DOCUMENTAZIONE PROJECT WORK 2021

BASSO ANDREA, BESCHIN PIETRO, FACCIOLI SAMUEL, VOLPATO CRISTIAN

INDICE DEI CONTENUTI

PREMESSE E CONSIDERAZIONI

Per la gestione remota di una linea di preparazione di flaconcini medicali sono stati realizzati diversi componenti interconnessi tra loro (vedi schema). Il sistema di preparazione di flaconcini è gestito da un dispositivo PLC, il quale si interfaccia ad un'applicazione esterna scritta in C#, utilizzata per l'inserimento dei dati. Per adempiere all'esigenza di una dashboard per la visualizzazione delle commesse, è stata creata una pagina web che si appoggia a un server in cloud (AWS) contenente un servizio Apache Web Server. I dati visualizzati sono ricavati da API fornite da un servizio NodeJS presente sulla stessa macchina. Per la comunicazione tra quest'ultima e il dispositivo PLC è stato sviluppato un servizio in C# che inoltra i dati in formato JSON con una chiamata tramite metodo POST. Tali dati sono storicizzati in un database basato su MongoDB, posto su un'ulteriore host in cloud. Inoltre, con il fine di espandere il range di device su cui è possibile visualizzare le informazioni riguardanti la linea di produzione, è stato creato un bot telegram basato su uno script Python, eseguito da un'istanza in cloud.
È stato scelto di utilizzare un database locale Microsoft SQL per l'utilizzo dell'applicazione dell'ufficio. Tale scelta è stata determinata dall'esigenza di avere un database in locale, disponibile anche in assenza di connessione internet, utilizzabile dall'applicativo anche come storage di backup. In alternativa, sarebbe stato possibile utilizzare uno o più file in formato JSON.
Si ritiene opportuno specificare che, per una strutturazione dell'applicativo più completa e per la fase di presentazione del progetto, nel database sono stati inserite le entità "cliente" e "prodotto", ma che non competerebbe all'applicativo di gestione l'amministrazione delle stesse.

Per quanto riguarda le tempistiche del progetto, si è ritenuto necessario svolgere le varie mansioni contemporaneamente, suddividendole tra i membri del gruppo in base alle propensioni di ciascuno. Andrea Basso si è occupato di gestire l'applicazione per ufficio che comunica con la macchina PLC e della realizzazione del bot telegram, inoltre ha collaborato allo sviluppo del codice PLC con Samuel. Samuel Faccioli si è occupato della realizzazione del servizio Windows che comunica con il Server Web per l'invio dei dati. Pietro Beschin si è occupato della gestione del Server Web e della parte Back-End, creando le varie API e gestendo le richieste. Cristian Volpato si è occupato della parte Front-End, creando una dashboard di visualizzazione. Ha inoltre contribuito nello sviluppo dell'aspetto grafico dell'intero progetto.
In totale sono state impiegate 44 ore, suddivise in 2 settimane, per completare la realizzazione del progetto. Il tempo rimanente prima dell'esposizione è stato sfruttato per aggiornare il progetto con delle migliorie, curare l'aspetto grafico e occuparsi della documentazione.
Quotidianamente ci siamo aggiornati sul lavoro svolto, dedicando inizialmente una decina di minuti al brainstorming e in seguito alla definizione della timeline da seguire.
Per agevolare la collaborazione, condivisione del codice e il controllo sulle versioni è stato utilizzato github. L'intero progetto è disponibile a questo link.
Inoltre, come spazio di condivisione generale, è stato utilizzato Google Drive.

APPLICAZIONE UFFICIO

È stato sviluppato un applicativo C# con framework .NET per la gestione delle commesse e della loro esecuzione dall'ufficio di "Pianificazione della produzione".
La soluzione si basa sulle seguenti librerie:

- CommesseDll, contiene la gestione di una lista di commesse (vedi class diagram);
- DBManagerDll, utilizzata per l'interazione con il database SQL;
- PLCConnectionDll, per la connessione al PLC per mezzo della libreria 7Sharp;
- AlertDll, utilizzata per mostrare degli avvisi in seguito a determinati eventi, posizionati in basso a destra dello schermo.

Per personalizzare la parte grafica è stato utilizzato il framework Bunifu. In particolar modo questa scelta ha permesso di strutturare l'applicazione su un'unica form principale, la quale contiene un pannello laterale per la navigazione e un componente "pages" per la gestione delle sessioni. Quest'ultime sono:

- gestione commesse, nella quale è possibile amministrare il PLC e le commesse. Da qui si effettua la connessione alla macchina, l'abilitazione al lavoro e l'avvio del ciclo di esecuzione, si visualizzano lo stato, la velocità di produzione, il numero di pezzi prodotti e scartati, la progressione percentuale. È possibile creare e gestire più commesse, sfruttando una coda di esecuzione per cui le stesse vengono inviate automaticamente al PLC al termine delle precedenti. La visualizzazione è in forma tabellare e mantiene sempre l'ordine della coda.
- anagrafica, dove si può visualizzare uno storico di tutte le commesse eseguite.
- gestione clienti e prodotti, nella quale è possibile aggiungere ed eliminare clienti e prodotti.
- allarmi e simulazione guasti, in cui troviamo una parte di visualizzazione degli allarmi in atto e una di simulazione dei guasti e del rallentamento della macchina.

La scrittura nei database del PLC (descritti nella sezione apposita) avviene ogni volta che una commessa passa allo stato di esecuzione, mentre la lettura è cadenzata da un timer con un intervallo di 1 secondo.
È stato inoltre definito un timer per scrivere sulla variabile del PLC utilizzata per il controllo sulla connessione tra applicativo e PLC (definito "watchdog" e spiegato nell'apposita sezione).
I dati delle commesse sono inserite in un database locale (vedi diagramma E/R), il cui utilizzo è già stato precedentemente spiegato.

SERVIZIO C# PER L'INVIO DEI DATI AL CLOUD

Per rendere fruibili i dati del lavoro del PLC alla dashboard è stato realizzato un servizio Windows in C# che può essere installato in una macchina dedicata connessa nella medesima rete del PLC.
Ogni secondo vengono lette le informazioni disponibili e trasmesse al server Web per mezzo delle seguenti API in POST:

- http://54.85.250.76:3000/api/history per il popolamento o aggiornamento dello storico delle commesse,
- http://54.85.250.76:3000/api/status per l'aggiornamento dello stato macchina.

Una volta verificata la presenza di connessione con il PLC, il servizio salva i valori letti su due oggetti, basati su due classi distinte e definite:

- JsonHistory: classe contenente tutti i dati dello storico della commessa.
- JsonStatus: classe contenente tutti i dati dello stato macchina.

Una volta memorizzati i valori viene effettuata una serializzazione degli oggetti in due stringhe in formato JSON. Successivamente queste stringhe vengono inviate tramite le chiamate al server già descritte. Una feature particolarmente interessante di tale servizio è l'implementazione di una coda dei dati inviati al server web, relativi allo stato delle commesse eseguite. In questo modo si garantisce che gli stessi non vengano persi in caso di assenza momentanea di connessione al server, ma che ci sia una ritrasmissione una volta che la stessa venga ristabilita.

PLC

È stato programmato un dispositivo PLC per il controllo della macchina per la preparazione dei flaconcini. In particolar modo sono state gestite: la modalità automatica e manuale, gli allarmi, i guasti, la comunicazione con l'applicazione uffici e l'utilizzo di un dispositivo touch screen.
Il PLC viene abilitato al lavoro per mezzo di una variabile denominata "control word", che gestisce anche l'avvio e il blocco del ciclo automatico, impostata dall'applicazione ufficio.
Tramite l'inserimento dei dati attinenti alla stessa in un database interno definito "DB_PC-PLC", il PLC riceve la commessa da eseguire in modalità automatica.
Quest'ultima inizia con il prelievo da un nastro di carico di un flaconcino grezzo (la presenza del prodotto sul nastro viene simulata attraverso l'input del pulsante 2). Il manipolatore si sposta poi davanti alla macchina che effettua la stampa a tampone, attende che questa sia pronta, inserisce il prodotto e attende il completamento della lavorazione (la disponibilità del tampone e il termine della stampa sono simulate per mezzo del SELETTORE 1). Terminata questa operazione, il prodotto finito viene sottoposto al controllo di una telecamera che ne dichiara la bontà o rivela eventuali imperfezioni (tale operazione viene simulata da un timer e lo scarto avviene sempre ogni 10 iterazioni del ciclo a scopo dimostrativo). Conseguentemente a queste possibilità, il flacone può essere scartato, oppure scaricato sul nastro apposito. Lo scarto avviene anche in caso di nastro pieno (l'occupazione del nastro viene simulata attraverso l'input del pulsante 3 entro 3 secondi dalla fine del controllo della videocamera).
Al termine di ogni ciclo di lavorazione vengono aggiornate le variabili nel database "DB_PLC-PC".

In modalità manuale il PLC può essere utilizzato mediante comandi posti sul dispositivo touchscreen, salvo sia stato disabilitato dall'ufficio per mezzo della "control word" citata precedentemente. Ogni coppia di comandi, formata da operazioni opposte tra loro (destra/sinistra, su/giù, avanti/indietro e apri/chiudi), è associata a un bit che accende o spegne l'elettrovalvola relativa al movimento che si vuole effettuare.
Sono stati implementati i controlli sui guasti relativi ai finecorsa e alle elettrovalvole. Inoltre sono presenti una serie di guasti simulati dall'applicazione dell'ufficio a scopo dimostrativo, i quali vengono scritti nel database "DB_PC-PLC" con un codice identificativo.
In entrambi i casi, o nell'eventualità in cui venga premuto il pulsante di emergenza, il lavoro della macchina viene bloccato. L'emergenza viene scritta sul database "DB_PLC-PC" attraverso la

codifica in codice binario, in modo tale che sia visualizzabile sia dal software di gestione, sia nella pagina degli allarmi del touchscreen.
Per il controllo sulla corretta connessione tra il PLC e l'applicativo di gestione viene utilizzato un bit che viene posto a 1 dal PC e a 0 dal PLC ogni secondo. Se una delle parti riconosce che il bit non è stato modificato dall'altra, viene notificato all'operatore che la connessione è fallita.

CLOUD

Sono state create tre istanze utilizzando il servizio EC2 disponibile su AWS, ognuna facente parte della stessa subnet e VPC. Un'istanza è adibita al servizio web server frontend con APACHE e backend con NODEJS, una per MONGODB e una per il bot telegram. Il primo server è impiegato per ricevere dati tramite API e inviare le rispettive risposte in formato JSON. A sua volta questo server si interfaccerà con il secondo per inviare e ricevere dati verso e dal database MONGODB.
Le due istanze fanno parte dello stesso dominio di broadcast. Il server contenente NODEJS possiede un indirizzo IP elastico, tramite il quale l'istanza potrà essere raggiungibile dalla rete pubblica sempre con lo stesso indirizzo IPv4.
Il server contenente MONGODB invece non presenta nessun tipo di indirizzo pubblico, quindi sarà raggiungibile esclusivamente dalle istanze facenti parte del proprio dominio di broadcast.
Si è scelto questo metodo per diminuire il rischio di attacchi esterni direttamente rivolti al server MONGODB.
La terza, come la prima, possiede un indirizzo IP elastico, poichè deve comunicare con l'esterno.
Per monitorare il tutto è stata creata una dashboard usando il servizio AWS Cloudwatch, la quale controlla metriche della cpu, traffico rete e scrittura/lettura disco per ogni istanza.
Sarebbero inoltre utili: l'implementazione del protocollo HTTPS, per rendere lo scambio di dati più sicuro; una coda SQS, la quale agirebbe da buffer per i dati passati in POST da scrivere finalmente nel database. Così facendo si diminuisce il rischio di perdita di dati in caso di arresto anomalo del server NODEJS.

BACK END

NODEJS installato sulla prima istanza si interfaccia con il client utilizzando la porta 3000. Al contempo, per la comunicazione con il database MONGODB viene utilizzata la porta 27017.
A seconda dell'indirizzo inserito, verranno elaborati e restituiti dati differenti, grazie alla presenza del router.
In seguito sono presenti altri componenti quali i vari controller e model, che contengono i metodi. Uno tra questi (metodo list) permette di restituire le commesse filtrate (per nome dell'articolo e per intervallo di date di prima esecuzione) in base alle variabili passate in GET.
I controller elaborano le richieste ricevute e le passano al model. E' presente inoltre un MONGOOSE schema, realizzato grazie all'omonimo middleware, il quale permette di interfacciarsi con il server MONGODB. Ad esso verranno richieste dai model le azioni da eseguire. I risultati finali verranno restituiti ai controller, inviando a loro volta verso l'esterno dati in formato JSON.
Riguardo all'inserimento delle commesse, esse vengono aggiornate se già esistenti o altrimenti create. Se viene inviata una commessa con `codice_commessa` vuoto oppure una nuova commessa viene creata, l'ultima commessa viene considerata come terminata, contrassegnandola come fallita o completata.

All'interno del database MONGODB (vedi schema) sono presenti due collezioni: history e status.
Rispettivamente contengono lo storico delle commesse e lo stato della macchina.
Quest'ultima consiste in un unico dato che sarà aggiornato continuamente.
Si è voluto dedicare un'intera collection ad esso nel caso in futuro si volesse implementare la cronologia dello stato.
Lato gestione errori, sono stati impostati degli stati HTTP e messaggi contenuti in JSON da restituire al client in base al successo o al tipo di un eventuale fallimento.
Se, eseguendo un'operazione sul database MONGODB, non vengono rispettati i criteri di uno dei due database schema, il risultato sarà un validation error, evitando così di scrivere i dati in un formato non corretto.
Sarebbe stato utile implementare, nel caso fosse assente la connessione tra NODEJS e MONGODB, una coda contenente le commesse ancora non scritte nel database, mantenendole nella memoria dell'istanza web-server, fino a che il database rimane irraggiungibile. Quando sarebbe stato disponibile, i dati "in attesa" sarebbero stati inviati e salvati nella base dati esterna.

FRONT END

Per quanto riguarda la dashboard di visualizzazione dei dati si è scelto di suddividere l'interfaccia grafica in tre sezioni.
La sezione "stato macchina" mostra in tempo reale i dati sulla produzione in corso relativi alla commessa in lavorazione, stato della macchina, progresso della lavorazione ed eventuali emergenze.
La sezione "storico commesse" elenca le commesse associate alla macchina, sia passate che in corso. Tramite l'apposito form di ricerca è possibile filtrare l'elenco per articolo prodotto e intervallo di date con l'opzione di visualizzare solo le commesse fallite - questa opzione è disponibile sia per l'elenco completo che per l'elenco filtrato. È stato aggiunto un pulsante per eliminare i filtri di ricerca per permettere all'utente di tornare velocemente all'elenco completo. L'elenco delle commesse è strutturato in vari accordion, che visualizzano il nome della commessa e la data di ultimo aggiornamento, ma se si clicca sopra a uno degli elementi compare una sezione con dettagli ulteriori relativi alla commessa. A fianco di ogni commessa è presente un indicatore per mostrare rapidamente lo stato della commessa - pallino verde se completata, spinner giallo se in esecuzione e pallino rosso se fallita. Per facilitare la navigazione tra le varie commesse all'utente, l'elenco è suddiviso in varie pagine. Come i dati relativi allo stato della macchina, anche l'elenco delle commesse e i dettagli relativi sono aggiornati in tempo reale.
Nella sezione "statistiche di produzione" è stato aggiunto un grafico che mostra la quantità totale di pezzi prodotti e pezzi scartati dalla macchina diviso per giorno, affinché si possa visualizzare la produzione della macchina nel tempo.
Per la realizzazione della dashboard si è scelto di usare HTML per la parte di struttura e jQuery per la parte di logica. Lo stile di layout scelto per la pagina è ispirato al tema dark, ed è stato costruito "from the ground up" facendo ampio uso della libreria CSS Bootstrap, che è tornata utile per la gestione della parte responsive della dashboard. La paginazione dell'elenco commesse è stata gestita tramite la libreria Datatable.js e il grafico è stato realizzato con la libreria Apexcharts.js.
Per quanto riguarda la parte di logica, al fine di evitare di sovraccaricare il server di chiamate si è deciso di impiegare diverse chiamate GET a diverse API specializzate per diversi scopi. Al primo

caricamento della pagina viene fatta una singola chiamata a un'API che restituisce tutti i dati utili per popolare le varie sezioni della dashboard, successivamente per l'aggiornamento in tempo reale dei dati vengono chiamate altre due API, una per i dati dello stato macchina, chiamata con un timer che aggiorna i campi di testo ogni secondo, e una per l'elenco delle commesse, a sua volta chiamata con un timer ogni due secondi. Per la ricerca delle commesse viene chiamata la stessa API utilizzata per l'aggiornamento delle commesse, che passa al server i filtri inseriti dall'utente e restituisce un array di oggetti di tipo "commessa" che viene poi utilizzato per ripopolare l'elenco.

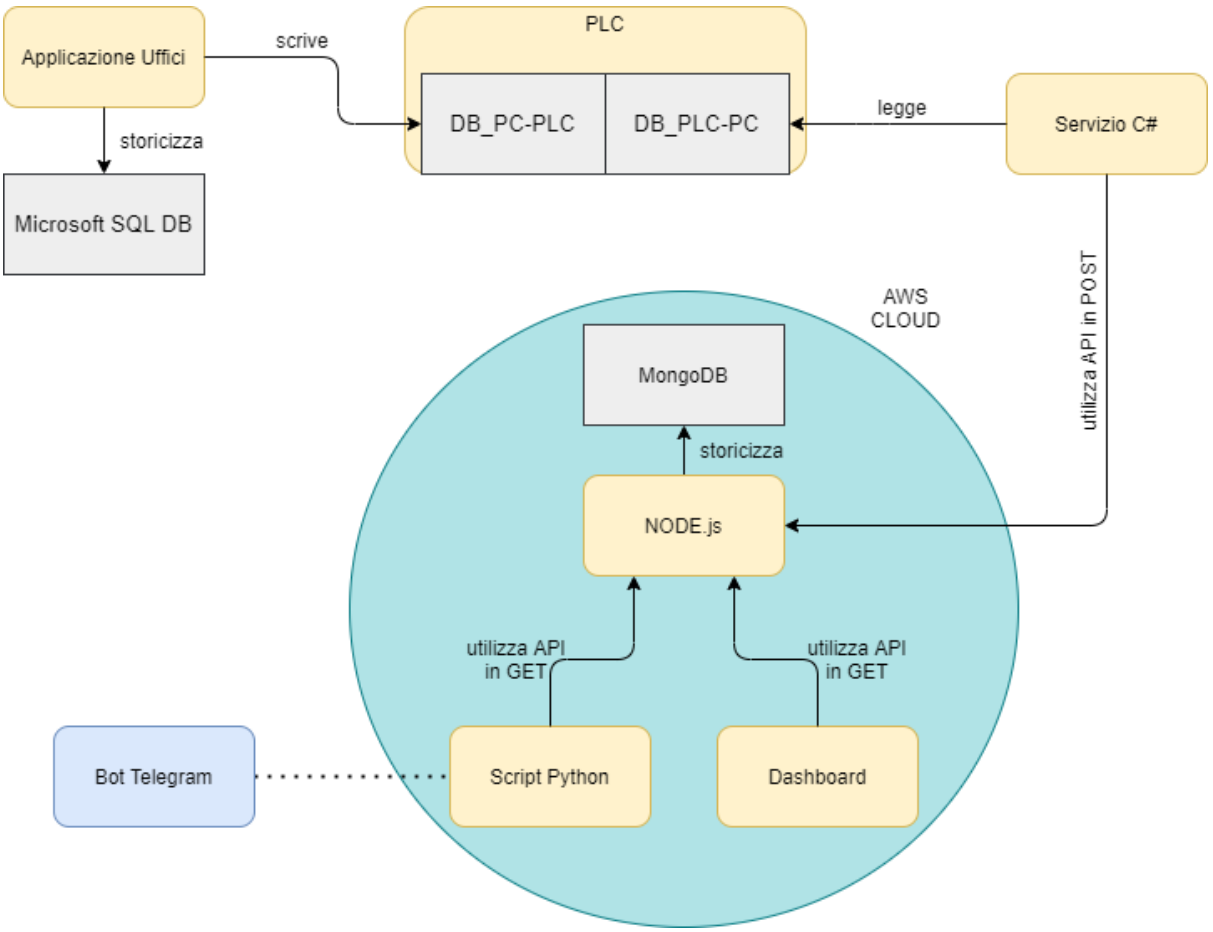
La scelta di chiamare la stessa API sia per la ricerca che per l'aggiornamento permette all'utente di continuare a visualizzare i risultati filtrati, nel caso abbia effettuato una ricerca, senza che il timer di aggiornamento della lista commesse ripopoli l'elenco completo.
La visualizzazione dell'elenco con le sole commesse fallite è stato effettuata con un controllo a lato client.

BOT TELEGRAM
Per una visualizzazione dei dati immediata è stato creato anche un bot telegram, sviluppato in Python attraverso l'utilizzo delle librerie "python-telegram-bot" per l'interazione con il bot e "requests" per le richieste alle API (vedi codice). Era previsto che tale codice fosse in esecuzione su AWS, sfruttando il servizio Lambda e un EventBridge, tuttavia per limitazioni dell'account fornito dall'istituto si trova su un'istanza EC2 dedicata.
I comandi al momento disponibili sono:

- info, ottiene lo stato dell'ultima commessa, la percentuale di avanzamento e il numero di pezzi prodotti all'ora;
- commessa, visualizza tutti i dati dell'ultima commessa eseguita;
- macchina, visualizza lo stato della macchina, la velocità e gli allarmi;
- storico, visualizza lo storico delle commesse.

APPENDICE

Schema generale



Class Diagram ***CommesseDll***

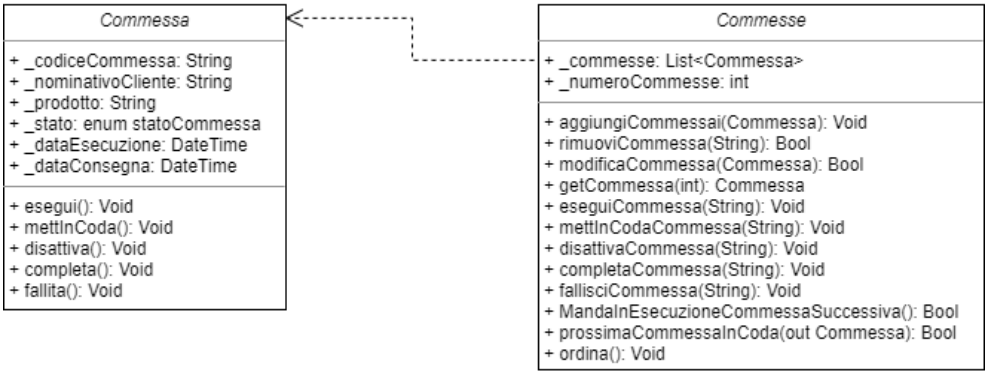
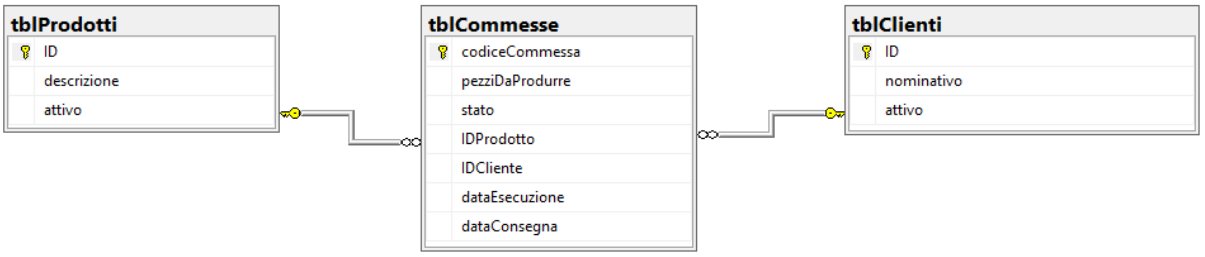


Diagramma E/R DB locale



Scrittura/lettura sul PLC da C#

```
//scrittura pezzi da produrre
CSerialDeserial.ReadFile(ref this.db1);
[...]
S7.SetDIntAt(this.db1Buffer, 104, pezziDaProdurre);
this.result = this.Client.DBWrite(1, 0, this.db1Buffer.Length, this.db1Buffer);

//lettura pezzi prodotti
this.result = this.Client.DBRead(2, 0, this.db2Buffer.Length, this.db2Buffer);
[...]
int pezziProdotti = S7.GetDIntAt(this.db2Buffer, 104);
CSerialDeserial.WriteFile(this.db2);
```

Invio dati dal servizio a Node.js

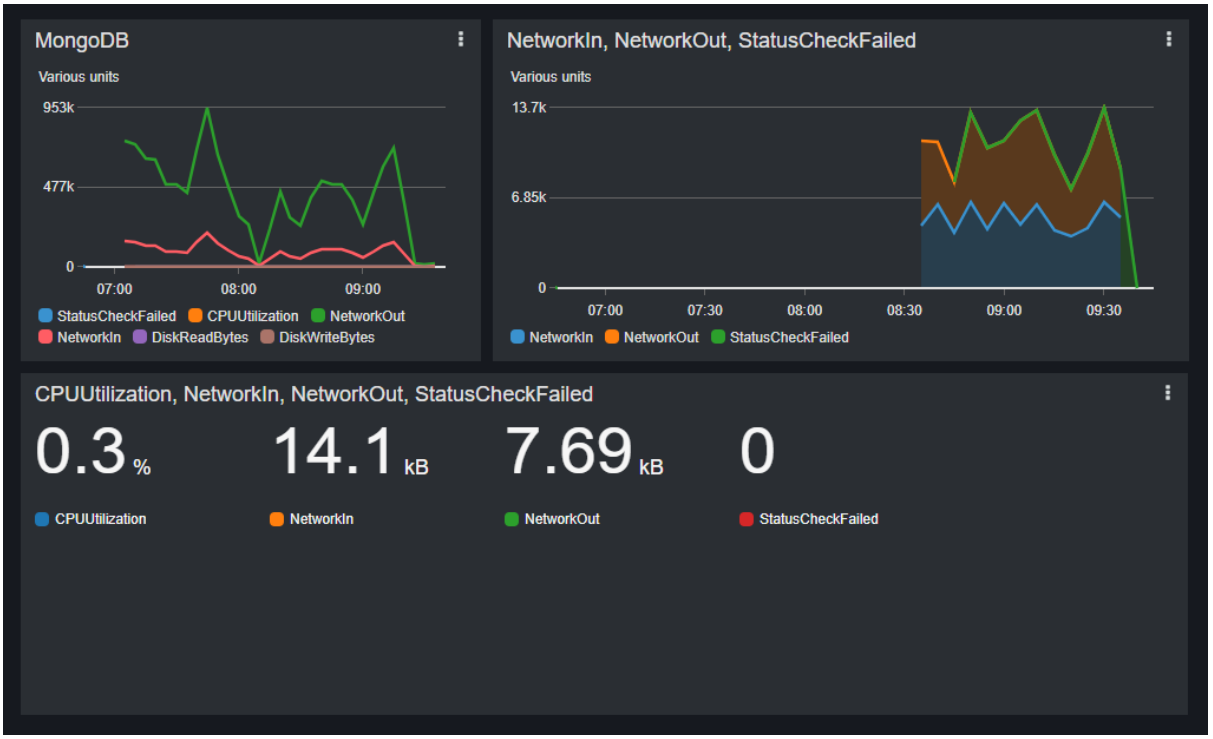
```
HttpWebRequest requestStatus = (HttpWebRequest)WebRequest.Create(postStatusURL);
requestStatus.Timeout = 500;
requestStatus.Method = "POST";
requestStatus.ContentType = "application/json";
StreamWriter swStatus = new StreamWriter(requestStatus.GetRequestStream());
swStatus.Write(js);

swStatus.Flush();
swStatus.Close();

HttpWebResponse responseStatus = (HttpWebResponse)requestStatus.GetResponse();
responseStatus.Close();
```

DB del PLC

codice commessa	String[50]	0.0
prodotto	String[50]	52.0
pezzi da produrre	DInt	104.0
CONTROL WORD	USInt	108.0
avviso per operatore d..	String[100]	110.0
Offset velocità macch...	DInt	212.0
Watch Dog	USInt	216.0
possibili guasti	USInt	217.0
data consegna	String[30]	218.0
data esecuzione	String[30]	250.0

codice commessa	String[50]	0.0
prodotto	String[50]	52.0
pezzi prodotti	DInt	104.0
pezzi scartati per scari..	DInt	108.0
pezzi scartati diffetosi	DInt	112.0
stato macchina	USInt	116.0
velocita macchina	DInt	118.0
H	DWord	122.0
I	DWord	126.0
avviso per ufficio da o...	String[100]	130.0
codice allarme	USInt	232.0

CloudWatch



Definizione API

```
router.get('/history', historyController.list);
router.get('/lastCommessa', historyController.getLastCommessa);
router.get('/status', statusController.getStatus);
router.post('/history', historyController.store);
router.post('/status', statusController.store);
router.get('/historyStatus', mainController.getHistoryStatus);
router.get('/lastCommessaStatus', mainController.getLastCommessaStatus);
```

Mongoose Schema

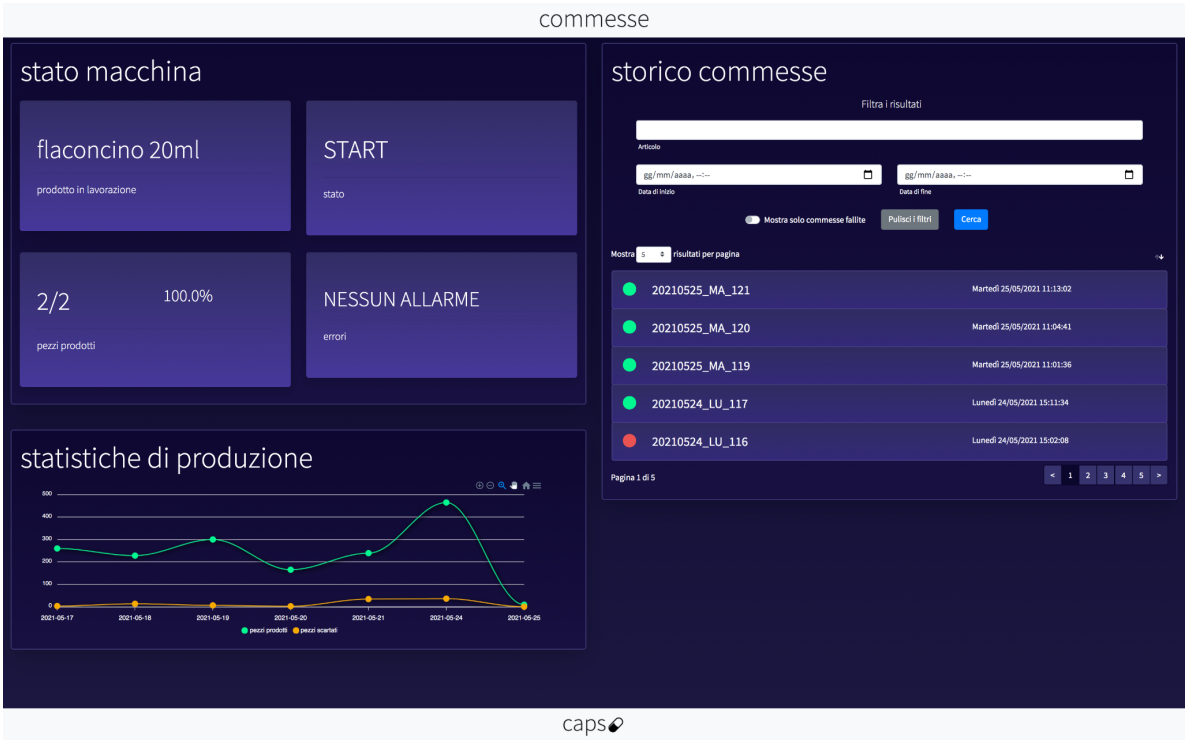
```
let historySchema = mongoose.Schema({
    id : false,
    versionKey : false,
    codice_commessa : String,
    articolo : String,
    quantita_prevista : Number,
    data_consegna : Date,
    data_aggiornamento : Date,
    data_esecuzione : Date,
    quantita_prodotta : Number,
    quantita_scarto_difettoso : Number,
    quantita_scarto_pieno : Number,
    stato: {
        type : String,
        enum : ['completata', 'fallita', 'in esecuzione']
    }
},
{
    collection : 'history'
});
```

```
let statusSchema = mongoose.Schema({
    id : false,
    versionKey : false,
    stato : {
        type : String,
        required : true
    },
    allarme : String,
    velocita : Number
},
{
    collection : 'status'
});
```

Chiamata API al caricamento della dashboard

```
// chiamata API effettuata al caricamento della pagina
const fetchAllData = () => {
    $.ajax({
        type: 'GET',
        url: `${baseURL}historyStatus`,
    }).then(result => {
        datiCommesse = result.history;

        // popolo lo storico commesse
        $('#accordion-commesse').empty();
        datiCommesse.forEach(commessa => {
            if ($('#switch-completato').is(":checked") === true) {
                if (commessa.stato === 'fallita') {
                    addCommessaToList(commessa)
                }
            } else {
                addCommessaToList(commessa)
            }
            updateTableRow(commessa);
        });
```

Schermata della dashboard



Script in python

```python
def view_info(update, context):
    r1 = requests.get('http://54.85.250.76:3000/api/status')
    r1json = r1.json()
    [...]
    rtext = 'stato: '+stato+'\n'+'avanzamento: '+percentualeproduzione+'%'+'\n'+'pezzi/ora: '+velocita
    update.message.reply_text(rtext)

def main():
    upd = Updater(TOKEN, use_context=True)
    disp = upd.dispatcher
    disp.add_handler(CommandHandler("info", view_info))
    upd.start_polling()
    upd.idle()
```