|  |  |
| --- | --- |
| **PROJECT WORK**  CdS Informatica per le aziende digitali (L-31)  ***Settori Scientifici Disciplinari*** Informatica (INF/01), Sistemi di elaborazione delle informazioni  (ING-INF/05), Impianti meccanici industriali(ING-IND/17) | |
| **Cognome e Nome:** | **Andrea D’Orazio** |
| **Numero di Matricola**: | **0312300107** |
| **Corso di Studio:** | Barrare la casella riferita al proprio corso di studio |
| ◊   L-5 Filosofia ed Etica |
| ◊   L-22 Scienze Motorie |
| * L-31 Informatica per le Aziende Digitali |
|  |
| **Tema n:** | 1 |
| **Titolo del tema:** | **La digitalizzazione dell’impresa** |
| **Traccia del PW n:** | **1.1** |
| **Titolo della traccia:** | **Lotto Economico di Ordinazione (EOQ) per materiali a domanda**  **indipendente** |
| **Titolo dell’elaborato:** | Attribuire un titolo al proprio elaborato progettuale |
| **PARTE PRIMA – DESCRIZIONE DEL PROCESSO** | |
| **Utilizzo delle conoscenze e abilità derivate dal percorso di studio:** | Per la realizzazione del progetto, le competenze e i concetti acquisiti nei corsi di Programmazione e Tecnologie Web, insieme al corso di Programmazione Distribuita, sono stati di fondamentale importanza. Questi corsi hanno fornito le competenze necessarie per la progettazione e l'implementazione del progetto. L'applicativo web sviluppato offre agli utenti uno strumento intuitivo per il calcolo del Lotto Economico di Ordinazione (EOQ), la sua realizzazione è stata possibile con l’utilizzo delle tecniche di programmazione e delle librerie più diffuse, apprese durante il corso di Tecnologie Web.  Per la parte teorica, sono stati utili gli argomenti di pianificazione e gestione delle scorte affrontati nel corso di 'Organizzazione aziendale e marketing', tuttavia, poiché l'argomento del calcolo del Lotto Economico di Ordinazione non era esplicitamente trattato, sono stati necessari ulteriori approfondimenti per acquisire le competenze necessarie [1][3][4]. Le nozioni apprese nel corso di Statistica sono state utilizzate invece per la realizzazione di un semplice esempio di previsione di serie storiche, integrato nell’applicativo. |
| **Fasi di lavoro e relativi tempi di implementazione per la predisposizione dell’elaborato:** | La realizzazione del project work è stata possibile pianificando una serie di fasi con relative tempistiche più o meno rispettate durante lo svolgimento dell’elaborato.  Di seguito le fasi pianificate dopo la scelta del tema e relativa traccia :   * analisi preliminare della scheda del project work; * approfondimento e studio del tema inerente al modello economico di acquisto (EOQ - Economic Order Quantity) con relativa selezione delle fonti; * approfondimento e utilizzo delle nozioni apprese dagli insegnamenti del Corso di Laurea necessari alla realizzazione del project work ; * selezione e scelta del tipo di architettura per la realizzazione dell’applicativo; * progettazione dell’algoritmo di calcolo (input, output e passi di elaborazione); * realizzazione dei diagrammi di caso d’uso per la realizzazione del software; * scelta degli ambienti di sviluppo e dei linguaggi di programmazione più idonei allo sviluppo dell’applicativo; * implementazione dell’algoritmo di calcolo e della logica di business; * realizzazione dell’interfaccia utente per il calcolo del Lotto Economico di Ordinazione.   Sintetizzando ho individuato quindi tre macrofasi con i rispettivi tempi di esecuzione:   * Fase 1, analisi, studio e approfondimento, selezione del materiale e delle fonti – circa tre settimane. * Fase 2, scelta dell’architettura software, dei framework e dei linguaggi di programmazione, progettazione dell’algoritmo di elaborazione e del software – due settimane. Un aspetto significativo di questa fase è stato il corretto dimensionamento del progetto, bilanciando la risposta alla traccia del project work con l'implementazione di aspetti aggiuntivi.   Questa fase si è parzialmente sovrapposta con la seguente (fase 3) in quanto è stato necessario valutare la fattibilità di alcune scelte operative, come ad esempio l’aggiunta di strumenti previsionali.   * Fase 3 realizzazione dell’applicativo (algoritmo, logica di business e interfaccia utente) – tre settimane. In questa fase è stato sviluppato tutto il codice per la realizzazione del software |
|  |
|  |
|  |
|  |
| **Risorse e strumenti impiegati:** | **Risorse utilizzate per l’approfondimento teorico.**  L’individuazione delle risorse necessarie alla realizzazione del project work è iniziata dalla ricerca del primo articolo riguardante il calcolo del Lotto Economico di Acquisto [1] e una sua analisi postuma [2], in seguito sono stati selezionati testi utilizzati come riferimento per l’analisi e la gestione delle scorte [3], [4].  Per contestualizzare il tema da un punto di vista attuale sono stati inoltre utilizzati due articoli [5][7] per sottolineare i limiti del modello classico del Lotto Economico di Ordinazione.  **Risorse utilizzate per la realizzazione del software**  Il software richiesto dalla traccia di questo project work è stato realizzato  utilizzando un’architettura **client-server** di tipo *web application* che ha previsto l’utilizzo del linguaggio HyperText Markup Language (HTML)[8] e il linguaggio JavaScript [9] per l’implementazione dell’interfaccia utente. Per un layout funzionale è stato selezionato il framework Bootstrap [10] che prevede sia funzioni JavaScript che fogli di stile (*Cascade Style Sheet – CSS*) [11]. Per la parte server è stato scelta la piattaforma *open-source* *NodeJS* [12] che consente l’esecuzione a *runtime* di codice JavaScript.  La disponibilità, inoltre, di numerosi moduli – *Node Package Manager – NPM* da possibilità di creare agevolmente il proprio web-server per la realizzazione di applicazioni facilmente scalabili e performanti. Il modulo Express[13] è stato concepito proprio per l’implementazione di applicazioni web minimali e flessibili, facilmente integrabili con eventuali sistemi aziendali già in uso.  Il web server utilizza uno stile architetturale di tipo *REST (Representational state tranfer)*[14]molto utilizzato in sistemi distribuiti, che sfrutta il protocollo HTTP (HyperText Transfer Protocol) per lo scambio di risorse, ad esempio dati strutturati nel formato standard de-facto JSON [15].  Per estendere le funzionalità dell’applicazione è stato utilizzato anche il linguaggio *Python*[16] per implementare l’algoritmo di calcolo, linguaggio versatile e ad oggi il più utilizzato [17]. La scelta di utilizzare Python è fondata su due motivi, il primo è che è dotato di una molteplicità di *packages* che ne estendono i campi di applicazione, il secondo è che può essere richiamato direttamente da NodeJS grazie al modulo *child\_process* che permette di istanziare sottoprocessi e l’interazione diretta con i risultati delle elaborazioni.  Tra i vari packages ho individuato StatsForecast[18] realizzato da Rob J Hyndman[19] che fornisce diversi modelli di previsione inclusi alcuni modelli automatici come AutoARIMA[20], utilizzato in questo progetto come esempio di una possibile previsione per una delle componenti di calcolo , la domanda.  **Altri strumenti utilizzati**  Sono stati necessari durante la stesura del project work, di ulteriori strumenti come draw.io[21] per la realizzazione di grafici e diagrammi, il portale W3School[22], come risorsa per le tecnologie web implementate nel lavoro e GitHub[23] come repository per la gestione e diffusione del codice sviluppato. Come strumento editor per la scrittura del codice ho utilizzato il prodotto Microsoft, Visual Studio Code[24].  La ricerca di strumenti e risorse ha impiegato principalmente la fase iniziale dell’elaborato, ed è stata condotta sia con ricerche online, sia nella consultazione di testi. Il reperimento delle informazioni non è stato complicato ma alcune difficoltà sono emerse nell’armonizzare le varie fonti per produrre una sintesi esaustiva per la sezione teorica del progetto, legata al calcolo dell’EOQ.  Per la parte di sviluppo dell’applicativo web invece il materiale individuato è stato esclusivamente online, ogni framework o linguaggio utilizzato infatti è sempre dotato di una ricca documentazione corredata di esempi esaustivi. Anche le librerie Python per la parte di calcolo statistico sono ampiamente documentate online nelle pagine web a supporto dei relativi progetti.  **Bibliografia:**  [1] Harris, F.W. ,1913, *How many parts to make at once*. Factory, The Magazine of Management, Volume 10, N° 2.  [2] Erlenkotter D. ,2014, *Ford Whitman Harris’s Economical Lot Size Model*. International Journal of Production Economics.  [3] G. Urgelletti Tinarelli ,1981, *La gestione delle scorte*, Etas Libri.  [4] Brandolesi A., Pozzetti A., Sianesi A. ,1991, *Gestione della produzione industriale*. Hoepli.  [5] *50 Years of EOQ: A Critical Review,* Journal of the Operational Research Society  [6] *Data Science e Machine Learning*, Michele di Nuzzo  **Sitografia:**  [7] *La crisi da Covid-19 - Dalla crisi sanitaria alla crisi economica* <https://www.consob.it/web/investor-education/crisi-sanitaria-economica>. |
| [8] WHATWG, “*HTML”, 2024.* [Online]. Disponibile: [https://HTML.spec.whatwg.org/multipage/](https://html.spec.whatwg.org/multipage/) [Accesso: 10-dic-2024].  [9] Mozilla Developer Network, *JavaScript, 2024.* [Online]. Disponibile: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Accesso: 10-dic-2024].  [10] Bootstrap, *Documentazione di Bootstrap*. [Online]. Disponibile: [https://getBootstrap.com](https://getbootstrap.com). [Accesso: 10-dic-2024]  [11] Mozilla Developer Network (MDN*), CSS: Cascading Style Sheets*. [Online]. Disponibile: <https://developer.mozilla.org/it/docs/Web/CSS>. [Accesso: 10-dic-2024].  [12] Node.js Foundation, Node.js. [Online]. Disponibile: <https://nodejs.org>. [Accesso: 10-dic-2024].  [13] Express.js Foundation, *Express - Fast, unopinionated, minimalist web framework for Node.js*. [Online]. Disponibile: <https://expressjs.com>. [Accesso: 10-dic-2024].  [14] Restfulapi.net, *Guida alle API RESTful*. [Online]. Disponibile: <https://restfulapi.net>. [Accesso: 10-dic-2024].  [15] ECMA International, JSON. [Online].Disponibile: . <https://www.json.org/json-en.html> [Accesso: 10-dic-2024].  [16] Python Software Foundation, *Python*. [Online]. Disponibile: [https://www.Python.org](https://www.python.org). [Accesso: 10-dic-2024].  [17] Tiobe, Index for December 2024.[Online]. Disponibile:  <https://www.tiobe.com/tiobe-index/>. [Accesso: 10-dic-2024].  [18] Nixtla. *Statsforecast: A fast and scalable forecasting library*. [Online]. Disponibile: <https://github.com/Nixtla/statsforecast>. [Accesso: 10-dic-2024].  [19] Rob J Hyndman Disponibile: <https://robjhyndman.com/>  [20] AutoARIMA Model Disponibile: https://nixtlaverse.nixtla.io/statsforecast/docs/models/autoarima.html  **Strumenti**  [21] JGraph Ltd. *draw.io: Diagramming tool*. [Online]. Disponibile: https://app.diagrams.net. [Accesso: 10-dic-2024].  [22] W3Schools. *W3Schools Online Web Tutorials*. [Online]. Disponibile: <https://www.w3schools.com>. [Accesso: 10-dic-2024].  [23] GitHub, Inc. *GitHub*. [Online]. Disponibile: <https://github.com>. [Accesso: 10-dic-2024].  [24] Microsoft Visual Studio Code. Disponibile: <https://code.visualstudio.com/> [Accesso: 10-dic-2024] |
| **PARTE SECONDA – PREDISPOSIZIONE DELL’ELABORATO** | |
| **Obiettivi dell’elaborato/progetto/artefatto:** | L’obiettivo del progetto è quello di realizzare uno strumento software che consenta, nell’ambito delle aziende che si occupano di gestione dei materiali a domanda indipendente, di calcolare i quantitativi ottimali di acquisto minimizzando i costi totali legati alla gestione delle scorte in magazzino. L’elaborazione dei dati per il calcolo è stato eseguito utilizzando la formula originaria proposta da Ford W.Harris nel 1913 nell’articolo “*How many parts to make at once*” [1].  La realizzazione dell’elaborato è stata interpretata per proporre uno strumento, benché in forma prototipale, semplice e pratico per il calcolo del Lotto economico di ordinazione (EOQ), pensato per una reale integrazione nel contesto aziendale. La scelta di sviluppare una *web application* è stata motivata dall'esigenza di garantire semplicità, performance e scalabilità del sistema.  Per rendere l’utilizzo ancora più realistico è stata inserita la possibilità di lavorare quantità di dati maggiori, consentendo il caricamento di file strutturati, permettendo così elaborazioni in serie storica per l’analisi dell’andamento sia del valore dell’EOQ che delle componenti di costo.  L’implementazione di funzionalità di analisi predittiva della domanda, estendono le potenzialità dello strumento oltre il semplice calcolo del lotto economico.  L’output ottenuto vuole offrire un quadro, seppur sintetico, che illustri in modo chiaro l’andamento dei costi e degli ordini negli anni, avvalendosi anche del supporto di grafici. |
| **Contestualizzazione:** | L’obiettivo di efficienza è oggi più che mai fondamentale durante i processi di produzione e la corretta gestione del magazzino e delle scorte rappresentano un elemento di importanza vitale per la sopravvivenza di un’azienda. Il modello economico di ordinazione (EOQ - Economic Order Quantity) a domanda indipendente nasce come modello per la corretta gestione delle scorte, con l’obiettivo di definire la quantità ottimale di acquisto necessaria per minimizzare il costo totale di acquisto ottenuto tenendo in considerazione sia i costi di approvvigionamento sia i costi di mantenimento delle scorte in magazzino.  Ideato e proposto da Ford W.Harris nel 1913 nell’articolo “How many parts to make at once” [1], il modello è stato ampiamente utilizzato, discusso e rielaborato negli anni [2]. Fondamentali gli aggiornamenti di R.H. Wilson (1935) che lo hanno fatto divenire un valido strumento, ancora attuale, per il controllo e la gestione delle scorte.  La gestione delle scorte [3] può essere vista come la ricerca di una risposta bilanciata a due interrogativi:   * *quanto*? Il volume di prodotti da acquistare ad ogni ordine * *quando*? Con quale frequenza effettuare un ordine   La sfida è infatti nel rispondere correttamente a queste due domande, che costituiscono il punto sul quale fanno leva i costi delle scorte. Entrano in gioco molte variabili che influenzano drasticamente e in maniera significativa le scelte adottate su quantità e tempistica degli ordini di acquisto. Ad esempio, lotti di ordinazione di maggiori dimensioni diminuiscono i costi relativi all’emissione degli ordini, oltre l’opportunità di ottenere prezzi d’acquisto più vantaggiosi, d’altro canto però i costi di capitale saranno più alti e si aggiungeranno anche i costi di immagazzinamento, oltre a rischi legati ad obsolescenza e deperimento dei materiali. Il modello del lotto economico di acquisto semplifica notevolmente le variabili in gioco, infatti, nella sua versione base così come concepita da Harris, prevede delle ipotesi iniziali abbastanza stringenti [3]:   * La domanda ***D*** totale annua viene considerata nota e costante; * I costi di gestione ***S*** degli ordini sono costanti; * Il tasso del costo di mantenimento ***I*** del singolo bene è costante; * Il costo di acquisto ***C*** per unità è costante e indipendente dalla quantità acquistata; * I tempi di consegna del lotto sono nulli; * Il lotto viene consegnato per intero; * Il tempo che intercorre tra due ordini (tempo di riordino) è definito e costante.   Secondo il modello, il dimensionamento degli ordini sarà influenzato da due tipologie di costi che costituiscono i costi totali [4].  Costi di *approvvigionamento* che rappresentano tutti i costi sostenuti per l’emissione di un ordine, vale a dire il momento in cui si decide di riordinare. Rappresentano la somma del costo di acquisto, costi amministrativi di emissione ordine, di trasporto, movimentazione interna al magazzino, verifica e controllo dell’ordine, dei materiali ricevuti. I costi di ordinazione dipendono dal numero di ordini emessi nel periodo, come mostrato in Figura 1 aumentando la quantità di prodotti acquistati ***Q*** per ogni ordine, visto che il numero di prodotti necessari è costante nel periodo (corrisponde alla domanda ***D***), il numero di ordini diminuisce con conseguente diminuzione del costo di approvvigionamento.  Definiamo quindi il Costo totale di approvvigionamento nel seguente modo:  ***Ca = S \* (D/Q)***  Dove (D/Q) rappresenta il numero di ordini nel periodo.    Figura 1 - Grafico andamento costo unitario di approvvigionamento - Fonte [3]  L’altra componente riguarda i costi di *immagazzinamento*, relativi alle spese per attrezzature, spazio di stoccaggio, assicurazioni per i locali, spese del personale di magazzino, eventuali danni e obsolescenza dei prodotti oltre ai costi riferiti al capitale immobilizzato nel tempo.  All’aumentare dei costi di immagazzinamento verrà privilegiato l’acquisto di lotti più piccoli con conseguente riduzione della giacenza in magazzino.  In Figura 2 è visibile l’andamento dei costi proporzionale alla quantità di prodotti in magazzino.    Figura 2 - Grafico andamento costi di immagazzinamento – Fonte [3]  I costi totali di immagazzinamento come:  ***Ci= I \* C \* Q/2***  Harris, definisce ***I*** come il tasso di interesse da calcolare sul prezzo unitario del prodotto come deprezzamento del valore, abbiamo quindi  ***I\*C*** che equivale al *costo unitario di mantenimento in magazzino* mentre ***Q/2*** rappresenta la *giacenza media* presente in magazzino, ottenuta ipotizzando il consumo completo dei prodotti a magazzino prima di un nuovo ordine.  Il costo totale è ottenuto sommando le componenti:  È necessario quindi un *trade-off* tra le componenti dei costi evidenziati, per ottenere il minimo è sufficiente derivare rispetto alla quantità *Q:*  L’ultima equazione corrisponde alla formula di Harris per il calcolo del quantitativo da ordinare.  Il grafico in Figura 3 mostra l’andamento della funzione dei costi totali (al netto dei costi di acquisto) rispetto ai costi di immagazzinamento e approvvigionamento. L’incrocio tra la retta e la curva dei due costi indica sull’asse delle ascisse la dimensione del Lotto Economico di Ordinazione, che corrisponde al livello che minimizza i costi totali (derivata prima uguale a zero).    Figura 3 - Grafico andamento costi totali – Fonte [1]  In figura 4 è visibile l’andamento della giacenza in magazzino nel tempo. Individuiamo il *livello di riordino* che indica la quantità minima in giacenza per cui si effettua un nuovo ordine di materiale e la distanza tra due punti di riordino rappresenta il *tempo di approvvigionamento*. Nel caso di esaurimento completo dei prodotti il livello di riordino è uguale a zero. La quantità *Q* di prodotti per ogni nuovo ordine corrisponde al Lotto Economico di Ordinazione - EOQ.    Figura 4 - Grafico punto di riordino  Il modello fin qui presentato nella sua formulazione originaria è, seppur efficace, estremamente semplicistico e difficilmente aderente alla realtà. Dalla sua nascita sono stati formulati diversi modelli [5] che migliorano l’impianto iniziale, nei quali le altre variabili in gioco non sono considerate costanti. Esempi come l’EOQ con *sconti di quantità* dove il prezzo di acquisto del prodotto è funzione della quantità acquistata, il modello con *tempo di approvvigionamento variabile* nel quale il tempo di consegna non è costante, oppure il modello con *deterioramento,* dove i prodotti immagazzinati subiscono una perdita di valore e sono soggetti a un loro ciclo vita, rappresentano un’evoluzione del modello iniziale.  In questo lavoro oltre al calcolo base dell’EOQ viene proposta anche l’implementazione del modello con *domanda variabile*, condizione che, specie in questi ultimi anni, ha evidenziato come la domanda di un bene possa variare influenzata da condizioni sociali ed economiche, esogene a qualsiasi legge di mercato, come sperimentato durante l’ultima pandemia e le crisi internazionali ancora in atto.  L’utilizzo di tecniche di *forecasting* rappresenta un valido strumento per la previsione dell’andamento della domanda di beni richiesti *D*, così da poter ottimizzare la gestione delle scorte e l’emissione di nuovi ordini.  Le previsioni si basano sull’analisi delle serie temporali [6], che rappresentano sequenze di osservazioni in un periodo ovvero la sequenza di valori che una dimensione, nel nostro caso la domanda, assume nel tempo.  Tra i modelli più semplici per prevedere i valori di una serie temporale è possibile utilizzare i *modelli autoregressivi AR*. Questi modelli suppongono che il valore *y* di una serie al tempo *t* sia composto dalla somma di due componenti: una che dipende dai valori precedenti ottenuto applicando una *regressione lineare* e l’altra da un valore aggiuntivo detto *rumore bianco* (white noise)  il valore di *y* al tempo *t* è dato dalla somma di , che rappresenta la regressione lineare applicata al periodo precedente più che rappresenta il valore del rumore.  Il modello ARIMA (*Autoregressivo a media mobile integrato*) [6] costruito dalla combinazione di modelli autoregressivi AR e il *modello a media mobile* MA, è quello che ho selezionato per un esercizio di applicazione.  Il modello ARIMA è indicato per serie temporali stazionarie che presentano media costante e varianza costante, condizioni non aderenti alla domanda di un bene nel mondo reale, ma apportano una semplificazione accettabile se rapportate alle condizioni inziali del calcolo del Lotto Economico di Ordinazione, dove la domanda è considerata costante. |
| **Descrizione dei principali aspetti progettuali:** | **GitHub:**<https://github.com/adora0/eoq_f> **Demo:**[Calcolo EOQ](http://82.165.144.82:3000/) (http://82.165.144.82:3000/)  Per la realizzazione dell’applicativo si è ipotizzato di implementare il prototipo di un piccolo sistema, che risponda alle richieste del progetto di calcolo del Lotto Economico di Ordinazione (EOQ), una piattaforma modulare che offra scalabilità ed estensibilità, in termini di funzionalità, e che sia in grado di adattarsi alle esigenze in evoluzione di un’ipotetica azienda che necessita di dotarsi di un’applicazione in grado di interfacciarsi con eventuali sistemi preesistenti.  L’applicazione deve consentire le seguenti operazioni di base:   * Inserimento dati da parte dell’utente; * Calcolo del Lotto Economico di Ordinazione e dei costi totali per diversi anni; * Offrire un’interfaccia *user-friendly* e intuitiva per l’inserimento dei dati di input e la lettura dell’output restituito.   **Progettazione e realizzazione**  L’idea di base per la realizzazione del software è stata quella di implementare un’applicazione web.  I vantaggi tangibili di un’applicazione di questo tipo sono principalmente, la concorrenza, ovvero il server web è in grado di gestire agevolmente richieste multiple e concorrenti, l’indipendenza dei livelli che consente la scalabilità orizzontale (*scale out*) del sistema con la possibilità di aumentarne la dimensione aggiungendo più computer (*host*). Altro elemento a vantaggio di architetture di questo tipo è l’affidabilità, è infatti agevole isolare eventuali problemi e il ripristino dei componenti avviene senza eccessive interruzioni dell’intero sistema.  Le applicazioni web per natura sono fruibili via rete ed utilizzano linguaggi e protocolli di comunicazioni tipici della rete Internet, garantendo una facilità d’implementazione e gestione, grazie ai numerosi linguaggi e framework disponibili.  In quest’ottica il software realizzato per questo progetto, seppure semplice, vuole essere una base idealmente ampliabile con funzioni di calcolo via via più complesse con la possibilità di ospitare i livelli su più host, migliorando l’efficienza e permettendo una gestione ottimale delle risorse disponibili con possibilità di distribuzione geografica, offrendo quindi un ideale strumento per aziende di diverse dimensioni e strutture.  L'architettura di base che è stata scelta è basata sul framework Node.js[12] e l’integrazione di Python [16], ha permesso di realizzare un applicativo con struttura modulare che unita all'utilizzo di tecnologie web standard lo rendono facilmente integrabile con eventuali software preesistenti. L'interfaccia utente, sviluppata con HTML e librerie JavaScript, ne semplificano l’esperienza di utilizzo dell’utente, oltre a una facile implementazione di personalizzazioni con l’integrazione di funzionalità aggiuntive.  L’applicazione è composta da tre componenti principali:   * *livello di presentazione* (Presentation Layer), relativo al client, quindi alla parte front-end, corrisponde al browser utilizzato dall’utente per collegarsi al sistema e include tutte le componenti responsabili dell’interfaccia utente e le interazioni con l’utente finale; * *livello logica di business* (Business Logic), corrisponde all’applicazione in back-end (server web) che ha il compito di gestire le richieste del client, veicolare gli input per restituire gli output; * *livello di elaborazione dati* (Data Processing Layer), corrisponde al modulo specifico che si occupa dell’elaborazione dei dati, potenzialmente con tecniche evolute, scalabili e indipendenti dal layer che le richiama.   Il dialogo tra il client e server (livello di presentazione e il livello di logica di business) avviene attraverso l’utilizzo del protocollo http e lo scambio di messaggi in formato JSON [15] implementando un’architettura RESTful [14]. Il server web espone infatti diversi endpoint sia per la gestione dell’interfaccia grafica, sia per l’esecuzione delle elaborazioni.    Figura 5 - Diagramma dei componenti  Il **livello di presentazione** è stato implementato realizzando pagine con il linguaggio HTML, l’utilizzo di fogli di stile CSS e codice JavaScript per gestirne la parte dinamica e il dialogo con il server. La tipologia è di tipo *Single Page Application* (SPA), quindi una pagina singola che cambia contenuto dinamicamente in risposta alle interazioni dell’utente.  Per realizzare l’interfaccia con un aspetto pratico e funzionale è stata utilizzata la libreria Bootstrap [10], composta sia da codice CSS che JavaScript, che offre numerosi strumenti, stili e componenti. In particolare, ho utilizzato il *Grid system* che realizza layout di tipo *responsive* che si adattano automaticamente alla dimensione dello schermo (desktop, laptop o smartphone) che sta utilizzando l’utente.  L’interfaccia web è strutturata nel seguente modo:   * file HTML: * index.html, pagina principale che contiene l’intera applicazione; * params.html, pagina per l’inserimento dei parametri di input e visualizzazione dell’output; * info.html, pagina con le informazioni sull’applicazione e il progetto; * JavaScript: * Bootstrap.bundle.min.js, libreria Bootstrap; * chart.js, libreria del framework ChartJS per I grafici di output; * script.js, libreria personale con funzioni JavaScript per la gestione dell’applicazione; * route.js, libreria personale con funzioni per il dialogo con il server web. * CSS: * Bootstrap.min.css, foglio di stile Bootstrap; * stile.css, foglio di stile personale.   La pagina principale (file index.html) è composta da sezioni definite con tag <div> con lo stile *container-fluid* di Bootstrap che definisce un contenitore responsive che si adatta alle dimensioni dello schermo del dispositivo che sta utilizzando l’utente. Il container racchiude il navigatore, il contenuto principale dinamico e una sezione a fondo pagina(footer).  Codice HTML del contenitore principale:    Tag div con id *main* checontiene la sezione dell’applicazione aggiornata dinamicamente:    Sezione footer della pagina:    Per l’interazione con l’utente è stata utilizzata la finestra di dialogo di tipo modal implementata nella libreria Bootstrap. Viene inserito nella pagina principale ed è di default nascosto e visualizzabile attraverso il richiamo di una funzione JavaScript. Div con classe modal :    Funzione JavaScript (file script.js) per la visualizzazione dei messaggi sia di errore che di notifica:  /\*showAlert      Funzione per la visualizzazione di un messaggio di alert di tipo modal      Parametri:      - titolo, titolo dell'alert;      - messaggio, corpo del messaggio di alert.  \*/  function showAlert(titolo, messaggio) {      let modal = new bootstrap.Modal(document.getElementById('alert'));      modal.show();      document.getElementById("alertTitolo").innerHTML = titolo;      document.getElementById("alertMessaggio").innerHTML = messaggio;  }  L’esecuzione del calcolo del lotto economico di ordinazione, l’inserimento degli input e la visualizzazione dell’output sono gestiti con la pagina HTML *params.html.* Questa viene visualizzata all’utente attraverso la funzione JavaScript *showParams* contenuta nel file *route.js,* richiamata dal menù dell’applicazione (voce EOQ).  Come mostrato dal codice della funzione:  /\*\*\*\*\*\*\*\*\*\*\*\*showParams    Funzione che richiama l'endpoint /params per il caricamento della porzione di pagina per l'inserimento dei parametri di calcolo dell'EOQ e relativo output.    Valore restituito:    file html: params.html file con i campi da compilare per il calcolo dell'EOQ  \*/  async function showParams() {      try {          /\*richiamo l'endpoint con il metodo POST\*/          await fetch('/params', {              method: "POST"          }).then(function (response) {              /\*restituisco la response dal server in formato testo/html\*/              return response.text();          }).then(function (data) {              /\*Aggiorna la parte della pagina con il nuovo contenuto\*/              document.getElementById('main').innerHTML = data;          });      /\*catturo eventuali errori\*/      } catch (error) {          showAlert('Errore', 'Errore durante il richiamo del form parametri' + error);      }  }  Codice completo:  https://github.com/adora0/eoq\_f/blob/main/public/js/route.js#L62  il dialogo con il server è effettuato con l’utilizzo di funzioni asincrone non bloccanti, per migliorare la reattività e la risposta dell’interfaccia utente. L’utilizzo del metodo *fetch* del browser rende le richieste HTTP verso il server più semplici, leggibili includendo la gestione di eventuali errori.  La porzione di pagina dedicata al calcolo dell’EOQ è composta da due sezioni principali, gestite da diverse funzioni JavaScript per il caricamento dei dati, l’invio al server e la visualizzazione dell’output.  Le sezioni sono:   * tabella dati, contiene le colonne dei parametri di input e di output.   Nell’intestazione sono definiti i pulsanti per il caricamento e la gestione dei dati e per l’avvio dell’elaborazione.   * Form dati di tipo modal per l’inserimento dei dati manualmente, con alcuni controlli di correttezza dei dati.   Nell’intestazione della tabella sono definiti tutti i pulsanti per le operazioni sui dati come il pulsante per il caricamento dati da file e l’inserimento manuale:      Ogni pulsante richiama la relativa funzione JavaScript per il caricamento dei dati. La funzione seguente è per l’inserimento manuale dei dati :  /\*showInserimentoDati      Funzione per la visualizazione del form inserimento dati di tipo modal.      Parametro:      - flagVuoto, boolean per aprire il modal pulito o no. Utilizzato per      mostrarlo di nuovo in caso di errori;      Pulisco e resetto le classi in caso di visualizzazione precedente.  \*/  function showInserimentoDati(flagVuoto) {      /\*istanzio l'oggetto Modal di bootstrap\*/      let modal = new bootstrap.Modal(document.getElementById('datainput'));      /\*visualizzo i campi per l'insermimento dati, se il flag è false        visualizzo       i dati precedentemente insetiti, provengo da        un inserimento parziale o errato\*/      if (flagVuoto) {          let f = document.getElementById("formDati");          let err = document.getElementById('modalError');          err.style.display = 'none';          Array.from(f.elements).forEach(function (input) {              if (input.tagName === 'INPUT') {                  /\*Imposto le classi iniziali e pulisco i campi di input\*/                  input.value = '';                  input.className = 'form-control';              }          });      }      modal.show();  }  Codice completo: <https://github.com/adora0/eoq_f/blob/main/public/js/script.js#L290>  Per l’inserimento dei dati in tabella è stata definita la funzione seguente che per ogni riga dell’oggetto JSON inserisce i relativi dati nelle celle della table HTML:  /\*inserisciDatiInTabella  funzione per il caricamento dei dati nella table HTML  Parametri:  - dati, oggetto JSON con i dati  - svuota, boolean per indicare se pulire la tabella o no. Nella fase di    caricamento da file la tabella va svuotata, con l'inserimento manuale no.  \*/  function inserisciDatiInTabella(dati, svuota) {      //Aggiorna la porzione di pagina con il nuovo contenuto      let tabella = document.getElementById('tabellaDati');      /\*se inserimento da file pulisco la tabella\*/      if (svuota) {          tabella.innerHTML = "";      }      //let dati = JSON.parse(data);      let i = 1;      JSON.parse(dati).reverse().forEach((d) => {          /\*aggiungo una riga\*/          let row = document.createElement('tr');            /\*aggiungo le celle con i dati\*/          let cell = document.createElement('td');          cell.textContent = d.periodo;          cell.setAttribute('name', 'periodo');          cell.classList.add("text");          row.appendChild(cell);  Codice completo : <https://github.com/adora0/eoq_f/blob/main/public/js/script.js#L12>  Per l’esecuzione del calcolo dell’EOQ è definito invece il pulsante    con il relativo pulsante switch per l’attivazione della previsione    La funzione JavaScript associata al pulsante *Calcola EOQ* invia i dati al server ( livello business logic), gestisce eventuali errori e se l’operazione va a buon fine richiama la funzione dedicata per la visualizzazione dei dati:  /\*\*\*\*\*\*\*\*\*\*\*\*elaboraEOQ    Funzione che richiama l'endpoint /elab per il calcolo del valore EOQ. I parametri per il calcolo vengono letti dai campi della pagina html    Valore restituito:    oggetto json:      - EOQ, valore del lotto ecnomico d'acquisto      - CTot, costo totale di riordino      - error. evventuale errore nel calcolo o nei parametri inviati  \*/  async function elaboraEOQ(tabellaDati) {          /\*variabile per il caricamento dei parametri\*/          const data = new URLSearchParams();          /\*trasformo la tabella dati da html a json\*/          let dati=convertiTabella(tabellaDati);          /\*verifico che la tabella non sia vuota\*/          if(dati=="") {              showAlert('Attenzione', 'Tabella dati vuota');              return;          }          document.getElementById("spinner").style.display = 'inline-block';          /\*aggiungo i dati json alla variabile data\*/          data.append("dati",convertiTabella(tabellaDati));            /\*verifico se è stato selezionata la richiesta di previsioni\*/          if(document.getElementById("checkForecast").checked){              data.append("flagPrev",1);          }else{              data.append("flagPrev", 0);          }          /\*richiamo l'endopoint con metodo POST passando i dati\*/          await fetch('/elab', {              method: "POST",              body:data          }).then(function (response) {              /\*se la risposta è 'ok' restituisco i dati json\*/              if(response.ok)                  return response.text();              else              /\*in caso di errore sollevo l'eccezione\*/                  throw new Error('Problema nel modulo Python');          }).then(function (data) {              /\*Se tutto bene, aggiorno la parte della pagina con il nuovo contenuto  \*/              inserisciDatiInTabella(data,true);              /\*aggiorno i risultati\*/              showResult(data);          }).catch(function(error) {              showAlert('Errore', 'Errore durante il calcolo EOQ:' + error);          });          document.getElementById("spinner").style.display = 'none';  }  Codice: <https://github.com/adora0/eoq_f/blob/main/public/js/route.js#L14>  I risultati della funzione di elaborazione sono passati alla funzione showResultper la formattazione dell’output composto da un riepilogo descrittivo sintetico, i valori da inserire nella tabella di output e due grafici per meglio comprendere il trend dei valori di EOQ e domanda.  Porzione del codice della funzione di output:  /\*showResult  Funzione che visualizza i risultati dell'elaborazione.  Vengono mostrati i valori dell'EOQ, dei costi totali,  di ordinazione e mantenimento, oltre ai grafici con  l'andamento della domanda e della dimensione del lotto\*/  function showResult(data) {      let d = JSON.parse(data);      let txtRisultati=document.getElementById("testoRisultati");      clearResult();      txtRisultati.innerHTML="<h4>Risultati elaborazione</h4><br />" + "Periodo: <b>" + d[0].periodo + "</b><br />"  + "EOQ: <b>" + d[0].valEOQ + "</b><br />"  + "Costo mantenimento: <b>" + d[0].valCM + "</b><br />"  + "Costo ordinazione: <b>" + d[0].valCO + "</b><br />"  + "Costo totale: <b>" + d[0].valCT + "</b><br />"      d.sort((a, b) => a.periodo - b.periodo);  Codice completo:  <https://github.com/adora0/eoq_f/blob/main/public/js/script.js#L328>  Il livello deputato alla **logica di business** rappresenta la struttura portante dell'applicativo. Il file JavaScript *app.js* istanzia il server web realizzato con NodeJs e il framework Express. Il server gestisce le richieste HTTP, espone gli endpoint e implementa le funzioni necessarie per il funzionamento dell'applicazione.  Grazie a NodeJs è possibile creare un server web minimale, semplice da gestire e altamente scalabile. L’esecuzione dello script app.js infatti può essere eseguita in locale sulla propria macchina o in alternativa su un server remoto aziendale.  Il server web viene configurato definendo i seguenti parametri:   * la porta di ascolto per connessioni http, definita dalla variabile *port*, in questo caso è definita la 3000 ma la scelta è arbitraria; * la cartella dove risiedono i file statici (html, css etc.) accessibili direttamente dal client, costituiscono di fatto il livello di presentazione; * l’encoding dei parametri scambiati durante le request e response del protocollo HTTP, per quest’applicazione è stato scelto il formato JSON.   Codice definizione del server web:  /\*librerie utilizzate per l'applicazione server\*/  const express = require('express');  const path = require('path');  const fs = require('fs');  /\*oggetto per l'esecuzione di sotto-processi\*/  const { spawn } = require('child\_process');  /\*istanza dell'applicazione express\*/  const app = express();  /\*porta di ascolto\*/  const port = 3000;  /\*imposto il folder public con le parti statiche dell'applicazione\*/  app.use(express.static('public'));  /\*imposto express per la ricezione dei parametri in JSON via POST\*/  app.use(express.urlencoded({ extended: true }));  app.use(express.json());  /\*inizio ascolto del server su porta 3000\*/  app.listen(port, () => {      log(`Server web in ascolto sulla porta:${port}`,'INFO');  });  Vengono poi definiti gli endpoint, che associano una funzione di callback, quindi un determinato comportamento dell’applicazione, ad una specifica URL e metodo HTTP.  Ad esempio l’endpoint definito dal codice:  /\*endpoint homepage applicazione - index.html\*/  app.get('/', (req, res) => {      res.sendFile(path.join(\_\_dirname, 'public' , 'index.html'));      log('Inviato file index.html','INFO');  });  definisce l’azione da compiere (funzione di callback) quando avviene una connessione alla radice dell’indirizzo dell’applicazione, se il server web viene eseguito in locale corrisponde all’indirizzo [*http://localhost:3000/*](http://localhost:3000/). La funzione di *callback* restituirà l’homepage dell’applicazione, inviando la pagina *index.html* al client.  Per il calcolo del lotto economico di ordinazione la funzione associata all’endpoint è definita dal seguente codice:  /\*endpoint /elab      richiamo delle funzioni Python per il calcolo dell'EOQ  \*/  app.post('/elab', (req, res,next) => {      log('Richiesta elaborazione EOQ', 'INFO');      const datiAcquisti=req.body;        /\*Creo il processo di esecuzione codice python, passando i dati come      argomento e il timeout per la gestione degli errori \*/      const python = spawn('python',                           [path.join(\_\_dirname,'pyelab','eoq\_elab.py'),                           JSON.stringify(datiAcquisti)]);      /\*Funzione callback per la lettura dell'output del sotto-processo \*/      python.stdout.on('data', function (data) {          log(`Codice python eseguito con successo`,'INFO');          res.send(data);      });      /\*Funzione callback per la lettura di errori del sottoprocesso\*/      python.stderr.on('data', (data) => {          log(`Errore: ${data}`,'ERROR');      });      /\*Funzione callback di termine del sotto-processo.      \* se il code è diverso da zero l'esecuzione non è andata a buon fine e      \* invio l'HTTP code 500 - internal server error      \*/      python.on('close', (code) => {          if(code!=0){              log(`Errore esecuzione codice python code:[${code}]`,'ERROR');              res.status(500).send('KO');          }      });    });    la funzione gestisce le richieste HTTP di tipo POST per la ricezione dei dati in formato JSON e attraverso il modulo *child\_process* crea un nuovo processo con la funzione *spawn* per l’esecuzione del programma Python dedicato al calcolo dell’EOQ. Terminata l’esecuzione i risultati vengono inviati al client attraverso la response con la funzione *send,* il tutto gestendo eventuali errori.  Il modulo child\_process di fatto richiama il **livello di elaborazione dati** che implementa l’algoritmo di calcolo dell’EOQ e dell’eventuale stima della domanda del prodotto. Il codice **Python** è contenuto nel file *eoq\_elab.py* :  import numpy as nm  import pandas as pd  #package StatsForecast https://github.com/Nixtla/statsforecast  from statsforecast import StatsForecast  from statsforecast.models import AutoARIMA  from datetime import datetime  #leggo i dati passati in json      dati = json.loads(sys.argv[1])      dati\_tabella = json.loads(dati["dati"]) #tabella dati      flag\_previsione = json.loads(dati["flagPrev"])#flag se richiesta previsione      #creo il DataFrame del package Pandas      dati\_mag=pd.DataFrame(dati\_tabella)        #se flag\_previsione è uguale a 1 eseguo la previsione      if flag\_previsione == 1 :          dati\_c=calcola\_forecast(dati\_mag)          #riga previsione          previsione=dati\_mag.head(1).copy()          previsione.loc[0,('valD')]=dati\_c.loc[0,('AutoARIMA')]          previsione.loc[0,('periodo')]=dati\_c.loc[0,('ds')]          #aggiungo la previsione al dataset          dati\_mag = pd.concat([previsione,dati\_mag])        #richiamo la funzione per il calcolo dell'EOQ      calcola\_EOQ(dati\_mag)  Codice completo: <https://github.com/adora0/eoq_f/blob/main/pyelab/eoq_elab.py>  Il programma dopo aver richiamato le librerie necessarie (Pandas, Numpy, StatsForecast) trasforma i dati ricevuti dal formato JSON nella struttura DataFrame del pacchetto Pandas.  Esempio struttura dati JSON :  [{"periodo":"2024","valC":"3.50","valS":"150","valH":"0.10","valD":"19000"},  {"periodo":"2023","valC":"3.45","valS":"148","valH":"0.09","valD":"18500"},  {"periodo":"2022","valC":"3.40","valS":"146","valH":"0.09","valD":"18000"},  {"periodo":"2021","valC":"3.35","valS":"144","valH":"0.08","valD":"17500"}]  Oltre ai dati viene passata come parametro la variabile booleana *flag\_previsione* che indica se al calcolo deve essere aggiunta la previsione della domanda per il calcolo dell’EOQ di un anno ulteriore.  Il calcolo del lotto e della previsione sono stati implementati in due funzioni distinte, **funzione calcolo EOQ** :  #calcola\_EOQ - Modulo per il calcolo del lotto economico di ordinazione  # Parametri:  #  df, dataframe con i parametri di calcolo  # Valore restituito:  #  dataframe, con i valori di EOQ, numero lotti e costi totali  def calcola\_EOQ(df):      #applico la formula per il calcolo dell"EOQ : sqrt((2\*D\*S)/(H))      df["valEOQ"]=round(nm.sqrt((2 \* df["valS"] \* df["valD"] / df["valH"])),0)      #calcolo numero di lotti: D/EOQ      df["valNL"]=round(df["valD"]/df["valEOQ"], 0)      #calcolo costi immagazzinamento      df["valCM"]=round(df["valH"] \* (df["valEOQ"] / 2 ),0)      #calcolo costi di ordinazione      df["valCO"]=round((df["valD"]/df["valEOQ"])\*df["valS"],0)      #calcolo costo totale      df["valCT"]=df["valCM"] +df["valCO"] +  (df["valD"] \* df["valC"])      return df  la funzione non calcola solo il valore dell’EOQ ma anche il numero di lotti nel periodo, il costo di immagazzinamento, il costo di ordinazione e il costo totale. I dati ricevuti possono essere relativi ad un periodo o a più periodi, il calcolo verrà eseguito per tutti i dati disponibili. Se è richiesta la previsione viene richiamata la funzione **calcola forecast :**  #calcola\_forecast - Modulo per il calcolo della previsione domanda  # Utilizza il modello AutoARIMA del package StatsForecast  # https://nixtlaverse.nixtla.io/statsforecast/src/core/models.html  # Parametri:  #  df, dataframe con la serie storica dei dati  # Valore restituito:  #  dataframe, completo con un periodo di previsione.  def calcola\_forecast(df):      #imposto il dataframe per l'utilizzo del modello      data\_f = pd.DataFrame({          'unique\_id':'eoq\_f',          'ds':df['periodo'],          'y':df['valD']      })        #configuro il modello AutoARIMA, con frequenza annuale      sf = StatsForecast(          models = [AutoARIMA(          stepwise=True, #scelta dei parametri velocizzata          seasonal=False, #disabilito stagionalità          season\_length = 1, #dati annuali)],          freq = 1 #frequenza annuale      )      #addestro il modello con i dati      sf.fit(data\_f)      #richiamo la predict per stimare un periodo      eoq\_f=sf.predict(h=1)      eoq\_f['AutoARIMA']=round(eoq\_f['AutoARIMA'],0)      return eoq\_f  il modello AutoARIMA viene impostato per previsioni annuali e che non prevedono stagionalità (effetti di alcuni fenomeni periodici sul valore di alcune variabili, come ad esempio l’aumento della domanda di beni durante il periodo delle festività natalizie) questo per essere il più generici possibili rispetto al tipo di prodotto del quale si vuole stimare la domanda.  Terminato il calcolo vengono restituiti i dati sempre in formato JSON:    #trasformo il dataframe in json e lo restituisco      print(dati\_mag.to\_json(orient="records"))      #codice fine processo OK      exit(0)  l’esecuzione torna al web server che invierà i dati al client.  **Utilizzo e funzionamento**  L’utilizzo base dell’applicazione è rappresentato dal diagramma UML del caso d’uso in figura 6.      Figura 6 - Caso d'uso calcolo EOQ  L’applicazione si presenta con una pagina iniziale con le informazioni di base per l’utilizzo e un menù orizzontale (Figura 7)    Figure 7 - Home page dell'applicazione  Selezionando dal menu la voce ***EOQ*** l’utente ha due possibilità di inserimento dati (Figura 8), manuale compilando un modulo (*form – Figura 9*) o effettuando il caricamento di un file esterno, di tipo *Comma Separated Value*, per velocizzare l’inserimento in caso di dati riguardanti più periodi.    Figure 8 - Pagina principale per l'inserimento dei dati e visualizzazione    Figure 9 - Form inserimento dati modalità manuale  In entrambe le modalità di inserimento le informazioni da fornire sono le seguenti:   * **Periodo**, di tipo numerico che rappresenta il periodo di riferimento dei dati, generalmente l’anno; * **Costo unitario** **(C)**, costo unitario di acquisto del prodotto; * **Costo di gestione dell’ordine (S)**, è il costo di emissione di un ordine, considerato per definizione costante nel periodo; * **Costo mantenimento (H)**, è il costo unitario per il mantenimento di un prodotto nel periodo; * **Domanda (D)**, è il valore della domanda di prodotto espressa come media del periodo.   I dati inseriti vengono mostrati nella tabella in Figura 10    Figure 10 - Tabella elaborazione e gestione dati  La tabella è predisposta per lavorare con dati in serie storica e sono predisposte le colonne per i valori calcolati di output:   * **EOQ**, valore del lotto economico di ordinazione; * **Numero lotti**, numero dei lotti da ordinare nel periodo * **Costo totale**, costo totale di acquisto calcolato sommando i costi di approvvigionamento, mantenimento e acquisto secondo la formula   In testata di tabella sono presenti le seguenti funzioni:   * Carica dati, modalità inserimento dati da file; * Aggiungi dati, modalità inserimento dati manuale; * Svuota tabella, per ripulire la tabella e inserire nuovi dati; * Selettore di tipo *switch*, per abilitare il calcolo dell’EOQ per un periodo ulteriore con stima del valore della domanda; * Calcola EOQ, avvio della procedura di calcolo del lotto economico di ordinazione per tutti i periodi presenti in tabella.   L’output finale è stato integrato anche con il dettaglio della composizione del costo totale e i grafici a linea per evidenziare l’andamento in serie storica del lotto di ordinazione e della domanda annua.        Figure 11 - Output di elaborazione |
| **Campi di applicazione:** | Il progetto è stato sviluppato con l’obiettivo di realizzare un’applicazione che possa essere adottata in realtà aziendali medio piccole, che necessitino di uno strumento per l’analisi dei costi di gestione dalle scorte di magazzino e la corretta valutazione dell’acquisto di prodotti. La tecnologia utilizzata per l’implementazione dell’applicativo, è stata selezionata per consentirne l’esecuzione sia su dispositivi in locale (personal computer) che in remoto (server).  Grazie alla flessibilità della configurazione e modularità del codice, ad esempio le diverse componenti che costituiscono l’applicativo possono essere ospitate volendo su diversi server. Il livello di elaborazione può essere adeguato agevolmente per essere eseguito da un altro web server collocato geograficamente altrove.  Altro elemento è rappresentato dalla facilità di integrazione con altri sistemi, ad esempio l’esecuzione del calcolo può essere eseguita semplicemente richiamando il relativo l’indirizzo (URL) <http://localhost:3000/elab>, con metodo POST inviando e ricevendo i dati strutturati nel formato JSON senza alcuna interazione con l’utente, tipica dei sistemi RESTful[14]. |
| **Valutazione dei risultati (potenzialità e criticità):** | Il prodotto di questo project work tenta di rappresentare l’esempio di un applicativo moderno e facilmente scalabile, caratteristiche che assumono oggi la priorità in questo campo. Dal punto di vista del calcolo del lotto economico di ordinazione si tratta comunque di un software limitato al calcolo del caso base, ma l’architettura ideata si presta ad eventuali estensioni e integrazioni per l’aggiunta di funzionalità che siano più aderenti alla realtà nel campo della gestione delle scorte.  Il linguaggio Python unito al framework NodeJS sono a mio avviso scelte vincenti se consideriamo il sempre più presente utilizzo di modelli di Intelligenza Artificiale. Entrambi gli strumenti infatti sono dotati di numerose librerie che consento un’agile integrazione con i numerosi modelli oggi disponibili. La stessa libreria StatsForecast, utilizzata proprio per dare enfasi alla possibilità di integrazione di moduli aggiuntivi, dispone già di un modello generativo AI addestrato per la stima di serie temporali (TimeGPT-1 <https://docs.nixtla.io/>). |