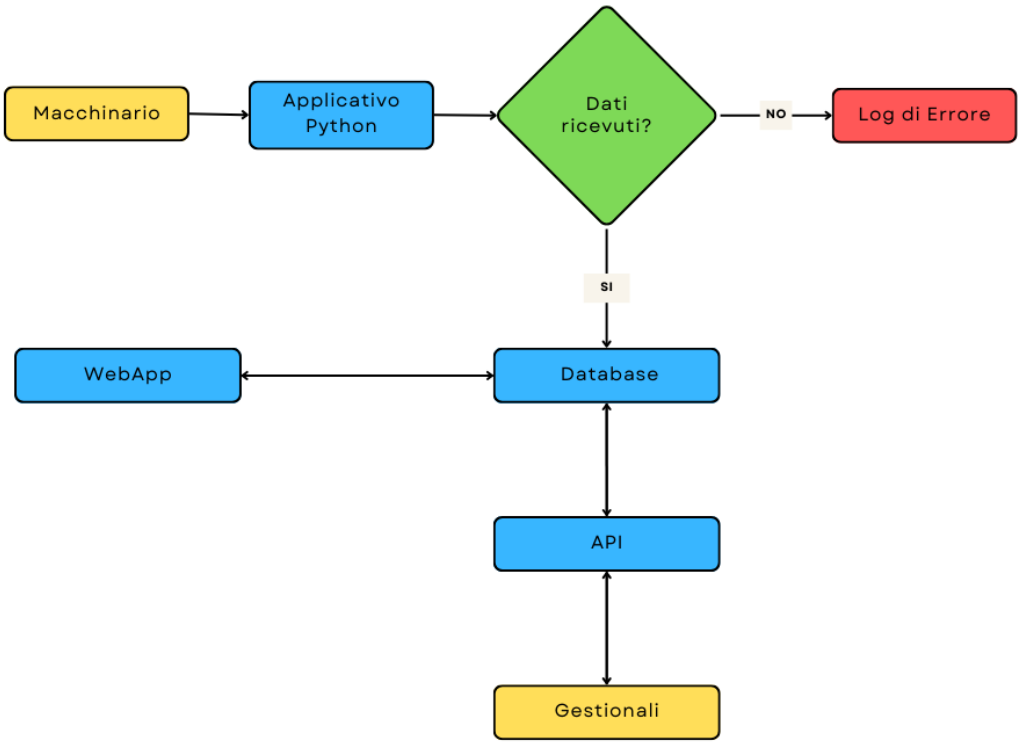


Figura 4.1: Diagramma di flusso dell'intero applicativo

Capitolo 5

Caratteristiche rilevanti del progetto

5.1 La gerarchia di classi

Il *design* dell'applicativo prevede l'utilizzo di una gerarchia di classi per strutturare il codice, che corrispondono a diversi livelli di astrazione. Le classi sono ordinate dalla più generica alla più specifica:

1. BaseMachine
2. CustomerMachine
3. *Protocol*Machine (Da rinominare in base al protocollo utilizzato)
4. *Number*Machine (Da rinominare in base al codice identificativo della singola macchina)

BaseMachine contiene metodi astratti per interrogare il macchinario sullo stato di esecuzione e implementa il loop generale che rimane in esecuzione durante tutto il periodo in cui il programma è attivo.

CustomerMachine è una classe vuota che può essere implementata se necessario, con lo scopo di offrire un'interfaccia per gestire personalizzazioni a livello di ditta cliente, se necessario. Nell'ambito dello stage ciò non si è rivelato necessario, ma è importante notare come la presenza della classe nella gerarchia non sia superflua, in quanto lo scopo è di rendere lo sviluppo più semplice e rapido nei casi in cui l'utilizzo della classe sia necessario.

*Protocol*Machine gestisce invece il layer che riguarda il protocollo di comunicazione utilizzato. Nel caso del progetto svolto la classe creata è stata la denominata FanucMachine.

Infine, le classi identificate da un numero sono quelle che implementano i comportamenti specifici di ogni singolo macchinario.

La struttura è dunque studiata per poter generare codice mantenibile, garantendo un elevato livello di flessibilità. Inoltre, se in futuro dovesse essere necessario produrre un programma analogo per una nuova ditta cliente, non sarebbe difficile riutilizzare buona parte della base di codice esistente e operare le dovute modifiche al livello opportuno della gerarchia, senza intaccare il resto delle classi. Questo consente agli sviluppatori di essere molto efficienti nel rilascio del codice.

5.2 La lettura dei dati dal macchinario

Per comunicare con macchinari industriali di tipo *FANUC* è stata reperita, in fase di analisi, una libreria scritta in *C* liberamente disponibile. Dal momento che il linguaggio scelto per la scrittura del programma è Python, per motivi di integrazione e compatibilità con altri applicativi sviluppati a livello aziendale, si è fatto ricorso al module *cytpes*.

Il modulo ctypes è una libreria di Python che offre un'interfaccia per chiamare funzioni di librerie condivise scritte in C e C++. È ampiamente utilizzato per l'integrazione di librerie di sistema o librerie scritte in linguaggi nativi all'interno di programmi Python. Una delle caratteristiche distintive di ctypes è la sua capacità di creare oggetti Python che possono essere utilizzati per interagire con le funzioni e i dati delle librerie condivise. Ciò significa che è possibile definire strutture di dati in Python che corrispondono alle strutture in C e utilizzarle per passare dati tra il codice Python e il codice C. Questa flessibilità consente agli sviluppatori di accedere facilmente a funzionalità avanzate fornite da librerie esterne senza dover riscrivere l'intera applicazione in C o C++. Inoltre, ctypes supporta la gestione dei tipi di dati, il che significa che è possibile specificare i tipi di dati degli argomenti delle funzioni C e delle variabili condivise, garantendo una corretta allocazione e gestione della memoria. Un'altra caratteristica interessante è la capacità di caricare dinamicamente le librerie condivise, il che significa che è possibile utilizzare ctypes per eseguire il binding con librerie esterne a tempo di esecuzione, rendendo il codice più flessibile e adattabile.

Sono state dunque create le classi Python corrispondenti alle strutture definite nella libreria *C*. Si prenda come esempio la seguente struttura, chiamata **ODBSYS**.

Listing 5.1: La struttura ODBSYS in C secondo la documentazione

```
typedef struct odbsys {
    short   addinfo ;      /* additional information */
    short   max_axis ;     /* maximum axis number */
    char    cnc_type[2] ;  /* cnc type <ascii char> */
    char    mt_type[2] ;   /* M/T/TT <ascii char> */
    char    series[4] ;    /* series NO. <ascii char> */
    char    version[4] ;   /* version NO.<ascii char> */
    char    axes[2] ;      /* axis number<ascii char> */
} ODBSYS ;
```

Listing 5.2: La corrispondente classe IODBPSD in Python

```
class ODBSYS(ctypes.Structure):
        _fields_ = [("addinfo",ctypes.c_short), ("max_axis",
            ctypes.c_char * 2), ("cnc_type",ctypes.c_char * 2),
            ("mt_type",ctypes.c_char * 2), ("series",ctypes.
            c_char * 4), ("version",ctypes.c_char * 4), ("axes",
            ctypes.c_char * 2)]
```

Le strutture dati presenti in queste classi rispecchiano quelle interne ai macchinari. Gli oggetti appartenenti a una classe così definita possono essere passati come parametri in funzioni C, chiamate tramite l'ausilio di *ctypes*. Le funzioni della libreria che consentono di ricevere dati relativi allo stato della macchina richiedono un oggetto appartenente alla classe passato per riferimento, i cui campi vengono riempiti con i dati reali forniti dalla macchina.

In questo contesto si inserisce una delle prime attività svolte nell'ambito del progetto. Alcune analisi svolte in via preliminare dall'azienda, prima dell'inizio del progetto, hanno riscontrato difficoltà nella lettura di un dato specifico, ovvero la quantità di pezzi prodotti dal macchinario. Il campo di tipo intero che avrebbe dovuto contenere questa informazione risulta infatti contenere il valore 0 dopo ogni chiamata, valore chiaramente incoerente con quanto atteso. È stato dunque necessario compiere un'attività di analisi e test per cercare di comprendere quale fosse il problema e porvi rimedio.

La struttura dati alla radice del problema è la seguente, denominata **IODBPSD**.

Listing 5.3: La struttura IODBPSD in C secondo la documentazione

```
typedef struct iodbpsd {
    short  datano ;   /* data number */
    short  type ;     /* axis number */
    union {
        char   cdata ;    /* parameter / setting data */
        short  idata ;
        long   ldata ;
        char   cdatas[MAX_AXIS] ;
        short  idatas[MAX_AXIS] ;
        long   ldatas[MAX_AXIS] ;
    } u ;
} IODBPSD ;
```

La struttura è costituita da due campi *short* ciascuno dei quali occupa 2 *byte* in memoria, e un'unione, la quale occupa uno spazio in memoria pari alla dimensione del membro più grande, in questo caso un *long* di 8 byte. La dimensione totale della struttura in memoria è dunque di 12 byte. I due campi short contengono parametri tecnici per identificare il dato letto, che è esso stesso contenuto all'interno dell'unione.

Il codice utilizzato per riempire una struttura dato IODBPSD con i valori adeguati è il seguente:

Listing 5.4: Chiamata alla funzione $\text{cnc}_r dparam3$

```
item_count_struct = IODBPSD()

ret = focas.cnc_rdparam3(self.handle, 6711, 0, 8, 0, ctypes.
    byref(item_count_struct));
```

Inizialmente viene creato un oggetto della classe IODBPSD. Successivamente, viene chiamata la funzione di libreria *cnc_ rdparam3*. Questa funzione prende come parametri la struttura da riempire, passata per riferimento, e una serie di altri parametri descritti nella documentazione utili a specificare quale insieme di dati si desidera leggere. In particolare, il parametro 6711 indica l'area di memoria da leggere nel macchinario. Analizzando il contenuto della struttura dopo la chiamata emerge che il campo che dovrebbe contenere il numero di pezzi prodotti non sembra contenere il valore atteso. Tuttavia, test simili su altri dati danno esito positivo, indicando quindi che il problema concerne questo tipo di dato nello specifico.

Per cercare di risolvere questa difficoltà si è deciso di leggere il contenuto dell'intera area di memoria della struttura, scrivendolo in output su un apposito file fi log. Analizzando il risultato, si è scoperto che un valore numerico veniva effettivamente

scritto nell'area di memoria riservata all'unione, ma come un intero di 4 byte, non corrispondente ai campi dati attesi.

Un esempio array di byte letta è il seguente:

Listing 5.5: Byte grezzi ricevuti

```
0x37 0x1A 0x00 0x00 0x42 0x1E 0x00 0x00 0x00 0x00 0x00 0x00
```

I primi due byte appartengono al campo dati *'datano'*, e contiene un valore numerico diverso da 0. La seconda coppia di byte è nulla e rappresenta il tipo del dato (che in effetti sembra essere errato in quanto non coerente con quello atteso). Vi è poi la sequenza di byte dell'unione.

Analizzando i primi due byte, ci si può accorgere che interpretandoli come un campo short codificato nella rappresentazione *little endian* rappresentano il numero 6711, che in effetti corrisponde al parametro utilizzato nella funzione. Consci di questa informazione si è deciso di fare lo stesso per il numero salvato nell'area di memoria dell'unione, scoprendo che corrisponde al valore di un intero a quattro byte che in decimale corrisponde a 7746. Una rapida verifica ha dimostrato come questo valore fosse quello corretto.

Consapevoli che il dato fosse effettivamente presente, dunque, si è deciso di 'estrarlo' in maniera automatica scrivendo del codice specifico per questa operazione. Si è dunque implementata una funzione che, leggendo il contenuto dell'unione come array di byte, ricostruisce il valore intero e lo assegna a una nuova variabile.

Listing 5.6: La funzione exctratcInt

```
def exctractInt(obj: bytes, start: int, stop: int, endianness:
    str = "little") -> int:
        n = ctypes.c_int()
        n = bytearray(obj)
        n = n[start:stop]
        n = int.from_bytes(n, endianness)
        return n
```

Combinando dunque la funzione di libreria con questa così definita, è stato risolto il problema descritto nel capitolo introduttivo ed è possibile portare a termine il progetto. La funzione per estrarre il dato completo è stata di conseguenza aggiornata nel seguente modo.

Listing 5.7: Estrazione del dato che indica il numero di pezzi prodotti

```
def readItemCount(self) -> int:

        item_count_struct = IODBPSD()

        ret = focas.cnc_rdparam3(self.handle, 6711, 0, 8, 0,
            ctypes.byref(item_count_struct));

        if ret != ErrorCodes.EW_OK:
                raise LibraryError(ret, "readItemCount()
                    cnc_rdparam3")
        else:
                item_count = exctractInt(item_count_struct, 4, 8)
```

```
                return item_count
```

5.3 La struttura dei progetti su GitLab

Ubware dispone di un server dedicato sul quale viene salvato il codice sviluppato, gestito tramite il sistema di versionamento git e la piattaforma GitLab. Questo consente agli sviluppatori di collaborare in maniera efficace, mantenendo l'intero storico delle versioni del codice prodotte. Strumenti come GitLab consentono anche di rendere più efficace e sicuro lo sviluppo consentendo la revisione di ogni operazione di commit da parte di altri programmatori, che possono identificare potenziali criticità e proporre miglioramenti. Ciò è utile a ridurre la presenza di bug, nonchè a migliorare la qualità del codice per quanto riguarda l'efficienza e la leggibilità.

Una delle caratteristiche principali di git è la facilità con cui consente di creare nuovi *branch*. Questo consente l'impiego di diversi *workflow*, ovvero metodi di lavoro che fanno uso delle potenzialità di questo software.

Il metodo di sviluppo adottato in Ubware consiste nella creazione di due branch principali, *master* e *dev*, per ciascun progetto. Avere un branch *master* e un branch *dev* nel workflow di Git rappresenta un approccio organizzato e strategico alla gestione del ciclo di sviluppo del software. Il branch *dev* funge da cuore pulsante dell'area di sviluppo, dove gli sviluppatori lavorano attivamente per implementare nuove funzionalità, correzioni di bug e miglioramenti. Questo ambiente di lavoro dedicato consente una collaborazione fluida tra membri del team, ognuno concentrato su specifici compiti senza intaccare la stabilità del software principale. D'altro canto, il branch *master* rappresenta la versione stabile e consolidata del software. Qui confluiscono solo le modifiche che hanno superato una serie di test rigorosi nel branch *dev*. Questa pratica separazione garantisce che il software principale sia sempre affidabile e pronto per il rilascio senza sorprese indesiderate. Inoltre, la presenza di un branch *dev* consente un controllo più efficace sulla qualità del codice. Gli sviluppatori possono condurre test approfonditi e revisioni del codice prima che le modifiche vengano integrate nel branch principale. Questo contribuisce a individuare e risolvere i problemi in modo proattivo, riducendo il rischio di bug nei rilasci. Quando il codice preparato all'interno dell'ambiente di sviluppo viene ritenuto pronto per passare in produzione, si può compiere un'operazione di *merge* verso il branch *master*.

Inoltre, in alcuni dei progetti sviluppati, come quello di interesse nel caso dello stage curricolare svolto, è possibile avere ulteriori branch contenenti il codice che riguarda uno specifico cliente. Questo permette di gestire in maniera efficace le personalizzazioni richieste da clienti diversi, mantenendo il codice ben separato e organizzato. Il branch di ogni cliente avrà a sua volta un secondo branch ausiliario di sviluppo.

Dal momento che, all'interno di un singolo progetto, i branch 'secondari' che riguardano i diversi clienti condividono una larga parte del codice, è possibile anche che si voglia aggiornare la parte comune a tutti, a prescindere dalle personalizzazioni. In questo caso, lo sviluppo avviene sul branch principale, e in un momento successivo le modifiche saranno propagate alle sezioni riguardanti i singoli clienti effettuando un'operazione di *merge* questa volta dal branch principale verso quelli secondari. Si noti che parte di questo processo può essere automatizzato con l'utilizzo di tool e

script, benché eventuali conflitti in fase di *merge* debbano necessariamente essere risolti tramite un intervento manuale.

Infine, è opportuno notare che quando un programmatore clona in locale un branch per iniziare a lavorare, ha la libertà di creare altri branch a proprio piacimento, se questo dovesse semplificare o aiutare il processo di sviluppo. Al termine, verrà creata una *pull request* chiedendo di integrare il branch di sviluppo con le modifiche apportate con quello attualmente presente sulla repository remota.

La seguente immagine rappresenta il processo qui descritto, immaginando un branch principale e due branch per clienti denominati 'ELAS STAMPI' e 'TAUMAT'.

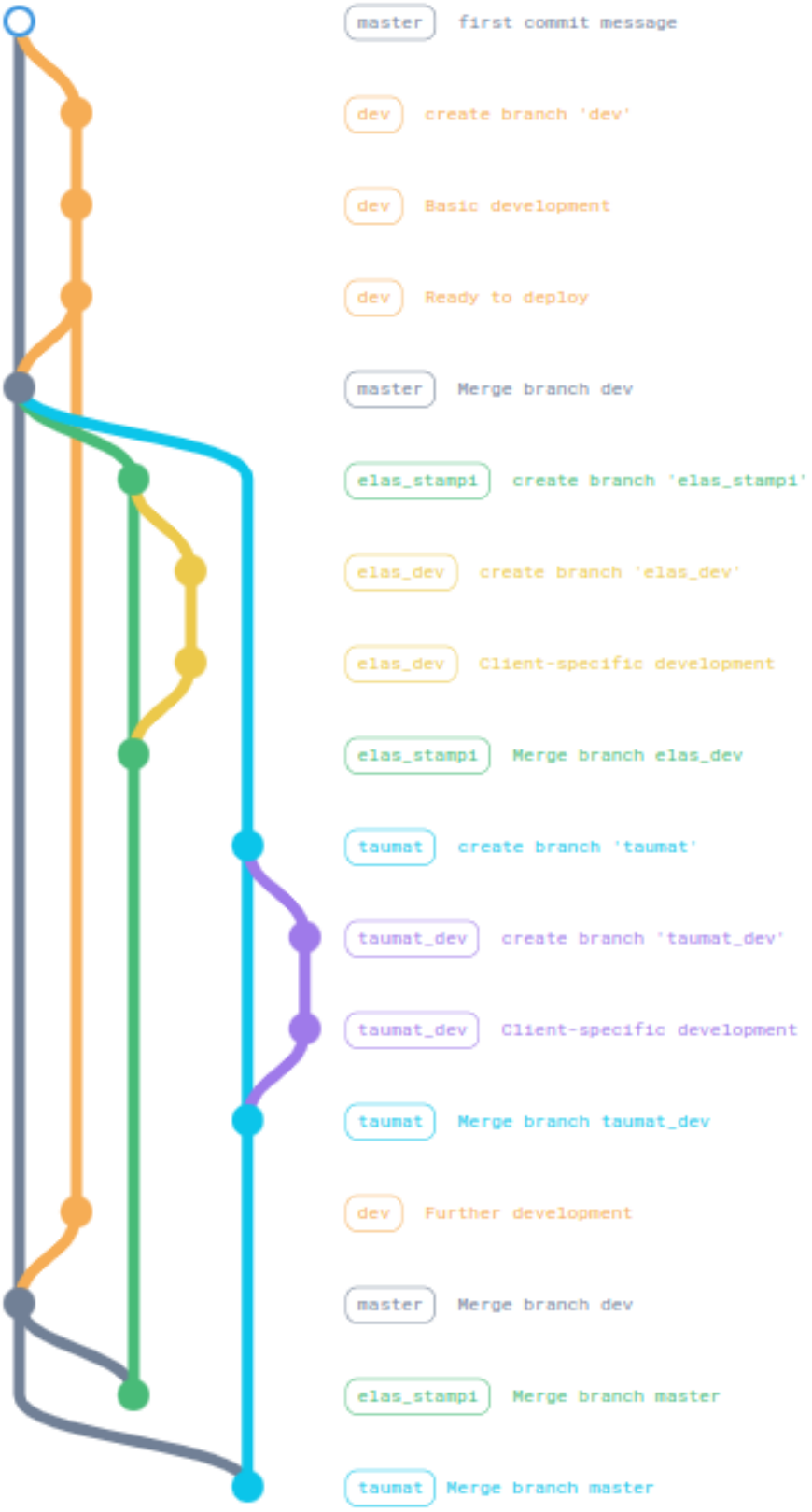


Figura 5.1: Git workflow

Capitolo 6

Test

Nel contesto del processo di sviluppo del progetto in questione, è stato essenziale dedicare particolare attenzione all'aspetto del testing e della validazione. L'obiettivo principale di questa fase è stato assicurare che il software risultante fosse affidabile, privo di difetti e allineato con le specifiche richieste.

In questo capitolo, verranno esaminati i vari aspetti di questo processo di validazione, con un focus particolare sulla metodologia adottata e sui risultati ottenuti. Ogni fase del processo di validazione è stata strutturata in modo rigoroso e meticoloso, al fine di garantire una copertura completa e approfondita di tutte le funzionalità e i componenti del software.

I test si sono svolti sia in ambiente di sviluppo, verificando la correttezza delle singole funzioni, sia in un ambiente adeguatamente preparato, facendo eseguire il codice sul server che sarebbe stato effettivamente utilizzato anche in produzione, utilizzando però un database separato esplicitamente adibito allo scopo.

6.1 Obiettivi

L'obiettivo prefissato è stato quello di assicurare, con il maggior livello di confidenza possibile, il corretto funzionamento del programma una volta rilasciato e la possibilità per il cliente di poter fruire dei dati raccolti. Per raggiungere questi obiettivi è stato previsto un periodo consistente dello stage da dedicare all'attività di testing, e durante l'analisi si è definito di effettuare sia test di unità che test che potessero verificare l'integrità del codice nel suo insieme in un ambiente opportunamente preparato e descritto in seguito.

6.2 Unit test

Nella fase di sviluppo degli unit test, è stato adottato un approccio rigoroso e strutturato per garantire la copertura completa del codice e la validazione di ogni componente del sistema. Questo processo è stato fondamentale per garantire la qualità e l'affidabilità del software.

Inizialmente, è stato necessario identificare le unità di codice che richiedevano test. Ogni modulo, funzione o metodo è stato considerato come un'unità di testabile.

I test sono stati scritti tenendo conto delle *best practice* utilizzate nell'industria, che includono:

- Indipendenza: Uno unit test dovrebbe essere indipendente dagli altri test e dall'ordine di esecuzione. Non dovrebbe dipendere da uno stato globale o da dati condivisi tra test.
- Isolamento: Uno unit test dovrebbe concentrarsi solo sull'unità di codice che sta testando. Tutte le dipendenze esterne dovrebbero essere sostituite da doppi (mocks o stubs) per isolare l'unità in esame.
- Ripetibilità: Uno unit test dovrebbe essere ripetibile e produrre risultati consistenti. Deve essere possibile eseguirlo più volte ottenendo lo stesso risultato.
- Automatizzazione: Un buon unit test deve essere automatizzato e facilmente eseguibile. L'automatizzazione permette di eseguire i test frequentemente e rapidamente.
- Determinismo: Il test deve essere deterministico, cioè deve produrre sempre lo stesso risultato per la stessa unità di codice.
- Copertura: L'unit test dovrebbe coprire tutti i percorsi del codice sorgente, inclusi i casi limite e le situazioni di errore.
- Leggibilità: Il test deve essere facilmente comprensibile. Il nome del test e gli assert dovrebbero rendere chiaro ciò che il test sta verificando.
- Velocità: I test dovrebbero essere veloci da eseguire. I test lenti possono rallentare il processo di sviluppo e scoraggiare il loro utilizzo frequente.
- Manutenibilità: I test dovrebbero essere facili da mantenere e aggiornare in caso di modifiche al codice sorgente.
- Segnalazione di Errori: I test dovrebbero essere in grado di rilevare e segnalare errori nel codice in modo chiaro e informativo, consentendo una rapida correzione.
- Controllabilità: I test dovrebbero essere controllabili e in grado di simulare diverse situazioni, inclusi casi di errore, senza dipendere da fattori esterni.
- Scalabilità: I test dovrebbero essere scalabili per adattarsi a progetti di diverse dimensioni e complessità.
- Aggiornamento costante: I test dovrebbero essere mantenuti e aggiornati regolarmente per riflettere le modifiche apportate al codice sorgente.
- Documentazione: I test dovrebbero includere una documentazione chiara e significativa che spieghi lo scopo del test e le condizioni previste.

Una volta scritti gli unit test, è possibile eseguirli in maniera automatica, caratteristica che rende possibile eseguirli ogni qualvolta siano apportate modifiche al codice, verificando che i test rimangano soddisfatti. Ciò ha consentito di individuare tempestivamente eventuali regressioni o errori e ha contribuito a mantenere un alto standard di qualità del codice. Inoltre, è stato istituito un processo di revisione dei test da parte dei colleghi sviluppatori.

6.3 Test sui dati reali

L'ambiente di test è stato preparato sul server allestito dal reparto sistemistico di Ubware. Sul server è stata installata una macchina virtuale che replica l'ambiente di sviluppo. È stata inoltre preparata una VPN per garantire l'accesso sicuro alla rete aziendale della ditta cliente.

Sono stati dunque svolti una serie di test per verificare le diverse operazioni che il software può eseguire, utilizzando macchinari effettivamente in uso. La differenza principale rispetto al rilascio finale del codice consiste nel fatto di utilizzare un diverso database, risultando quindi molto simile all'ambiente finale di produzione.

Sono inoltre stati utilizzati Postman e l'interfaccia di Swagger per testare la parte di API che è stata implementata.

I test svolti hanno dato esito positivo, e non sono state riscontrate particolari criticità.

6.4 Monitoraggio in produzione

Questa fase è stata pianificata nell'ambito dello stage, anche se non eseguita data la conclusione dello stesso. Sono stati previsti dei controlli ad intervalli di tempo definiti, concordati con i clienti, che hanno inoltre la possibilità di contattare Ubware non appena dovessero notare delle anomalie. È stato inoltre implementato uno script scritto in *bash* ed eseguito tramite il tool *cron* sul server linux ove è situato il database per controllare che la ricezione dei dati sia costante. Nel caso i cui il flusso di dati in entrata dovesse essere interrotto, sarà inviata automaticamente una mail al reparto sviluppo per notificare la situazione.

Capitolo 7

Conclusioni

7.1 Consuntivo finale

Di seguito la tabella contenente le ore previste per portare a termine ciascun requisito e il confronto con le ore effettivamente richieste.

Codice	Ore Previste	Ore Effettive
RO-1	80	90
RO-1.1	40	45
RO-1.2	40	45
RO-2	80	65
RO-2.1	40	35
RO-2.2	40	25
RO-3	80	85
RD-1	40	40 - non completato
RF-1	40	45 - non completato

Tabella 7.1: Consuntivo

7.2 Raggiungimento degli obiettivi

I requisiti obbligatori previsti dal progetto sono stati realizzati. Il tempo pianificato per la formazione iniziale e la realizzazione del programma in Python si è rivelato essere adeguato. Sarebbe stato possibile, avendo ulteriore tempo a disposizione, migliorare ulteriormente l'efficienza delle chiamate al database effettuate dal codice della WebApp e completare la realizzazione e il testing delle API.

L'applicativo prodotto è un software che, una volta caricato sul server di produzione, stabilisce una connessione con il macchinario prescelto e avvia uno scambio di dati. A intervalli regolari, che possono essere stabiliti e modificati tramite un apposito parametro, viene effettuata una lettura completa dello stato della macchina. Tutti i

dati raccolti vengono automaticamente caricati su un apposito database SQL, creando un nuovo record nella tabella dedicata. Eventuali errori di connessione o di altro tipo vengono gestiti opportunamente, creando log da consultare per verificare il problema e notificando il team di sviluppo. L'utente finale ha la possibilità di consultare i dati raccolti in modo semplice ed efficace tramite un'interfaccia web pensata per rendere la lettura del database accessibile a personale non tecnico.

Una volta terminate, le API consentiranno anche a software gestionali di terze parte di accedere ai dati, previa opportuna identificazione, permettendo di automatizzare ulteriormente i processi di analisi dei dati e la produttività delle aziende clienti. La seguente tabella riassume lo stato di completamento dei requisiti elencati nel capitolo 2.

Codice	**Descrizione**
RO-1	Implementato
RO-1.1	Implementato
RO-1.2	Implementato
RO-2	Implementato
RO-2.1	Implementato
RO-2.2	Implementato
RO-3	Implementato
RO-3.1	Implementato
RO-3.2	Implementato
RD-1	Parzialmente implementato
RF-1	Parzialmente implementato

Tabella 7.2: Implementazione requisiti

7.3 Sviluppi futuri

Il lavoro svolto può essere ampliato e migliorato in diversi modi. Innanzitutto è possibile portare a termine in maniera esaustiva i requisiti desiderabili e opzionali che sono stati solo parzialmente soddisfatti, principalmente completare il processo di sviluppo e test delle API. Si può anche analizzare e cercare di affinare ulteriormente la robustezza e l'efficienza del codice prodotto. Inoltre, grazie alla libreria sviluppata nel corso del progetto per la comunicazione coi macchinari *FANUC*, sarà possibile in futuro sviluppare progetti simili per nuovi macchinari in tempi molto brevi, riutilizzando il codice consegnato all'azienda. Infine, vi è sempre la possibilità di sviluppare nuove librerie *ad hoc* per nuovi protocolli di comunicazione così da avere pronta una codebase più ampia e consentire il rilascio molto rapido di questo tipo di applicativi per un gruppo più ampio di macchinari.

7.4 Conoscenze acquisite e valutazione personale

Lo stage svolto ha avuto esito estremamente proficuo sia dal punto di vista accademico che dal punto di vista professionale. Mi ha consentito di applicare in un contesto pratico le conoscenze teoriche acquisite durante il percorso accademico, ampliandole grazie al lavoro di ricerca necessario per superare le difficoltà incontrate durante il percorso.

La possibilità di sperimentare in prima persona i metodi di lavoro applicati in un reale contesto lavorativo è stata preziosa per la mia crescita professionale. Ho acquisito maggiore confidenza con gli strumenti di sviluppo, come IDE e debugger, e con strumenti collaborativi, in particolare Git e il suo utilizzo all'interno di un team. Ho avuto modo di confrontarmi con la necessità di apprendere l'utilizzo di framework e librerie nuove in breve tempo, capacità molto utile e apprezzata in ambito lavorativo.

Il progetto è stato interessante anche perché mi ha consentito di integrare ambiti diversi dell'informatica, dalla programmazione ad oggetti alle basi di dati e allo sviluppo web. Inoltre, durante i mesi trascorsi all'interno dell'azienda ho avuto modo di interessarmi ai diversi prodotti software che essa sviluppa anche al di fuori del progetto del mio stage, che spaziano dallo sviluppo web allo sviluppo di reti neurali.

Acronimi

API Application Program Interface

IDE Integrated Development Environment

ORM Object Relational Mapping

Glossario

Angular Angular è un framework di sviluppo front-end open source che consente agli sviluppatori di creare applicazioni web dinamiche e ricche di funzionalità. Basato su TypeScript, Angular offre un'architettura modulare e un'ampia gamma di strumenti per la gestione dello stato, il routing e l'interazione con i servizi back-end. Uno dei punti di forza di Angular è la sua capacità di creare interfacce utente responsive e complesse attraverso il concetto di componenti riutilizzabili. Inoltre, Angular offre una solida suite di strumenti per il testing automatizzato, garantendo la qualità del codice e facilitando il processo di sviluppo.

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione.

Branch Un branch in Git è una linea di sviluppo separata dalla branch principale (solitamente chiamata "master") che permette di lavorare su nuove funzionalità o correzioni di bug in modo isolato. Ogni branch contiene una copia dei file del progetto e consente agli sviluppatori di lavorare indipendentemente l'uno dall'altro. Una volta completato il lavoro su un branch, è possibile unire le modifiche nella branch principale attraverso un processo chiamato "merge".

C Il linguaggio di programmazione C è uno strumento informatico fondamentale, noto per la sua semplicità e potenza. Creato nei primi anni '70 da Dennis Ritchie, C offre un controllo dettagliato sull'hardware e una portabilità tra diverse piattaforme. È una scelta comune per lo sviluppo di software di sistema, embedded e applicazioni multipiattaforma. La sua influenza nella storia dell'informatica è indiscutibile, ed è ancora ampiamente utilizzato oggi.

ctypes Il modulo Python ctypes è una libreria integrata che permette agli sviluppatori di chiamare funzioni C esistenti e utilizzare librerie condivise in linguaggio C direttamente da Python. Questo modulo offre una via di comunicazione tra il codice Python e librerie scritte in C, rendendo possibile l'accesso a risorse di sistema e funzionalità non disponibili direttamente in Python. È spesso utilizzato per l'integrazione di librerie esterne e per l'ottimizzazione di prestazioni in applicazioni Python.

MySQL MySQL è un sistema di gestione di database relazionali open-source ampiamente utilizzato. È noto per la sua affidabilità, velocità e facilità d'uso ed

è utilizzato in una vasta gamma di applicazioni, dalla gestione di piccoli database personali a sistemi aziendali su larga scala. MySQL supporta il linguaggio SQL per l'interazione con il database ed è compatibile con molte piattaforme e linguaggi di programmazione.

FANUC FANUC è il nome di un gruppo di aziende giapponese che si occupano della costruzione di macchinare e robot industriali. Nel contesto di questa tesi il termine si riferisce anche al protocollo di comunicazione utilizzato dai macchinari prodotti dalle aziende del gruppo.

Git Git è un sistema di controllo versione distribuito ampiamente utilizzato per il tracciamento delle modifiche nel codice sorgente. Consente ai team di sviluppo di collaborare in modo efficiente, tenere traccia delle versioni del codice e risolvere conflitti durante il processo di sviluppo software. Git è noto per la sua flessibilità, velocità e capacità di gestire progetti di qualsiasi dimensione.

GitLab GitLab è una piattaforma di gestione del ciclo di vita del software basata su Git, che offre repository Git, issue tracking, integrazione continua e distribuzione continua (CI/CD) e molto altro. È disponibile sia come soluzione self-hosted che come servizio basato su cloud, offrendo flessibilità nella gestione dei tuoi progetti software. Con la sua ampia personalizzazione e integrazioni con strumenti di terze parti, GitLab è una scelta versatile per semplificare il tuo processo di sviluppo software.

IDE Un Integrated Development Environment (IDE) è un ambiente di sviluppo software completo che offre agli sviluppatori una vasta gamma di strumenti per scrivere, testare e gestire il codice delle applicazioni. Uno dei principali vantaggi di un IDE è il suo avanzato editor di codice, che fornisce funzionalità di evidenziazione della sintassi, completamento automatico e refactoring per aumentare la produttività nella scrittura del codice. Inoltre, un IDE offre potenti strumenti di debugging che aiutano gli sviluppatori a individuare e risolvere rapidamente gli errori nel codice. Questi strumenti forniscono un'analisi approfondita del flusso di esecuzione del programma e consentono di monitorare lo stato delle variabili in tempo reale, semplificando il processo di correzione degli errori. Gli IDE includono anche funzionalità di gestione dei progetti, che consentono agli sviluppatori di organizzare il codice in modo logico, facilitando la navigazione tra i file e la gestione delle risorse. Inoltre, molti IDE supportano l'integrazione con sistemi di controllo versione come Git, semplificando il monitoraggio delle modifiche al codice.

ORM L'Object-Relational Mapping (ORM) è una tecnica di sviluppo software che consente di mappare gli oggetti delle applicazioni direttamente alle tabelle di un database relazionale. Questo approccio semplifica l'interazione tra il codice dell'applicazione e il database, consentendo agli sviluppatori di utilizzare oggetti e classi orientati agli oggetti anziché scrivere query SQL manualmente. L'ORM automatizza gran parte del lavoro di accesso ai dati e semplifica lo sviluppo delle applicazioni che utilizzano database relazionali.

Postman Postman è una potente piattaforma di sviluppo e test delle API che consente agli sviluppatori di semplificare il processo di creazione, testing e documentazione delle API. Questa applicazione offre un'interfaccia utente intuitiva che permette di inviare richieste HTTP alle API e ricevere risposte

in tempo reale, facilitando il debug e il controllo delle API. Inoltre, Postman fornisce strumenti per la generazione automatica della documentazione delle API, rendendo più agevole la condivisione delle informazioni sull'utilizzo delle API con il team di sviluppo e gli utenti.

Stored Procedure Una stored procedure è un insieme di istruzioni SQL predefinite e memorizzate nel database. Queste procedure possono essere richiamate da applicazioni o altre query SQL per eseguire operazioni specifiche sul database, consentendo la riutilizzazione del codice e aumentando l'efficienza delle operazioni database. Le stored procedure sono spesso utilizzate per migliorare la gestione dei dati e per eseguire azioni complesse all'interno di un database SQL.

Swagger Swagger è una potente serie di strumenti per progettare, costruire e documentare API RESTful. Questa piattaforma consente agli sviluppatori di definire facilmente le specifiche delle loro API e generare automaticamente documentazione interattiva basata su queste specifiche. Swagger favorisce la comunicazione tra sviluppatori, semplifica la creazione di API robuste e promuove una migliore comprensione delle risorse API attraverso una documentazione chiara e strutturata.

Trigger Un trigger in un database SQL è una procedura automatica che si attiva in risposta a eventi specifici, come l'inserimento, l'aggiornamento o la cancellazione di dati in una tabella. Questo strumento è utilizzato per garantire l'integrità dei dati e per eseguire azioni predefinite quando si verificano determinate condizioni. I trigger sono ampiamente utilizzati per automatizzare processi e per eseguire operazioni personalizzate all'interno di un database SQL.

Bibliografia

Test consultati

- **The Object-Oriented Thought Process** Matt Weisfeld, 2019
- **Fluent Python, Second Edition** Luciano Ramalho, 2022
- **ProGit** Scott Chacon - Ben Straub, 2009

Siti Web consultati

- **Documentazione libreria focas per l'interazione coi macchinari fanuc** https://www.inventcom.net
- **Documentazione funzione CNC_RDPARAM3** https://www.inventcom.net/fanuc-focas-library/ncdata/cnc_rdparam3
- **Documentazione GitLab** https://docs.gitlab.com/
- **Documentazione Angular** https://angular.io/docs
- **Documentazione Swagger** https://swagger.io/docs/