



UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in INFORMATICA

PROGETTAZIONE E SVILUPPO DI UNO STRUMENTO DI CLASSIFICAZIONE BASATO SU GRAFI DI REGOLE

Relatore:

Prof. Massimo SANTINI

Tesi di Laurea di:
Andrea Luigi CAMBIAGHI
Matricola 943005

Anno Accademico: 2022/2023

*“The weaker the data available upon which to base one’s conclusion,
the greater the precision which should be quoted in order to give the
data authenticity. “*

— Norman Ralph Augustine

Indice

1	Introduzione	1
1.1	Analisi dati manuale	1
1.2	Analisi dati automatica	2
1.3	Analisi dati metabolomici	2
1.3.1	Esempio pratico	3
1.4	Possibili utilizzi	3
2	Tecnologie utilizzate	4
2.1	Introduzione	4
2.2	Frontend	5
2.2.1	HTML	5
2.2.2	CSS	5
2.2.3	Javascript	6
2.2.4	VueJS	7
2.2.5	BulmaJS	8
2.2.6	IziToast	9
2.2.7	Sweetalert	9
2.2.8	BaklavaJS	9
2.2.9	JQuery	12
2.2.10	AJAX	13
2.2.11	JSON	14
2.3	Backend	16
2.3.1	Python	16
3	Descrizione soluzione	20
3.1	Introduzione	20
3.2	Scelte implementative	21
3.2.1	Interfaccia	21
3.2.2	Flask	22
3.3	Architettura del software	22
3.3.1	Classi principali BaklavaJS	23

3.3.2	Comportamento Python	25
3.3.3	Comunicazione Python - Javascript	26
3.4	Utilizzo applicazione	26
3.4.1	Avvio applicazione	26
3.4.2	Creazione di un nodo	29
3.5	Creazione delle regole	34
3.5.1	Caricare un Grafo	35
3.5.2	Formato JSON	37
3.6	Modalità visualizzazione	41
3.7	Applicazione delle regole	46
3.7.1	Script controllo condizioni	46
3.7.2	Script applicazione delle regole	46
3.8	Esecuzione tramite script	47
4	Conclusione	50
4.1	Sviluppi Futuri	51
Sitografia		52
Bibliografia		53

Elenco delle figure

2.1 Esempio utilizzo BulmaJS	8
2.2 Esempio di notifica di IziToast	9
2.3 Esempio di alert realizzato con SweetAlert	9
2.4 Codice per realizzaralo	9
2.5 Esempio di grafo realizzato con BaklavaJS	11
2.6 Esempio di grafo realizzato con BaklavaJS	11
2.7 Esempio di grafo realizzato con BaklavaJS	12
2.8 Esempio di grafo realizzato con BaklavaJS	12
2.9 Esempio di codice di una richiesta AJAX	14
2.10 Esempio di codice JSON	15
3.1 Schermata iniziale	28
3.2 Selezione nodo "Foglia"	29
3.3 Selezione nodo "Start"	30
3.4 Selezione nodo "Nodo" (nodo intermedio)	30
3.5 Nodo categorico	31
3.6 Nodo Range	32
3.7 Nodo Range	33
3.8 Esempio creazione nodi	34
3.9 Alert errore nodo start	35
3.10 Alert errore cicli	35
3.11 Alert errore nodo con nome esistente	35
3.12 Esempio grafo	37
3.13 Richiesta file CSV	41
3.14 Nodo con condizione problematica	42
3.15 Esempio funzionante completo	43
3.16 Grafo in modalità visualizzazione	44
3.17 Nodo Start selezionato	44
3.18 Nodo Foglia selezionato	45
3.19 Schermata caricamento file	48
3.20 File caricati	49

Capitolo 1

Introduzione

L'analisi dei dati è diventata una parte essenziale della tecnologia moderna, poiché diventa sempre più importante capire il significato di tutte le informazioni disponibili. Con l'enorme quantità di dati a disposizione oggi, è essenziale avere strumenti efficienti e avanzati per gestirli e analizzarli in modo accurato. È necessario adottare approcci innovativi e sofisticati per affrontare la complessità e la varietà dei dati disponibili.

1.1 Analisi dati manuale

La gestione delle informazioni tramite carta e penna, oltre a essere suscettibile a errori di trascrizione, diventa rapidamente impraticabile quando ci si trova a dover analizzare grandi volumi di dati.

Allo stesso modo, l'utilizzo di strumenti comuni come fogli di calcolo, tra cui Excel, sebbene diffusi e utilizzati ampiamente, presenta limitazioni significative in termini di scalabilità e complessità analitica.

Gli strumenti basati su fogli di calcolo sono spesso la scelta tradizionale per la gestione dei dati, ma incontrano difficoltà quando si tratta di affrontare complessità crescenti e volumi di dati massicci.

La manipolazione manuale dei dati in fogli di calcolo può essere un processo laborioso e soggetto a errori umani, con la possibilità di trascrizioni errate o manipolazioni inadeguate. Inoltre, tali strumenti sono spesso limitati nelle funzionalità di analisi avanzate e nella gestione di dati eterogenei.

1.2 Analisi dati automatica

Nel contesto attuale degli strumenti di analisi dati, emerge la necessità di soluzioni più complete e automatizzate per gestire efficacemente la complessità legata all’analisi di grandi quantità di dati. Molti degli strumenti attualmente disponibili presentano limiti nelle funzionalità offerte o richiedono competenze tecniche avanzate, rendendo l’analisi dati una sfida per un vasto pubblico di utenti.

L’obiettivo fondamentale di questa tesi è colmare questa lacuna, proponendo uno strumento innovativo basato sull’utilizzo di grafi di regole per l’analisi automatica dei dati. Attraverso questo approccio avanzato, l’obiettivo è rendere accessibile a un’ampia gamma di utenti la capacità di analizzare tabelle di grandi dimensioni con facilità e intuizione. La mancanza di strumenti che possano eseguire questa attività in modo completo e, allo stesso tempo, intuitivo, è ciò che motiva la ricerca e lo sviluppo di una soluzione che possa semplificare significativamente il processo di analisi dei dati, rendendolo alla portata di tutti gli utenti interessati, indipendentemente dal livello di competenza tecnica.

1.3 Analisi dati metabolomici

Nel campo della chimica, e più specificamente nell’ambito della metabolomica, i professionisti, gli studiosi e i ricercatori del dipartimento di chimica si trovano spesso a dover affrontare una sfida considerevole: la gestione e l’analisi di estesi set di dati metabolomici racchiusi in tabelle di notevoli dimensioni.

Fino ad oggi, l’analisi è stata condotta principalmente utilizzando strumenti convenzionali come fogli di calcolo e la creazione manuale dei grafici. che per lungo tempo sono stati strumenti utilizzati nelle pratiche di analisi, presenta ormai limiti evidenti in termini di scalabilità e precisione, come precedentemente discusso. Tuttavia, con l’incremento massiccio dei dati disponibili, è diventato indispensabile adottare approcci più innovativi e complessi per condurre queste analisi.

Affrontare grandi volumi di dati attraverso questi metodi tradizionali richiede sforzi considerevoli e, ciò che è più critico, può portare a errori umani e inefficienze nel processo di analisi.

Dunque, la chiara necessità di una metodologia più avanzata e automatizzata per gestire in modo efficiente questa mole di dati complessi è già emersa, guidando la ricerca di soluzioni innovative.

1.3.1 Esempio pratico

Immaginiamo di avere un dataset metabolomico con informazioni su diversi composti chimici rilevati in campioni biologici. Tradizionalmente, la classificazione di queste molecole potrebbe richiedere ore di lavoro manuale, con la creazione di tabelle complesse su fogli di calcolo e la definizione di regole attraverso annotazioni cartacee.

Il nuovo strumento permetterà all'utente di importare i propri dati nel formato csv direttamente nell'applicazione, utilizzando un'interfaccia intuitiva e uno strumento grafico. Inoltre, l'utente potrà progettare il proprio grafo di regole personalizzato.

Il sistema analizzerà quindi automaticamente i dati in base alle regole definite dall'utente. Ad esempio, potrebbe identificare gruppi di composti chimici con caratteristiche simili in base a regole predefinite, come la presenza di specifici picchi di massa o la correlazione tra determinate concentrazioni.

In questo modo, la progettazione del grafo diventa un processo accessibile anche a utenti senza particolari competenze tecniche, consentendo una personalizzazione precisa dell'analisi in base alle esigenze specifiche del progetto.

1.4 Possibili utilizzi

Sebbene questo strumento sia stato sviluppato originariamente per soddisfare le esigenze specifiche dell'analisi dei dati metabolomici, la sua versatilità e flessibilità consentono l'applicazione in una vasta gamma di campi.

Ad esempio, professionisti e ricercatori nel campo della biologia possono utilizzare la piattaforma per esplorare le relazioni tra diversi geni o proteine in un set di dati.

Nel contesto aziendale, analizzando dati finanziari complessi, è possibile identificare pattern e correlazioni cruciali per prendere decisioni strategiche informate.

Persino nell'ambito delle scienze sociali, questo strumento può essere impiegato per esaminare schemi e tendenze nei dati di sondaggi o di analisi dei sentimenti.

La forza distintiva di questo strumento risiede nella sua adattabilità, consentendo agli utenti di creare grafi di regole personalizzati per soddisfare le specifiche necessità del proprio dominio. Questa flessibilità apre nuove prospettive per l'analisi dei dati, offrendo un approccio innovativo e accessibile in vari contesti professionali e accademici.

Capitolo 2

Tecnologie utilizzate

2.1 Introduzione

Questo capitolo costituisce un'analisi dettagliata delle metodologie e delle tecnologie utilizzate all'interno del mio lavoro. L'obiettivo principale di questa sezione è presentare chiaramente le tecnologie front-end e back-end impiegate, senza entrare nei dettagli delle motivazioni sottostanti. La comprensione di queste componenti è fondamentale per contestualizzare adeguatamente il mio lavoro e agevolare la chiara interpretazione dei risultati ottenuti.

Nella prima parte di questo capitolo, saranno esaminate le tecnologie front-end adoperate per l'implementazione e la visualizzazione dell'interfaccia utente. Verranno presentati gli strumenti e le piattaforme che hanno contribuito a creare un'esperienza utente efficace.

Nella seconda parte, ci concentreremo sulle tecnologie back-end che hanno sostenuto il funzionamento e la gestione dei dati. Saranno descritti i framework e le infrastrutture che hanno permesso l'elaborazione dei dati. L'analisi approfondita di queste tecnologie offrirà una chiara visione delle risorse tecniche coinvolte nell'ambito del mio lavoro.

2.2 Frontend

2.2.1 HTML

HTML, acronimo di "Hyper Text Markup Language", rappresenta un linguaggio di markup fondamentale per la creazione di pagine web e interfacce utente online. Fondato sulle nozioni di SGML (Standard Generalized Markup Language) e XML (eXtensible Markup Language), HTML utilizza tag o, più formalmente, elementi, per definire la struttura e il contenuto delle pagine in modo coerente e ben strutturato.

La sua fama non è solo dovuta alla sua semplicità, ma anche alla sua efficacia grazie alla chiara definizione di ogni elemento. Ogni tag/elemento ha un significato specifico che contribuisce all'organizzazione strutturata dei contenuti, che possono includere paragrafi, titoli, liste, tabelle e altri elementi. Questa struttura ben definita permette ai browser web e ad altri dispositivi di interpretare e presentare uniformemente i contenuti, indipendentemente dalla piattaforma utilizzata.

La capacità di HTML di offrire una struttura chiara e standardizzata favorisce la creazione e l'interpretazione delle pagine web. La distinzione tra pagine "statiche" e "dinamiche" dipende dalla natura dei contenuti: le prime mostrano informazioni fisse, mentre le seconde possono essere personalizzate in tempo reale in base all'interazione dell'utente o ad altre variabili.

HTML, con la sua presenza diffusa e flessibilità, si presenta come un linguaggio fondamentale nell'ambito del web. È il linguaggio principale per la creazione di pagine web statiche e dinamiche, consentendo agli sviluppatori di realizzare interfacce web personalizzate per soddisfare le esigenze degli utenti.

Il linguaggio di markup HTML è diffuso nell'ambito dei programmati, celebre per il suo rispetto degli standard aperti. Tale attributo sottolinea la sua continua evoluzione in risposta alle mutevoli dinamiche e alle sfide inerenti al contesto web contemporaneo. La sua integrazione armoniosa con tecnologie complementari, come CSS (Cascading Style Sheets) e JavaScript, ne fa uno strumento di elevata efficacia per migliorare l'aspetto visivo e l'interattività delle pagine web.

2.2.2 CSS

Nel settore delle applicazioni online e dei servizi internet, l'aspetto visivo e l'estetica delle interfacce utente rivestono un ruolo fondamentale. Durante la fase di progettazione e sviluppo di un'applicazione web o di un sito internet, l'obiettivo primario consiste spesso nella creazione di un'esperienza utente coinvolgente, intuitiva ed esteticamente piacevole.

In questo contesto, **CSS**, acronimo di Cascading Style Sheets, rappresenta un elemento chiave nel campo dello sviluppo web. **CSS** consente ai designer e ai programmatore di definire la presentazione e la formattazione degli elementi **HTML**, offrendo un controllo preciso sull'aspetto visivo dell'interfaccia utente.

CSS opera in collaborazione con l'**HTML** (Hypertext Markup Language), separando chiaramente la presentazione e il design dall'essenza del contenuto. Questa separazione consente un design flessibile e personalizzato delle pagine web, mantenendo allo stesso tempo la struttura dei contenuti.

I fogli di stile **CSS** possono essere esterni, interni o in linea. Gli stili esterni controllano l'aspetto di elementi su diverse pagine del sito, mentre gli stili interni influenzano l'aspetto di una singola pagina.

Gli stili in linea, invece, controllano specifici elementi all'interno di una pagina.

L'utilizzo di **CSS** richiede la scrittura di regole di stile mediante un editor di testo, e questi stili vengono collegati alle pagine **HTML** per influenzarne l'aspetto. La versatilità di **CSS** e la sua capacità di creare pagine web esteticamente gradevoli lo rendono un elemento essenziale per la personalizzazione e l'ottimizzazione delle interfacce web.

2.2.3 Javascript

JavaScript, basato sullo standard **ECMAScript**, è un linguaggio di scripting ampiamente impiegato nello sviluppo web, essenziale per la creazione di applicazioni interattive. La sua versatilità e la capacità di eseguire codice direttamente nel browser web lo rendono uno strumento ideale per una vasta gamma di funzioni.

Questo linguaggio è noto per gestire eventi come clic del mouse, pressioni dei tasti e input dell'utente, permettendo risposte immediate e interattive. **JavaScript** consente di manipolare il **DOM** (Document Object Model), la struttura gerarchica degli elementi di una pagina web, permettendo così la creazione di interfacce utente reattive, che rispondono in tempo reale agli input dell'utente, e dinamiche, in quanto possono adattarsi e modificarsi durante l'interazione.

Inoltre, **JavaScript** agevola la comunicazione con servizi web esterni, permettendo il recupero e l'aggiornamento dinamico dei dati. Questa funzionalità apre la porta all'integrazione di contenuti dinamici, migliorando l'aggiornamento e la pertinenza delle informazioni visualizzate.

In passato, quando le pagine web erano prevalentemente statiche e non potevano rispondere alle interazioni degli utenti, **Java**, un linguaggio differente da **JavaScript**, era spesso utilizzato per alcune di queste funzionalità. **JavaScript** è emerso come tecnologia latto browser per rendere più dinamiche le applicazioni web, permettendo ai browser di rispondere alle azioni degli utenti e modificare il layout dei contenuti della pagina web.

Oggi, **JavaScript** è ampiamente utilizzato sia lato client che lato server, e i suoi casi d'uso si sono evoluti nel corso degli anni.

Un'estensione significativa di **JavaScript** è **TypeScript**, che aggiunge supporto per il controllo dei tipi. Questo fornisce una maggiore robustezza al codice, facilitando la manutenzione e migliorando la scalabilità dei progetti. Molte organizzazioni e sviluppatori preferiscono **TypeScript** per progetti di grandi dimensioni in quanto offre un ambiente di sviluppo più strutturato e predittivo.

2.2.4 VueJS

Vue è un framework **JavaScript** open source progettato per la creazione di interfacce utente (UI, User Interface). È uno dei framework più diffusi e apprezzati per semplificare lo sviluppo web. **Vue** si focalizza sulla view layer e si integra senza sforzi in progetti di sviluppo front-end di vasta portata. La sua reputazione deriva dalla sua notevole semplicità d'uso e dalla capacità di creare UI reattive e progressive. Il termine "reattive" si riferisce alla capacità del framework di rispondere dinamicamente ai cambiamenti nei dati senza la necessità di aggiornare l'intera pagina. D'altra parte, l'aggettivo "progressive" sottolinea la facilità con cui **Vue** può essere gradualmente implementato o migliorato all'interno di un progetto esistente.

Vue offre diverse modalità di integrazione, consentendo agli sviluppatori di iniziare il loro percorso con il framework in vari modi, ad esempio utilizzando la **CDN**, **npm** o i comandi della **@vue/cli**.

Il framework presenta due modalità di programmazione: Options API e Composition API. I componenti **Vue** sono concepiti come una combinazione di oggetti **JavaScript**, che gestiscono i dati dell'applicazione, e una sintassi di template basata su **HTML** che mappa la struttura del **DOM** sottostante. La leggerezza di **Vue** si manifesta nella sua dimensione complessiva ridotta e nella sua capacità di mantenere elevate prestazioni anche in progetti di grandi dimensioni. Questa caratteristica contribuisce a una migliore efficienza e velocità di caricamento delle pagine.

Alcune delle caratteristiche chiave di **Vue** includono la sua solidità del sistema di strumenti, la flessibilità e la possibilità di creare applicazioni a pagina singola (**SPA**, Single Page Applications) in modo intuitivo e performante.

Inoltre, **Vue** utilizza il **virtual DOM**, che è anche utilizzato da altri framework come **React** e **Ember**, per migliorare le prestazioni e l'esperienza utente. La comunità di sviluppatori di **Vue** è in costante crescita e la documentazione ufficiale è molto completa.

2.2.5 BulmaJS

BulmaJS rappresenta un'estensione in JavaScript del framework CSS Bulma, progettata per migliorare l'aspetto visivo e l'interattività dei progetti web.

La libreria offre un'ampia gamma di componenti predefiniti e classi CSS, semplificando così l'implementazione di stili e comportamenti dinamici. Senza approfondire specifiche modalità d'uso, si analizza il contributo di BulmaJS all'arricchimento delle interfacce web. Una caratteristica distintiva di BulmaJS risiede nella sua fondazione su Flexbox, un sistema di layout flessibile. Quest'aspetto riveste importanza poiché semplifica la gestione di layout responsive, adattando automaticamente l'interfaccia utente a diverse dimensioni di schermo e dispositivi. Ciò migliora notevolmente l'esperienza utente, consentendo un'adeguata presentazione del progetto su schermi di varie dimensioni.

Inoltre, BulmaJS offre la possibilità di personalizzare dettagli mediante l'utilizzo di classi CSS aggiuntive o personalizzate, consentendo l'adattamento del design e del comportamento dell'interfaccia utente alle specifiche esigenze del progetto.

Un ulteriore vantaggio derivante dall'utilizzo di BulmaJS è la sua appartenenza a una comunità web attiva, che facilita l'accesso a risorse, documentazione e soluzioni immediate per affrontare problematiche comuni. Tale comunità dinamica semplifica la gestione e la manutenzione del progetto, agevolando il mantenimento di un'applicazione web conforme alle aspettative degli utenti.

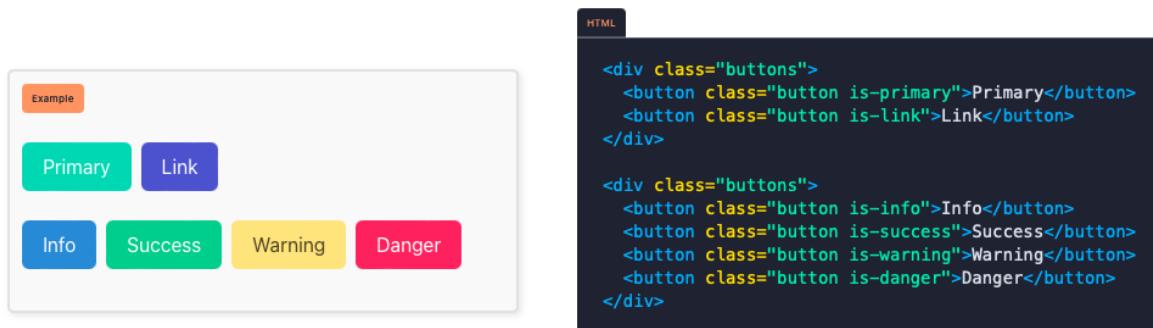


Figura 2.1: Esempio utilizzo BulmaJS

2.2.6 IziToast

Izitoast è una libreria JavaScript che arricchisce l'interfaccia utente fornendo notifiche a comparsa facili da usare e personalizzabili.

Queste notifiche possono essere utilizzate per comunicare messaggi importanti, avvisi o aggiornamenti agli utenti in modo visivamente attraente e non invadente.

Izitoast semplifica il processo di aggiunta di messaggi di notifica alle applicazioni web, migliorando complessivamente l'esperienza dell'utente.

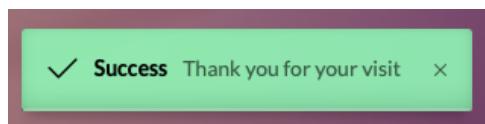


Figura 2.2: Esempio di notifica di IziToast

2.2.7 SweetAlert

SweetAlert è una libreria JavaScript che offre un modo semplice ed elegante per visualizzare popup di avviso personalizzati sulle pagine web.

Questi popup possono essere utilizzati per avvisi, conferme o messaggi informativi e sono progettati per essere belli e facili da personalizzare.

SweetAlert semplifica l'aggiunta di queste finestre di dialogo al proprio sito web, migliorando l'interazione con gli utenti e migliorando l'aspetto generale dell'interfaccia.

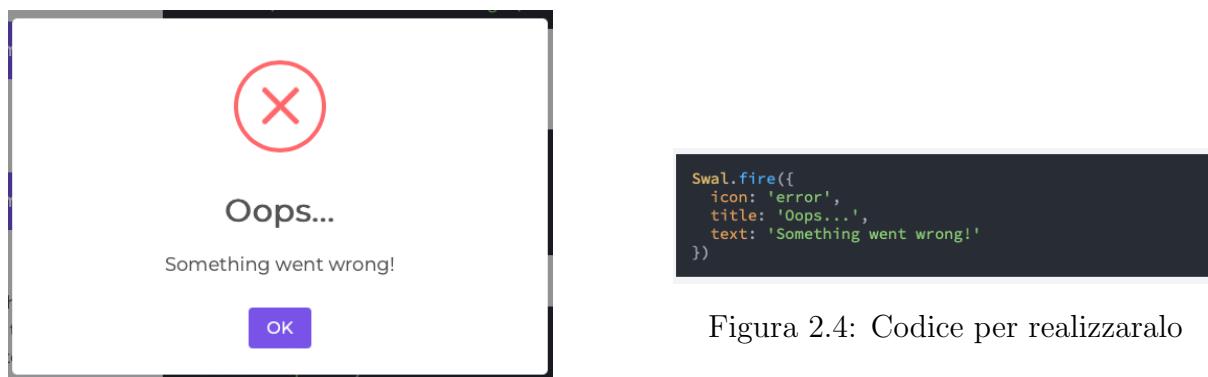


Figura 2.3: Esempio di alert realizzato con SweetAlert

Figura 2.4: Codice per realizzaralo

2.2.8 BaklavaJS

BaklavaJS è una libreria avanzata per la gestione di grafi in applicazioni web. Questa libreria si focalizza sulla manipolazione e la gestione di grafi composti da nodi e archi, rappresentando relazioni complesse tra entità in modo altamente efficiente.

BaklavaJS offre agli sviluppatori uno strumento potente per la creazione, la modifica e l'analisi di grafi, consentendo una vasta gamma di applicazioni, dall'elaborazione delle reti sociali alla gestione dei flussi di dati. Per "analisi di grafi", si intende l'esplorazione delle connessioni e delle proprietà dei nodi e degli archi, facilitando la comprensione delle relazioni all'interno del grafo.

La versatilità di **BaklavaJS** si riflette nella sua capacità di consentire agli sviluppatori di creare grafi di varie tipologie, inclusi grafi orientati e non orientati, grafi pesati e non pesati, e molto altro.

Inoltre, **BaklavaJS** è accompagnata da una documentazione completa che guida gli sviluppatori nell'utilizzo della libreria. Questa risorsa è di grande aiuto per sviluppatori di diversi livelli di esperienza, sia per coloro che si avvicinano per la prima volta ai grafi che per esperti del settore.

Una caratteristica notevole di **BaklavaJS** è la sua indipendenza dagli utenti finali. Gli sviluppatori possono sfruttare la libreria per creare e gestire grafi senza dover fare affidamento su terze parti o sviluppatori esterni, garantendo la sicurezza e l'autonomia dei progetti. Con ciò si intende che **BaklavaJS** permette agli sviluppatori di mantenere il controllo completo sulle funzionalità e l'implementazione del grafico, senza dover dipendere da contributi esterni o vincoli di utilizzo.

D'altra parte, **BaklavaJS** è una libreria open source basata su **VueJS** (un framework **JavaScript** ampiamente riconosciuto, che garantisce un'interfaccia utente altamente reattiva e dinamica).

Inoltre, **BaklavaJS** è implementata in **TypeScript**, un linguaggio che garantisce una maggiore sicurezza dei tipi durante lo sviluppo, riducendo la possibilità di errori a livello di tipo.

Con **BaklavaJS**, è possibile creare editor di grafi personalizzati definendo nodi e le relazioni tra di essi. La libreria offre una flessibilità che la rende adatta a una vasta gamma di casi d'uso e mette a disposizione un sistema di plugin per integrare nuove funzionalità nell'editor di grafi, rendendola estensibile e adattabile alle esigenze specifiche del progetto. La versione 2 di **BaklavaJS**, attualmente in fase beta, introduce miglioramenti e funzionalità aggiuntive per semplificare ulteriormente lo sviluppo. Nonostante le sue potenzialità, **BaklavaJS** mantiene una ridotta occupazione di risorse, con una dimensione inferiore a 60 KB una volta compressa.

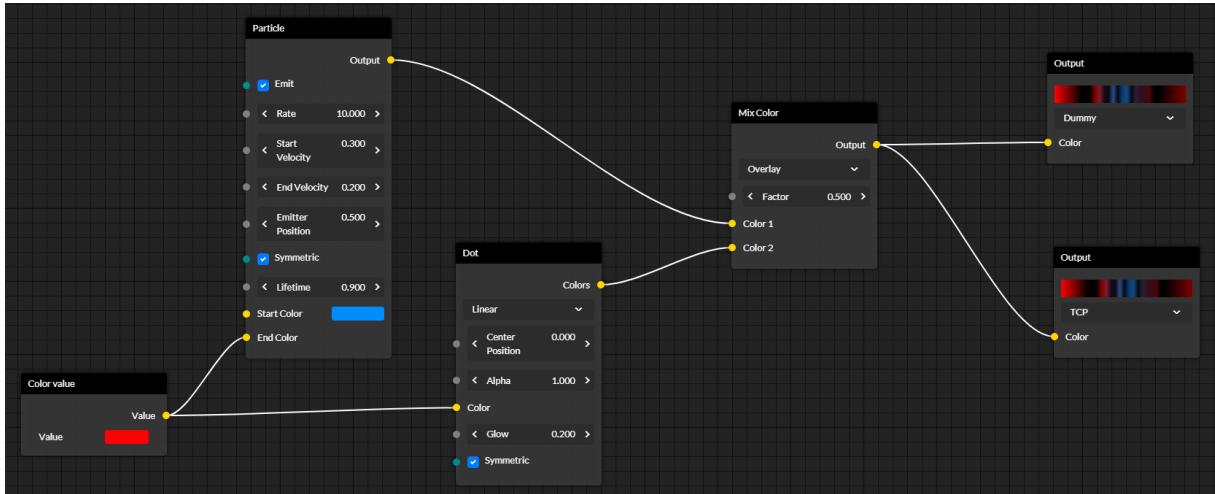


Figura 2.5: Esempio di grafo realizzato con BaklavaJS

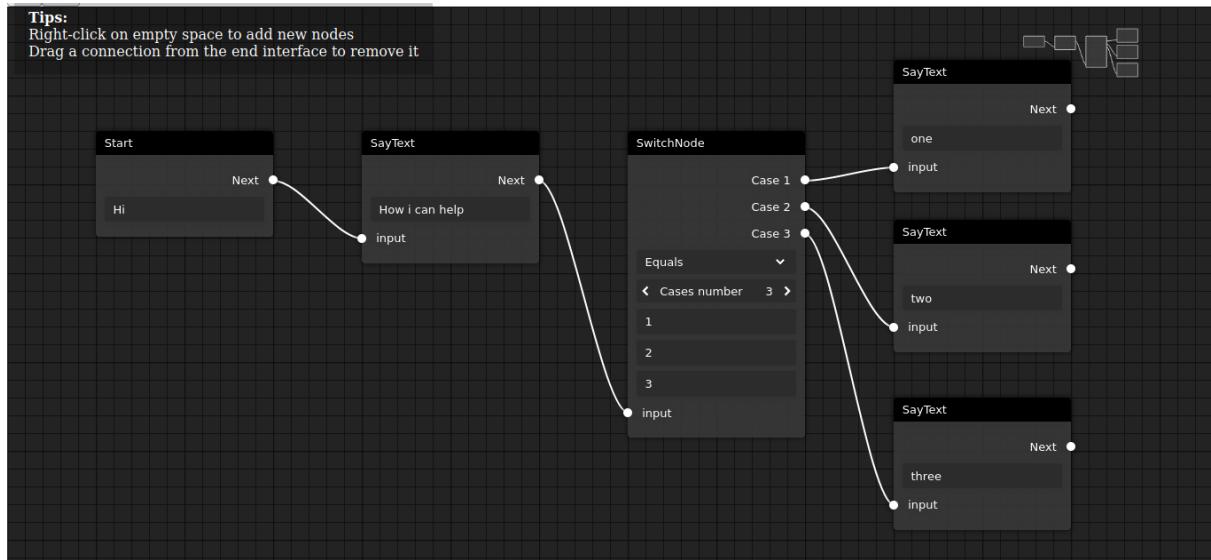


Figura 2.6: Esempio di grafo realizzato con BaklavaJS

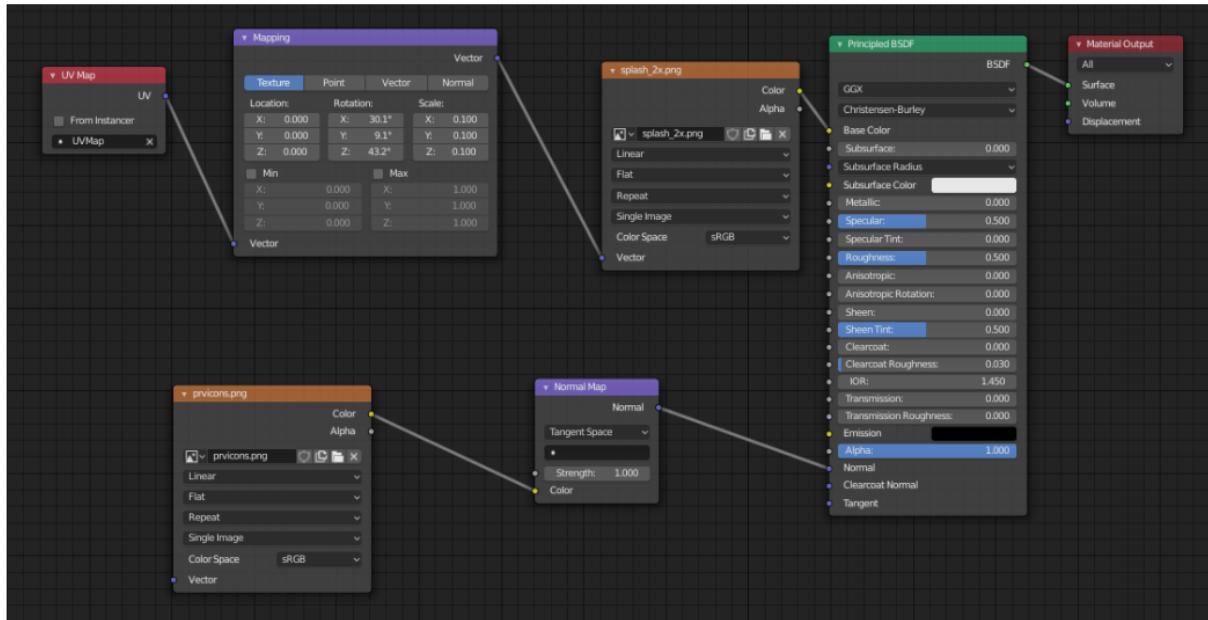


Figura 2.7: Esempio di grafo realizzato con BaklavaJS

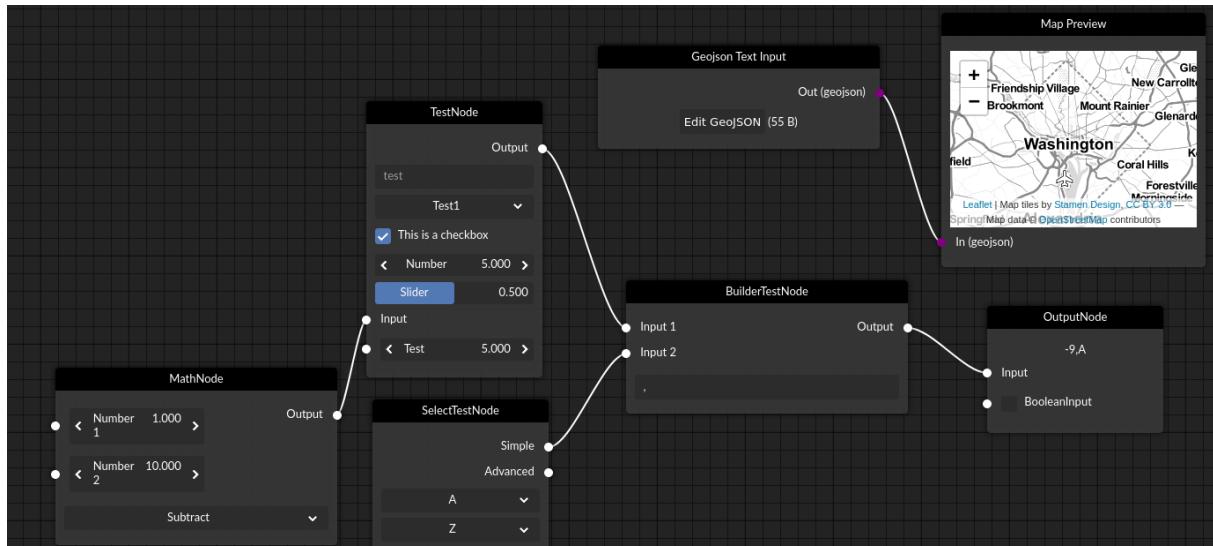


Figura 2.8: Esempio di grafo realizzato con BaklavaJS

2.2.9 JQuery

jQuery è una libreria JavaScript open source ampiamente utilizzata nella programmazione web. Fondata sulla filosofia di semplificare lo sviluppo web, jQuery offre una vasta gamma di funzionalità per manipolare il DOM, gestire eventi, creare animazioni e gestire richieste AJAX. Questa libreria è nota per la sua leggerezza e facilità d'uso, rendendo più accessibile la creazione di interazioni utente dinamiche e effetti visivi senza dover scrivere lunghe righe di codice JavaScript.

Uno dei punti di forza di **jQuery** è il suo potente sistema di selezione, che consente di individuare e modificare elementi HTML in base a criteri specifici. Grazie a questo sistema, è possibile applicare modifiche a interi gruppi di elementi in modo efficiente.

La gestione degli eventi è un'altra caratteristica chiave di **jQuery**, consentendo agli sviluppatori di assegnare azioni agli eventi HTML, come clic, hover e tastiera, in modo semplice ed elegante.

Inoltre, **jQuery** offre la possibilità di creare animazioni e transizioni su elementi HTML, migliorando l'aspetto e l'interattività delle pagine web. Le animazioni CSS possono essere applicate a una vasta gamma di proprietà, permettendo agli sviluppatori di creare effetti visivi sorprendenti.

La libreria è altamente compatibile con i browser più diffusi, garantendo che le applicazioni web funzionino in modo coerente su piattaforme diverse.

La documentazione dettagliata e la comunità di sviluppatori attiva sono risorse preziose per apprendere e risolvere problemi legati a **jQuery**.

2.2.10 AJAX

AJAX (Asynchronous JavaScript and XML), acronimo che rappresenta un significativo progresso nello sviluppo di applicazioni web, offre un'esperienza utente avanzata ed interattiva.

Ma come funziona esattamente?

Innanzitutto, vale la pena sottolineare che il nome di questa tecnica fornisce indizi significativi sui suoi componenti chiave: il linguaggio di programmazione **JavaScript** e il formato dati **XML** (anche se ormai **JSON** è più comune per la sua leggerezza).

La caratteristica principale di **AJAX** è la sua capacità di aggiornare il contenuto di una pagina web in modo asincrono, ovvero senza necessità di ricaricare completamente la pagina stessa. Ciò consente all'utente di interagire con la pagina in modo dinamico, senza dover attendere il ricaricamento completo a ogni modifica o aggiornamento. Questa flessibilità è resa possibile attraverso richieste asincrone al server web.

Ecco alcuni aspetti chiave di **AJAX**:

- **Richieste asincrone:** **AJAX** permette di inviare richieste asincrone al server web, senza dover ricaricare completamente la pagina. In questo modo, l'utente può interagire con la pagina senza dover attendere il caricamento completo.
- **Manipolazione diretta del DOM:** **AJAX** consente la manipolazione diretta del Document Object Model (DOM) della pagina web. Ciò significa che è possibile aggiornare specifiche parti della pagina, come elementi HTML, senza dover ricaricare tutto il contenuto.

- **Gestione degli errori:** AJAX offre un sistema di gestione degli errori, per evitare problemi di sicurezza e garantire un’esperienza utente fluida.
- **Librerie:** Esistono molte librerie JavaScript che semplificano l’utilizzo di AJAX, ad esempio **jQuery**, **AngularJS**, **React**, ecc.

Per utilizzare **AJAX**, è necessario conoscere il linguaggio **JavaScript** e le API per inviare richieste asincrone al server web. Ecco un esempio di codice per inviare una richiesta **AJAX** utilizzando la libreria **jQuery**:

```
1 $.ajax({  
2     url: '/api/data',  
3     type: 'GET',  
4     dataType: 'json',  
5     success: function(data) {  
6         // Manipolazione dei dati ricevuti  
7     },  
8     error: function(xhr, status, error) {  
9         // Gestione degli errori  
10    }  
11});
```

Figura 2.9: Esempio di codice di una richiesta AJAX

In questo esempio, la funzione `$.ajax()` di **jQuery** è utilizzata per inviare una richiesta GET al server web. Quando il server risponde con i dati, la funzione ‘success’ viene eseguita per manipolare i dati ricevuti. In caso di errore, la funzione ‘error’ gestisce la situazione. Quindi, **AJAX** è una tecnica di programmazione web che consente di migliorare l’esperienza utente online, rendendo le applicazioni web più dinamiche e reattive.

Grazie alle richieste asincrone e alla manipolazione del DOM, gli utenti possono interagire in modo più fluido con le pagine web. Le librerie **JavaScript** specializzate semplificano l’implementazione, consentendo agli sviluppatori di creare applicazioni web più sofisticate e interattive.

2.2.11 JSON

JSON, acronimo di ”**JavaScript Object Notation**,” è un formato di serializzazione basato su testo ampiamente utilizzato per lo scambio di dati tra applicazioni. La sua popolarità deriva dalla sua semplicità, flessibilità e versatilità.

La sintassi di JSON è basata su coppie chiave-valore e può rappresentare una vasta gamma di strutture dati. Oltre a oggetti e array, JSON può gestire tipi di dati primitivi come numeri, stringhe, booleani e valori ‘null’. Questa flessibilità consente di rappresentare in modo efficace dati complessi e strutturati, rendendo JSON adatto per la trasmissione e lo scambio di informazioni tra applicazioni eterogenee.

Un esempio di dato JSON potrebbe apparire così:

```
1 {
2   "nome": "John",
3   "cognome": "Doe",
4   "eta)": 30,
5   "email": "john.doe@example.com"
6 }
```

Figura 2.10: Esempio di codice JSON

JSON è un formato indipendente dal linguaggio di programmazione, il che significa che può essere utilizzato in una varietà di linguaggi come JavaScript, Python, Java, C#, e molti altri. Mentre questa caratteristica è comune a molti formati di dati, è particolarmente rilevante in quanto JSON offre una struttura leggibile e flessibile per lo scambio di informazioni tra applicazioni scritte in linguaggi diversi. Contrariamente ad alcuni formati, come XML o protobuf, JSON mantiene la sua leggibilità e versatilità in modo uniforme, indipendentemente dal linguaggio di programmazione utilizzato.

I file JSON utilizzano l'estensione ‘.json’ e sono comunemente utilizzati per memorizzare dati di configurazione, dati di persistenza e altri dati strutturati. JSON è anche ampiamente utilizzato nelle API RESTful e nei servizi web per trasmettere dati tra il client e il server.

2.3 Backend

2.3.1 Python

Python, un linguaggio di programmazione rinomato per la sua flessibilità e potenza, si distingue come un'importante risorsa in diversi ambiti. Dall'analisi dei dati alla creazione di sofisticate applicazioni web, dalla semplice automazione di compiti quotidiani alla gestione di complessi algoritmi di machine learning, Python si adatta perfettamente a una vasta gamma di necessità.

La sua sintassi chiara e leggibile rende Python accessibile a programmatore di tutti i livelli, facilitando la scrittura di codice pulito e facilmente comprensibile.

Uno dei punti di forza di Python è la sua ricca libreria standard, fornendo moduli e pacchetti preconfezionati per una varietà di compiti. Questa vasta raccolta semplifica il lavoro dei programmatore, offrendo soluzioni già pronte per l'uso in aree come l'analisi dati, l'elaborazione di immagini e lo sviluppo di interfacce grafiche.

La portabilità di Python è un altro vantaggio, essendo supportato su diverse piattaforme come Windows, macOS e Linux. Questa caratteristica garantisce che il codice Python possa essere eseguito su una gamma diversificata di sistemi senza richiedere modifiche sostanziali.

L'ecosistema versatile di Python si riflette nella sua vasta gamma di librerie e framework. Questa flessibilità consente ai programmatore di personalizzare soluzioni per una varietà di contesti, con librerie di terze parti che ampliano ulteriormente le opzioni disponibili.

Python è un il linguaggio principale utilizzato nell'ambito dell'elaborazione dati e dell'analisi statistica, grazie a strumenti come NumPy e Pandas, semplificando operazioni di manipolazione e analisi dei dati. Nel campo del machine learning e dell'intelligenza artificiale, Python brilla con framework avanzati come TensorFlow e PyTorch, che consentono lo sviluppo di modelli sofisticati per una vasta gamma di applicazioni.

Json

La libreria JSON di Python è una componente integrante delle funzionalità standard di Python ed è progettata per semplificare la gestione di dati nel formato JSON. JSON è un formato di dati comunemente utilizzato per rappresentare informazioni in maniera leggibile sia per gli esseri umani che per le applicazioni software, ed è ampiamente adottato per lo scambio di dati tra varie applicazioni.

Sys

La libreria sys di Python costituisce una risorsa fondamentale che offre una serie di funzioni e variabili per ottenere informazioni dettagliate sul sistema operativo e la configurazione dell'interprete Python.

In breve, la libreria `sys` fornisce gli strumenti necessari per acquisire una comprensione approfondita di ciò che avviene all'interno del programma Python.

Importlib

La libreria Python `importlib` è uno dei moduli della Standard Library di Python. Questo modulo fornisce un'interfaccia per il caricamento di moduli in modo dinamico durante l'esecuzione del programma. Ciò significa che è possibile importare moduli in modo dinamico, ovvero senza doverli importare esplicitamente all'inizio del programma.

Questa flessibilità è particolarmente utile in situazioni in cui non si conosce il nome del modulo in anticipo ma è necessario determinarlo in base a criteri logici durante l'esecuzione del programma. Questo significa che è possibile importare moduli all'interno del codice sorgente a tempo di esecuzione, utilizzando funzioni come `import_module` o il metodo `import`.

Tempfile

La libreria `tempfile` in Python fornisce funzionalità per la creazione e la gestione di file temporanei e directory temporanee. Questi file e directory sono utili per operazioni che richiedono dati temporanei e non necessitano di memorizzazione permanente. Con `tempfile`, è possibile creare file temporanei in modo sicuro, evitando conflitti di nomi e garantendo la pulizia automatica dei file temporanei una volta terminato il programma. Inoltre, la libreria offre diverse opzioni per la gestione dei file temporanei, come la creazione di file con nomi personalizzati o la scelta del percorso di salvataggio.

Shutil

`Shutil` è un modulo Python che offre una serie di strumenti per la gestione di operazioni di file system, come copia, spostamento e rimozione di file e directory.

Ast

Il modulo `ast` di Python si configura come una buona risorsa per l'analisi e la manipolazione di codice Python. Offrendo funzionalità avanzate di parsing e trasformazione del codice in alberi sintattici astratti, `ast` fornisce strumenti potenti per l'analisi statica del codice e la creazione di strumenti di analisi automatica

Pandas

Pandas è una libreria open source per la manipolazione e l'analisi di dati in Python. È stata sviluppata per fornire un'ampia gamma di strumenti per la gestione di dati strutturati e non strutturati, come ad esempio dati tabulari, serie temporali e dati di tipo matriciale.

La libreria è basata su **NumPy**, un'altra libreria Python per la manipolazione di array numerici, e offre una vasta gamma di funzionalità per la pulizia, la trasformazione e l'analisi dei dati. Il fulcro di **Pandas** è rappresentato dal DataFrame, una struttura dati simile a una tabella con righe e colonne. Questo può essere alimentato con dati da diverse fonti, tra cui file CSV, fogli di calcolo Excel o database. Una volta creato, il DataFrame può essere sottoposto a una serie di operazioni come la pulizia, la trasformazione e la manipolazione attraverso un'ampia gamma di funzioni e metodi integrati.

Pandas si distingue per la gestione dei dati mancanti, offrendo diverse opzioni per trattare i valori nulli, come la compilazione, la rimozione o la sostituzione con valori specifici.

Inoltre, questa libreria vanta un sofisticato sistema di indicizzazione, che agevola l'accesso a dati specifici sia per righe che per colonne.

Oltre alla manipolazione dei dati, **Pandas** offre un ricco set di funzioni per l'analisi e la visualizzazione dei dati. Queste includono il calcolo di statistiche di riepilogo, l'aggregazione dei dati e la creazione di tabelle pivot. **Pandas** è in grado di fondere, unire e rimodellare i dati in modo efficiente, offrendo versatilità nelle operazioni di analisi dati.

Grazie alla sua facilità d'uso, alla sintassi intuitiva, **Pandas** è ampiamente utilizzato in vari settori, tra cui la finanza, la scienza dei dati, la ricerca accademica e l'analisi dei social media. La libreria è supportata da una documentazione completa e da una serie di tutorial, rendendo l'apprendimento e l'utilizzo di **Pandas** accessibili a tutti gli utenti. Può essere installato agevolmente tramite pip o come parte della distribuzione **Anaconda**, una piattaforma di analisi dati multi-piattaforma.

Flask

Flask è un micro-framework open source scritto in Python, noto per la sua leggerezza, flessibilità e semplicità d'uso. Si basa sulla libreria Web Server Gateway Interface (WSGI) Werkzeug, che fornisce funzionalità essenziali per la gestione di richieste e risposte web. Questo framework è ampiamente utilizzato per lo sviluppo di applicazioni web, grazie alla sua capacità di gestire richieste HTTP, creare pagine HTML dinamiche e supportare funzionalità come la gestione delle sessioni utente.

Flask offre alcune caratteristiche chiave:

- **Routing:** Flask consente di definire le rotte dell'applicazione, consentendo agli utenti di accedere a diverse parti dell'applicazione tramite URL.

- **Templates:** Utilizza il motore di template Jinja2 per generare pagine HTML dinamiche, consentendo la separazione della logica dell'applicazione dalla presentazione dei dati.

`Flask` offre anche funzionalità aggiuntive tramite estensioni:

- **Forms:** Sebbene `Flask` non gestisca direttamente i form HTML, fornisce estensioni e moduli che semplificano la gestione dei form, inclusa la validazione dei dati inseriti dagli utenti, garantendo la sicurezza e l'integrità dei dati.
- **Database:** `Flask` offre un'interfaccia per interagire con vari tipi di database, consentendo di inserire, modificare o eliminare dati.
- **Altre Estensioni:** `Flask` dispone di numerose estensioni che consentono di aggiungere funzionalità aggiuntive all'applicazione, come l'autenticazione degli utenti, l'invio di email e la gestione dei file statici.

Un'applicazione `Flask` può essere creata facilmente, con la possibilità di definire rotte e funzioni eseguite quando un utente accede a un'URL specifico. Ad esempio, è possibile creare una semplice homepage che restituisce un messaggio di saluto quando l'utente accede all'URL principale.

`Flask` offre inoltre funzionalità avanzate, tra cui la gestione delle sessioni utente, la gestione degli errori e la gestione delle richieste AJAX. Questo rende `Flask` un'opzione ideale per lo sviluppo di applicazioni web, dalla più semplice alla più complessa.

Capitolo 3

Descrizione soluzione

In questo capitolo, verrà esplorata la soluzione implementativa e concettuale del progetto, fornendo dettagli approfonditi sugli algoritmi, le tecniche implementative e le tecnologie utilizzate.

Sarà dedicato spazio alla discussione delle sfide affrontate durante lo sviluppo, illustrando le metodologie adottate e presentando dettagliatamente le soluzioni implementate.

Concluderà con una guida pratica sull'utilizzo dell'applicazione, pensata per essere comprensibile sia agli utenti finali che agli sviluppatori impegnati nella manutenzione futura. L'obiettivo principale è rendere chiara e agevole la comprensione dell'applicazione, evidenziando l'aspetto pratico e operativo per massimizzare la sua utilità in diversi contesti.

3.1 Introduzione

Il fulcro della soluzione implementata risiede nella capacità di assegnare categorie specifiche alle righe di una tabella, basandosi su condizioni definite attraverso le colonne.

Questa funzionalità è stata sviluppata al fine di automatizzare il processo di categorizzazione dei dati, con l'obiettivo di ridurre l'onere manuale e aumentare l'efficienza nell'analisi dei dati.

L'applicazione esamina attentamente le condizioni specificate per le colonne della tabella e, attraverso un processo dinamico, assegna a ciascuna riga una categoria appropriata. Questo approccio permette di evidenziare le caratteristiche e i pattern nascosti nei dati, semplificando il processo di categorizzazione e consentendo agli utenti di ottenere informazioni rilevanti ed esaustive dai dati sottostanti.

Per realizzare questa funzionalità, l'applicazione utilizza un approccio fondato su grafì di regole, in cui le condizioni vengono applicate alle colonne e le connessioni tra tali condizioni e le righe delineano il modo in cui ciascuna riga interagisce con tali condizioni.

Questa struttura grafica intelligente crea un legame visuale diretto tra le condizioni applicate alle colonne della tabella e le categorie assegnate a ciascuna riga. L’automazione di questo processo elimina la necessità di effettuare manualmente la categorizzazione, riducendo così la possibilità di errori umani e garantendo uniformità nelle assegnazioni. L’importanza di questa soluzione risiede nella semplificazione del complesso processo di categorizzazione dei dati. Questo strumento non solo aumenta l’efficienza dell’analisi dati, ma offre anche una rappresentazione chiara delle caratteristiche dei dati, aiutando gli utenti a individuare tendenze, comportamenti e relazioni all’interno delle informazioni analizzate.

Nel contesto dell’analisi dati, questa capacità di identificare automaticamente le categorie contribuisce in modo significativo all’efficacia dell’interpretazione dei risultati e alla formulazione di strategie basate su informazioni concrete.

3.2 Scelte implementative

3.2.1 Interfaccia

Per la realizzazione dell’applicazione, è stata utilizzata una libreria open source trovata su GitHub, chiamata `BaklavaJS`. Tra le numerose risorse considerate, questa libreria si è distinta come eccellente opzione. Tale decisione è stata motivata non solo dalla sua presenza in diverse alternative, ma soprattutto dalla completezza del suo codice sorgente, reso completamente disponibile. Questo aspetto ha facilitato un’analisi dettagliata del codice, consentendo una migliore comprensione e, quando necessario, una personalizzazione mirata alle esigenze del progetto. Ulteriori meriti vanno alla flessibilità delle interfacce dei nodi, che si sono dimostrate altamente configurabili e intuitive.

Tali considerazioni hanno giocato un ruolo determinante nella selezione di `BaklavaJS`, poiché le caratteristiche richieste per l’applicazione trovavano riscontro in modo completo ed efficiente in questa libreria. La sua adattabilità alle specifiche esigenze del progetto, unite alla sua solidità e alla chiarezza delle sue interfacce, hanno reso questa libreria la scelta ideale per la realizzazione del sistema.

In aggiunta, sono state integrate librerie di supporto quali `IziToast` e `SweetAlert` al fine di migliorare l’usabilità dell’applicazione sviluppata. Un’attenzione particolare è stata riservata all’esperienza utente, evidenziando la volontà di rendere l’interazione con il sistema quanto più intuitiva e gradevole possibile. Questa scelta riflette l’impegno nel fornire agli utenti uno strumento capace di soddisfare pienamente le loro esigenze, sia nella fase di creazione dei risultati desiderati, sia nell’eventualità di errori.

Grazie a tali librerie, è stato possibile implementare notifiche e avvisi visivi che, in modo chiaro e immediato, forniscono indicazioni utili agli utenti, facilitando la comprensione degli errori eventualmente verificatisi e indicando possibili soluzioni.

3.2.2 Flask

L'adozione di **Flask** come framework back-end per l'applicazione è stata motivata da considerazioni fondamentali e strategiche che hanno richiesto un'analisi approfondita. Rispetto a soluzioni alternative come **Django**, la scelta di **Flask** è stata dettata da considerazioni pratiche legate alla sua struttura modulare e alla sua agilità nello sviluppo. **Flask** si distingue per la sua struttura snella e modulare, che consente un deployment più agevole rispetto a framework più complessi come **Django**. Questo aspetto risulta particolarmente vantaggioso in scenari in cui è necessario massimizzare le risorse e mantenere il sistema snello e reattivo.

Per quanto riguarda l'elaborazione dei dati e l'applicazione di filtri alle tabelle, l'impiego di **Python** emerge come scelta naturale e vantaggiosa. **Python**, grazie alla sua sintassi chiara e alla ricchezza delle sue librerie, offre strumenti potenti e versatili per la manipolazione dei dati. In particolare, l'utilizzo di librerie come **Pandas** consente di gestire in modo efficiente operazioni complesse su grandi volumi di dati tabellari.

Questo approccio consente di alleggerire il carico di lavoro del browser e di offrire prestazioni più elevate, specialmente in presenza di dataset di grandi dimensioni o di operazioni computazionalmente intensive.

3.3 Architettura del software

Il software si configura come una complessa composizione di diverse componenti, ognuna delle quali svolge un ruolo cruciale nell'ecosistema applicativo.

Inizialmente, si procede con la creazione di un'applicazione web utilizzando **HTML**, **CSS** e **JavaScript**, la quale verrà eseguita localmente attraverso l'ausilio di **Flask**. Questa scelta tecnologica consente una gestione flessibile e dinamica dell'interfaccia utente, garantendo un'esperienza fluida e intuitiva per gli utenti finali. **Flask**, in particolare, funge da ponte tra il front-end e il back-end dell'applicazione, facilitando la gestione efficiente delle richieste e delle risposte.

Successivamente, dopo la fase di avvio dell'applicazione web e la creazione del grafo di riferimento, il processo procede con la fase di controllo ed esecuzione delle condizioni, che determinano il comportamento del sistema. Durante questa fase, all'utente viene data la possibilità di selezionare il dataset su cui basare l'analisi, tramite l'interfaccia dell'applicazione web.

Una volta effettuate tali scelte, i dati vengono trasformati in formato JSON e inviati tramite richiesta AJAX a un'entità specifica, il cui ruolo verrà delineato in seguito.

Va sottolineato che Python assume un ruolo predominante all'interno del sistema, occupandosi sia della gestione del grafo, interpretato nel formato JSON, sia del processo di elaborazione delle informazioni. Grazie alla sua versatilità e potenza computazionale, Python consente una gestione efficiente e scalabile dei dati, permettendo l'implementazione di algoritmi complessi e procedure di filtraggio.

In dettaglio, una volta che Python ha ricevuto il grafo e ha elaborato le informazioni in base alle condizioni specificate dall'utente, restituisce la risposta desiderata. Tale risposta si presenta sotto forma di un dataframe, il quale rappresenta la tabella che sarà visualizzata nella pagina web. È importante notare che per ogni nodo del grafo viene associata una tabella separata, filtrata in base alle condizioni specificate, garantendo un'analisi dettagliata e granulare dei dati.

3.3.1 Classi principali BaklavaJS

BaklavaJS offre diverse classi fondamentali che sono ampiamente utilizzate nel progetto. Ogni classe è costituita da metodi (funzioni che definiscono il comportamento della classe), proprietà (caratteristiche o dati associati alla classe) e accessori (metodi per accedere o modificare le proprietà della classe):

- **Editor:** La classe Editor in BaklavaJS rappresenta un insieme fondamentale di nodi e connessioni all'interno del sistema. Questa classe agisce come un'interfaccia centrale per manipolare e gestire la struttura complessiva del grafo. Inoltre, offre funzionalità per il caricamento e il salvataggio dei dati del grafo, semplificando il processo di lavoro su progetti complessi.

Methods:

 `addGraphTemplate`  `load`  `registerGraph`  `registerNodeType`
 `removeGraphTemplate`  `save`  `unregisterGraph`  `unregisterNodeType`

Properties:

 `connectionEvents`  `events`  `graphEvents`  `graphHooks`
 `graphTemplateEvents`  `graphTemplateHooks`  `hooks`  `nodeEvents`
 `nodeHooks`

Accessors:

 `graph`  `graphTemplates`  `graphs`  `loading`  `nodeTypes`

- **Node:** Un nodo è l'elemento centrale che mette a disposizione Baklavajs. Nel contesto di un grafo, i nodi rappresentano gli elementi fondamentali, insieme alle connessioni tra di essi. Ogni nodo può essere considerato come un'istanza di una classe, la cui definizione è necessaria prima di crearne uno. Gli attributi di un nodo includono il tipo, che è unico e utilizzato per salvare e caricare grafici, e il titolo, che è opzionale e può essere utilizzato come identificativo predefinito se non specificato diversamente. Altri attributi opzionali comprendono gli input e gli output, che specificano le interfacce di input e output del nodo, e una funzione di calcolo che viene eseguita ogni volta che il nodo viene eseguito. Maggiori dettagli sull'esecuzione del nodo sono disponibili nella pagina dedicata.

Methods:

M load **M** onDestroy **M** onPlaced **M** registerGraph **M** save

Accessors:

A graph **A** title

Properties:

P calculate? **P** events **P** hooks **P** id **P** inputs **P** outputs
P type

- **NodeInterface:** Le interfacce dei nodi definiscono gli input e gli output di un nodo e possono essere collegate tra loro. Ogni interfaccia di nodo può avere un port, utilizzato per collegare un'interfaccia di output di un nodo a un'interfaccia di input di un altro nodo. Di default, i port sono visualizzati, ma in determinati scenari potrebbero non essere necessari. Ad esempio, un'interfaccia di output può essere utilizzata solo per visualizzare dati, mentre un'interfaccia di input potrebbe essere destinata solo all'input diretto dell'utente, senza aspettarsi dati da altri nodi. In tali casi, il port può essere disattivato durante la creazione dell'interfaccia del nodo.

Methods:

M load **M** save **M** setComponent **M** setHidden **M** setPort
M use

Properties:

P component? **P** events **P** hidden **P** hooks **P** id **P** isInput?
P name **P** nodeId **P** port **P** templateId?

Accessors:

A connectionCount **A** value

- **Connections:** Le connessioni sono fondamentali per facilitare la comunicazione tra le diverse interfacce di nodo all'interno del grafo. Queste connessioni consentono il passaggio di dati da un'interfaccia di output di un nodo a un'interfaccia di input di un altro nodo. Grazie alle connessioni, le interfacce di nodo possono scambiarsi informazioni e lavorare in sinergia per l'elaborazione dei dati. La gestione delle connessioni offre flessibilità nel design del grafo, consentendo agli utenti di creare e rimuovere collegamenti secondo le proprie esigenze. Il flusso di dati attraverso le connessioni è essenziale per il corretto funzionamento del grafo, permettendo un processo fluido di elaborazione e trasmissione dei dati tra i vari nodi.

Methods:

 **destruct**

Properties:

 **destructed**  **events**  **from**  **id**  **to**

- **Events:** Ci sono due tipi di eventi: eventi normali e eventi prevenibili. Gli eventi normali vengono solitamente generati dopo un'azione, per reagire a quell'azione. Gli eventi prevenibili, d'altra parte, vengono generati prima di un'azione e possono essere utilizzati dal listener per impedire che l'azione si verifichi. Ciò viene fatto chiamando la funzione `prevent` in una funzione di listener. La maggior parte degli eventi prevenibili ha il prefisso `before` nel loro nome. Gli eventi forniscono un meccanismo essenziale per la comunicazione e la gestione delle azioni all'interno di BaklavaJS, consentendo una maggiore flessibilità e controllo nel flusso del programma.

3.3.2 Comportamento Python

Dopo aver ricevuto il grafo in formato JSON, il codice Python utilizza diverse librerie per eseguire una serie di operazioni volte alla trasformazione e all'analisi dei dati.

La libreria `json` viene utilizzata per caricare il file JSON contenente il grafo. Questo file JSON contiene informazioni dettagliate sui nodi del grafo, tra cui `id`, `nome`, `tipo` e `opzioni`. Successivamente, la libreria `pandas` viene coinvolta per elaborare i dati tabellari. I dati vengono caricati in un `dataframe`, una struttura dati potente e flessibile che consente di eseguire operazioni complesse su grandi volumi di dati.

La libreria `ast` gioca un ruolo chiave nell'analisi delle condizioni definite per ciascun nodo del grafo. Questa libreria consente di analizzare le espressioni Python all'interno delle condizioni e di risolvere le funzioni definite al loro interno.

Per importare funzioni personalizzate utilizzate nel processo di analisi, viene utilizzata la libreria `importlib`. Questo consente al programma di utilizzare funzioni definite in un file esterno, offrendo maggiore flessibilità e modularità al sistema.

Infine, la libreria `csv` viene coinvolta per esportare i risultati dell'analisi in formato CSV.

3.3.3 Comunicazione Python - Javascript

La comunicazione tra JavaScript e Python avviene tramite richieste HTTP, con il frontend che invia richieste al backend e riceve risposte. Questo scambio di dati permette al frontend di interagire con il backend in modo dinamico, consentendo all'utente di eseguire azioni e visualizzare risultati in tempo reale.

Nel backend, le richieste vengono gestite da endpoint definiti dall'applicazione, ognuno dei quali esegue operazioni specifiche quando viene contattato. Queste operazioni possono includere l'elaborazione di dati, l'esecuzione di algoritmi o il recupero di informazioni da un database.

Una volta eseguite le operazioni nel backend, i risultati vengono solitamente formattati in JSON e inviati come risposta alla richiesta originale del frontend. Il frontend riceve quindi i dati JSON e li utilizza per aggiornare l'interfaccia utente, visualizzando i risultati o eseguendo azioni appropriate.

Questo modello di comunicazione client-server consente un'interazione fluida tra frontend e backend, consentendo agli sviluppatori di creare applicazioni web complesse e interattive.

3.4 Utilizzo applicazione

Nella seguente sezione, verrà illustrato il processo di utilizzo dell'applicazione con l'ausilio di dati campione. Si tratteranno dettagliatamente le scelte metodologiche adottate e la loro applicazione pratica.

3.4.1 Avvio applicazione

Di seguito si espone il procedimento per avviare l'applicazione nell'ambito della presente tesi:

Listing 3.1: Avvio applicazione

```
1  #!/bin/bash
2  python app.py file_function1.py ... file_functionN.py
```

Qui di seguito viene fornito un esempio di file contenente del codice Python, il quale rappresenta funzioni personalizzate utili durante l'impiego dell'applicazione:

Listing 3.2: Esempio di file Python con funzioni

```
1 # functions_file.py
2 def raddoppia(x):
3     return x * 2
4
5 def aggiungi10(y):
6     return y + 10
```

Per illustrare questa procedura, immaginiamo di utilizzare un set di dati di piccole dimensioni che riguarda monete. Il dataset è strutturato con i seguenti campi:

- **Nome:** Il nome della moneta.
- **Paese:** Il paese d'origine della moneta.
- **Valuta:** La valuta associata alla moneta.
- **Anno:** L'anno di emissione della moneta.
- **Prezzo (€):** Il valore in euro della moneta.
- **Rarità:** Un indicatore di rarità della moneta.
- **Peso (g):** Il peso della moneta in grammi.
- **Diametro (mm):** Il diametro della moneta in millimetri.
- **Materiale:** Il materiale con cui è stata coniata la moneta.
- **Conservazione:** Stato di conservazione della moneta

Questo dataset, focalizzato sulle monete, costituirà l'esempio di riferimento per comprendere il funzionamento dell'applicazione e la sua capacità di gestire dati eterogenei attraverso la definizione di regole specifiche.

Il file "app.py" contiene l'applicazione **Flask** che avvia un sito web locale. Questa app richiede zero o più argomenti, ovvero il nome di uno o più file **Python** che conterranno le funzioni necessarie che eventualmente possono essere utilizzate per definire le condizioni. L'applicazione **Flask** gestisce le richieste dell'utente tramite un'interfaccia web, consentendo di interagire con la funzione implementata in **Python**.

Questo file è cruciale per il progetto, poiché abilita l'applicazione web a utilizzare le funzionalità Python. L'applicazione si avvia in locale (localhost), permettendo all'utente di accedervi tramite un browser.

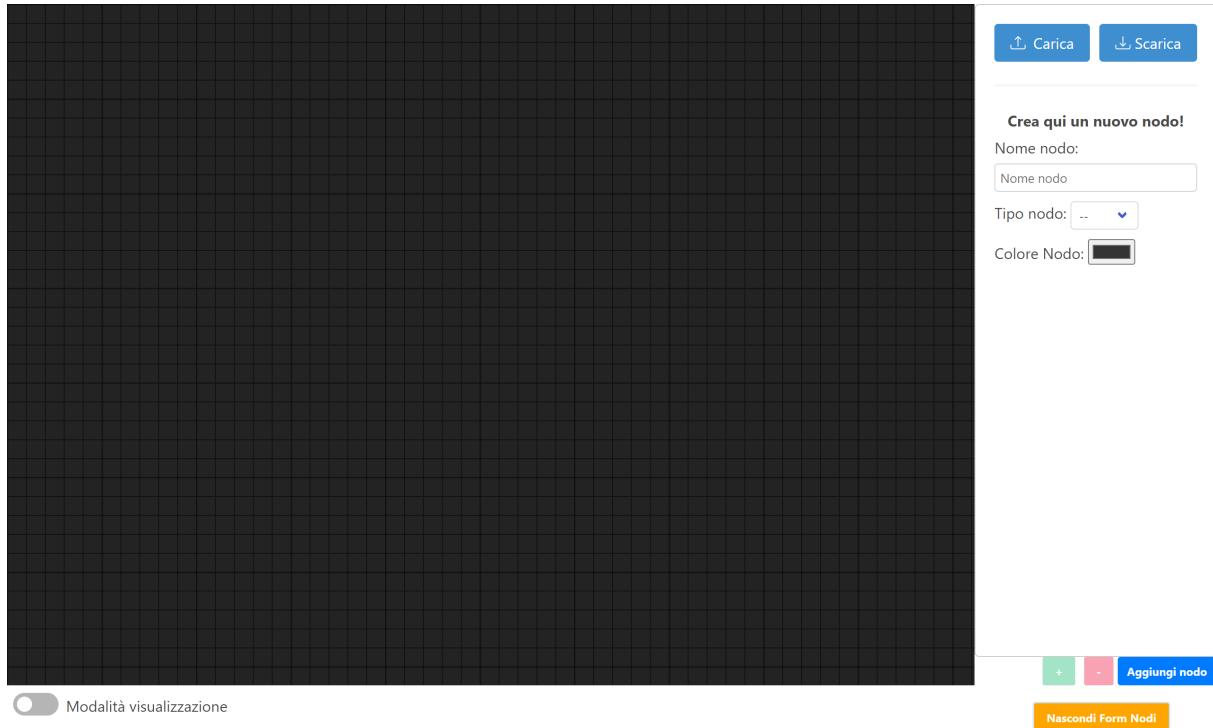


Figura 3.1: Schermata iniziale

All'avvio dell'applicazione, si presenta la schermata iniziale, illustrata nella Figura 3.1. Posizionata a sinistra dell'interfaccia, emerge l'editor, una componente cardine dell'apparato applicativo, fornito tramite l'impiego della libreria JavaScript BaklavaJS. Questa strumentazione agevola l'utente nell'atto di generare, disporre e modificare i nodi costituenti il nucleo delle regole di classificazione.

La rappresentazione visiva dei nodi facilita l'organizzazione e la comprensione delle regole, contribuendo a rendere il processo di classificazione più trasparente e accessibile. In opposizione, si colloca sulla destra dell'interfaccia un modulo interattivo, che si presta alla creazione di nodi personalizzati. La creazione di un nodo nuovo attraverso tale modulo si traduce agevolmente nell'ulteriore possibile inserimento all'interno dell'editor, consentendo successive manipolazioni ed elaborazioni.

3.4.2 Creazione di un nodo

Per la creazione del grafo, è stato implementato un sistema che consente la generazione dei nodi a partire da tre categorie principali: il ”*nodo di partenza*” o ”*start*”, che presenta solo connessioni in uscita e ne può essere aggiunto uno solo nell’editor; i ”*nodi intermedi*”, che dispongono sia di connessioni in ingresso che in uscita; e i ”*nodi terminali*” o ”*foglia*”, che presentano solamente connessioni in ingresso.

La tipologia di nodo può essere selezionata tramite la sezione ”*Tipo nodo*” all’interno del modulo, e in base alla scelta effettuata, vengono mostrati ulteriori campi da compilare. Nel caso di un nodo ”*foglia*”, non vengono visualizzati ulteriori parametri, poiché l’unico dato richiesto è il nome del nodo. Questo nome non ha alcuna influenza sul funzionamento del programma, ma serve esclusivamente a scopi di identificazione. È importante notare che i nomi dei nodi devono essere univoci, e se voluto, è possibile selezionare un colore personalizzato per il nodo, altrimenti verrà utilizzato il colore predefinito.

Crea qui un nuovo nodo!

Nome nodo:

Nome nodo

Tipo nodo: Foglia

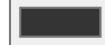
Colore Nodo: 

Figura 3.2: Selezione nodo ”Foglia”

La procedura varia a seconda della selezione tra ”*nodo di partenza*” e ”*nodo intermedio*”. Anche se è possibile aggiungere solo un ”*nodo di partenza*” nell’editor, è importante notare che è consentito crearne un numero illimitato. Una volta effettuata la scelta tra questi due tipi di nodi, verrà richiesta un’ulteriore specifica, ovvero il tipo di output desiderato.

Crea qui un nuovo nodo!

Nome nodo:

Tipo nodo: Start



Colore Nodo:

Tipo output: --



Figura 3.3: Selezione nodo "Start"

Crea qui un nuovo nodo!

Nome nodo:

Tipo nodo: Nodo



Colore Nodo:

Tipo output: --



Figura 3.4: Selezione nodo "Nodo" (nodo intermedio)

All'interno dell'applicazione, sono disponibili tre tipi di output tra cui è possibile scegliere: *"Vero/Falso"*, *"Categorico"* e *"Range"*. Nel caso dell'opzione *"Vero/Falso"*, si configura un nodo in cui gli output ammissibili sono esclusivamente *"vero"* o *"falso"*. All'interno di questo nodo, l'utente è tenuto a definire una condizione booleana in modo inequivocabile. Tale condizione può comprendere sia funzioni provenienti dal file di input, caricato al momento dell'avvio del programma, che nomi di colonne presenti in un dataset specifico. La condizione definita determina il percorso all'interno del grafo. È inoltre possibile combinare queste opzioni in una singola condizione, consentendo una maggiore flessibilità nella definizione delle regole.

Listing 3.3: Esempio condizione booleana

```
1 paese == "Italia" or raddoppia(peso) > 100 and anno < 2000
```

Attraverso l'opzione *"Categorico"*, l'utente ha la possibilità di selezionare output personalizzati, fornendo una notevole flessibilità nella definizione dei risultati desiderati. All'interno del nodo associato a questa opzione, a differenza di altre modalità, non è richiesta la definizione di una condizione specifica da soddisfare. Invece, l'aspetto chiave consiste nell'identificare e specificare il nome di una colonna all'interno del dataset. Il percorso all'interno del grafo di processo sarà quindi determinato in base al valore presente in quella colonna. Questo approccio consente una personalizzazione avanzata in cui l'utente può indirizzare il flusso di lavoro in base a variabili specifiche. Un elemento importante da notare è l'opzione denominata 'Altro' che può essere abilitata tramite un toggle. Questa opzione agisce come un meccanismo di gestione dei casi che non corrispondono a output specifici precedentemente definiti. In altre parole, funge da una sorta di *'else'* nel processo decisionale.

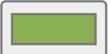
Questa caratteristica consente di gestire situazioni non previste o eccezionali in modo efficace, garantendo che il flusso di lavoro possa essere completato anche in presenza di dati inattesi.

Crea qui un nuovo nodo!

Nome nodo:

Test

Tipo nodo:

Colore Nodo: 

Tipo output:

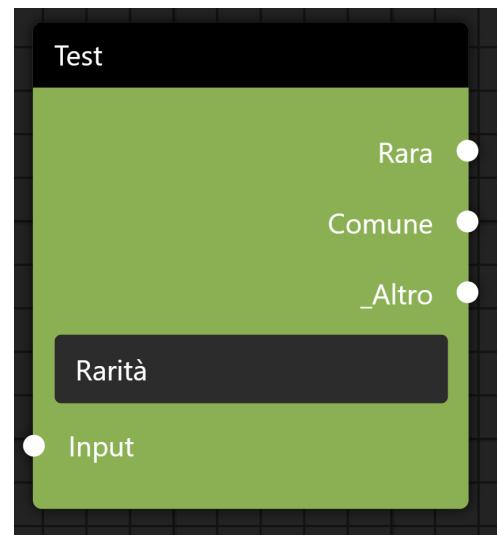
Valori categorici di output:

Rara

Comune

Altro

(a) Selezione nodo categorico



(b) Nodo categorico risultante

Figura 3.5: Nodo categorico

L'ultima opzione, *”Range”*, rappresenta una modalità specifica che consente di gestire dati numerici all'interno del processo decisionale. In questa configurazione, l'utente è tenuto a fornire il nome di una colonna contenente valori numerici all'interno del dataset. Questo approccio offre notevoli opportunità per eseguire valutazioni basate su dati quantitativi. Un elemento chiave dell'opzione *”Range”* è la possibilità di definire intervalli numerici. L'utente ha il controllo completo per stabilire questi intervalli, consentendo una personalizzazione avanzata. Ad esempio, è possibile creare intervalli specifici come '0-10', '11-20', '21-30', ecc. In questo modo, i dati numerici possono essere suddivisi in categorie significative.

Crea qui un nuovo nodo!

Nome nodo:

Test

Tipo nodo:

Colore Nodo:

Tipo output:

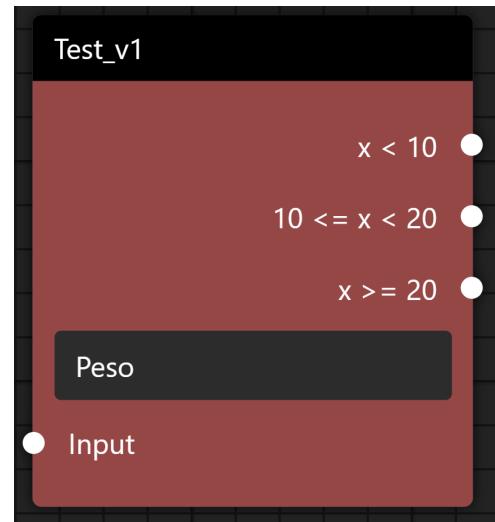
Valori categorici di output:

10

20

Altro

(a) Selezione nodo range



(b) Nodo range risultante

Figura 3.6: Nodo Range

Inoltre, nell'opzione 'Range', è prevista un'ulteriore opzione di configurazione che consente di specificare se gli intervalli sono aperti da sinistra o da destra. Questa scelta determina l'inclusione o l'esclusione dei limiti degli intervalli nei calcoli. Ad esempio, se si opta per un intervallo aperto da sinistra, il valore più basso sarà incluso nell'intervallo, mentre il valore più alto sarà escluso. Questa caratteristica offre una maggiore precisione nell'assegnazione dei dati numerici a intervalli specifici.

Crea qui un nuovo nodo!

Nome nodo:

Test_v2

Tipo nodo: Nodo

Colore Nodo:

Tipo output: Range

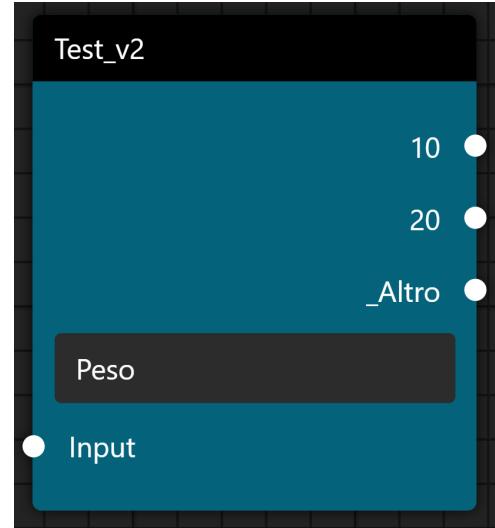
Valori range:

10

20

V2

(a) Selezione nodo range



(b) Nodo range risultante

Figura 3.7: Nodo Range

L’implementazione di queste diverse opzioni all’interno dell’applicazione offre agli utenti un’ampia flessibilità nella definizione delle condizioni e nella gestione dei dati. Questo consente agli utenti di creare grafi di regole altamente personalizzati per condurre analisi dati dettagliate e mirate.

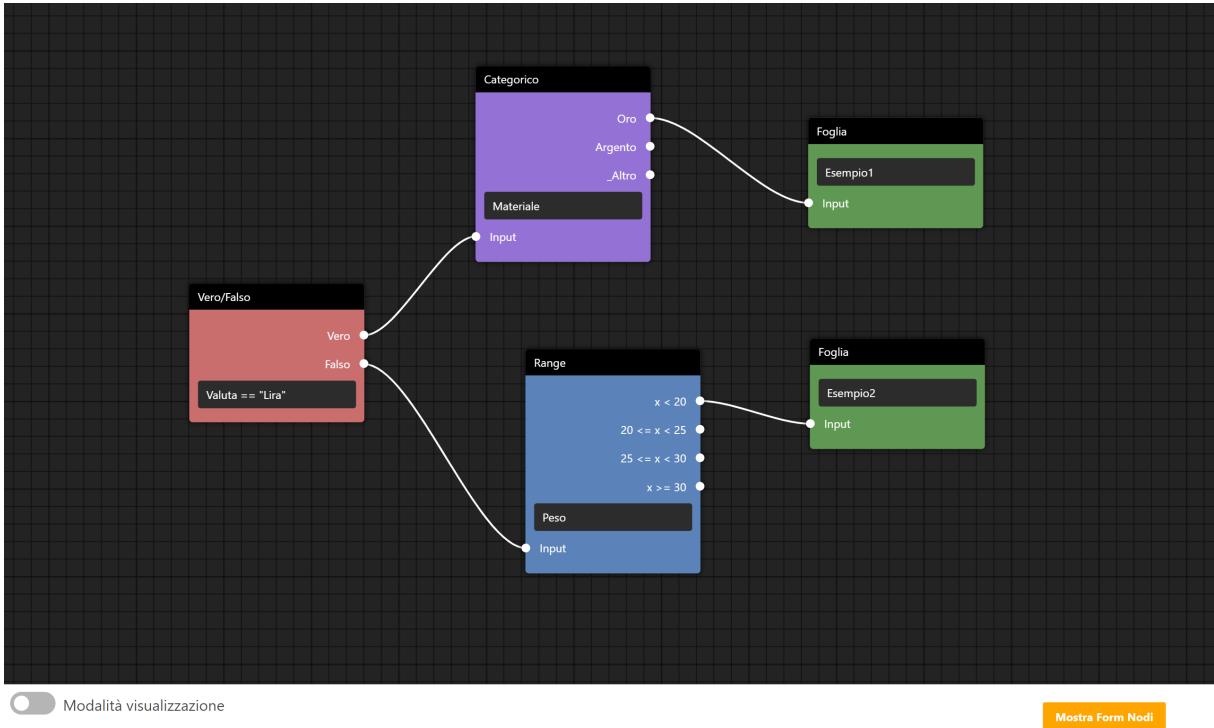


Figura 3.8: Esempio creazione nodi

Nella figura (3.8) è stato illustrato un esempio di grafo contenente tutti i diversi tipi di output.

3.5 Creazione delle regole

Dopo aver creato diversi nodi, il processo di creazione del grafo decisionale inizia con l'inserimento di tali nodi nell'editor. La disposizione dei nodi all'interno dell'editor è fondamentale e sarà guidata dalle condizioni che vengono definito all'interno di ciascun nodo. Questo processo di progettazione è influenzato anche dal comportamento delle righe della tabella, che risponderanno alle condizioni impostate nel grafo.

È importante precisare che ciascun nodo deve essere correttamente connesso. Ciò significa che non possono esserci nodi senza connessioni input, cioè nodi che non ricevono informazioni da un nodo precedente, oppure nodi con uno o più output non collegati a un nodo successivo. Questa connessione è fondamentale per garantire che il flusso decisionale sia ininterrotto.

In aggiunta, è fondamentale che ogni nodo contenga una condizione chiaramente definita. La mancanza di una definizione precisa della condizione all'interno di un nodo non solo può causare errori durante l'esecuzione del programma, ma può persino impedire l'avvio del programma stesso. Ci sono anche alcune regole da seguire durante la creazione del grafo.

Per esempio, non è consentito aggiungere più di un nodo ”*Start*” all’interno dell’editor. Inoltre, non è possibile creare nodi con lo stesso nome, ma è possibile avere nodi con lo stesso nome presenti nell’editor, il che può essere utile in determinati contesti.

Un altro aspetto cruciale riguarda la presenza di cicli nel grafo. Il grafo decisionale non deve contenere situazioni cicliche, poiché quando il processo verrà eseguito, è necessario arrivare a un nodo ”*Foglia*” per determinare quale percorso seguire in base alle condizioni. La presenza di cicli potrebbe creare ambiguità e interrompere il flusso decisionale.

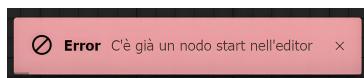


Figura 3.9: Alert errore nodo start



Figura 3.10: Alert errore cicli



Figura 3.11: Alert errore nodo con nome esistente

Per garantire l’assenza di cicli, viene applicato un controllo mediante l’algoritmo di Depth-First Search (DFS). Questo algoritmo esamina le connessioni nel grafo e identifica eventuali cicli, consentendo così di mantenere l’integrità e la coerenza del flusso decisionale.

Una volta che queste condizioni sono rispettate e si è soddisfatti della struttura del grafo appena creato, è possibile iniziare ad eseguire lo script e passare alla modalità di visualizzazione. Durante l’esecuzione, il grafo prenderà decisioni basate sulle condizioni che sono definite, guidando il processo in base ai dati in ingresso e fornendo risultati significativi per ciascuna riga della tabella o situazione specifica.

Inoltre, attraverso il pulsante ”scarica”, gli utenti avranno la possibilità di esportare il grafo in un file JSON. Questo file seguirà un formato specifico, garantendo che le informazioni sul grafo siano organizzate in modo strutturato e facilmente leggibile per scopi futuri o condivisione con altri.

3.5.1 Caricare un Grafo

Per ulteriore flessibilità, è possibile importare un grafo direttamente utilizzando il pulsante ”Carica” situato nella parte superiore della form. Questa funzione richiede l’upload di un file in formato JSON che segua uno specifico standard. In generale, il sistema prevede che il file JSON in input sia stato precedentemente scaricato dall’applicazione stessa, data la complessità della struttura. Qualsiasi tentativo di caricare un file che non sia in formato JSON o che non rispetti il formato previsto dall’applicazione comporterà la segnalazione di un errore.

Algorithm 1 Algoritmo per rilevare cicli

```
1: function HASCYCLES(connections)
2:   Status  $\leftarrow$  {NEW : 'NEW', SEEN : 'SEEN', DONE : 'DONE'}
3:   status  $\leftarrow$  {}
4:   graph  $\leftarrow$  new Map()
5:   for each conn in connections do
6:     fromNodeId  $\leftarrow$  conn.to.isInput ? conn.to.parent.id : conn.from.parent.id
7:     toNodeId  $\leftarrow$  conn.to.isInput ? conn.to.parent.id : conn.from.parent.id
8:     if  $\neg$ graph.has(fromNodeId) then
9:       graph.set(fromNodeId, [])
10:    end if
11:    if  $\neg$ graph.has(toNodeId) then
12:      graph.set(toNodeId, [])
13:    end if
14:    graph.get(fromNodeId)  $\leftarrow$  graph.get(fromNodeId) + [toNodeId]
15:  end for
16:  function DFS(u)
17:    status[u]  $\leftarrow$  Status.SEEN
18:    for each v in graph.get(u) do
19:      if status[v] == Status.SEEN then
20:        return true                                 $\triangleright$  Ciclo trovato
21:      end if
22:      if status[v] == Status.NEW and DFS(v) then
23:        return true                                 $\triangleright$  Ciclo trovato in un sottoalbero
24:      end if
25:    end for
26:    status[u]  $\leftarrow$  Status.DONE
27:    return false
28:  end function
29:  for each u in graph.keys() do
30:    status[u]  $\leftarrow$  Status.NEW
31:  end for
32:  for each u in graph.keys() do
33:    if status[u] == Status.NEW and DFS(u) then
34:      return true                                 $\triangleright$  Ciclo trovato
35:    end if
36:  end for
37:  return false                                 $\triangleright$  Nessun ciclo trovato
38: end function
```

3.5.2 Formato JSON

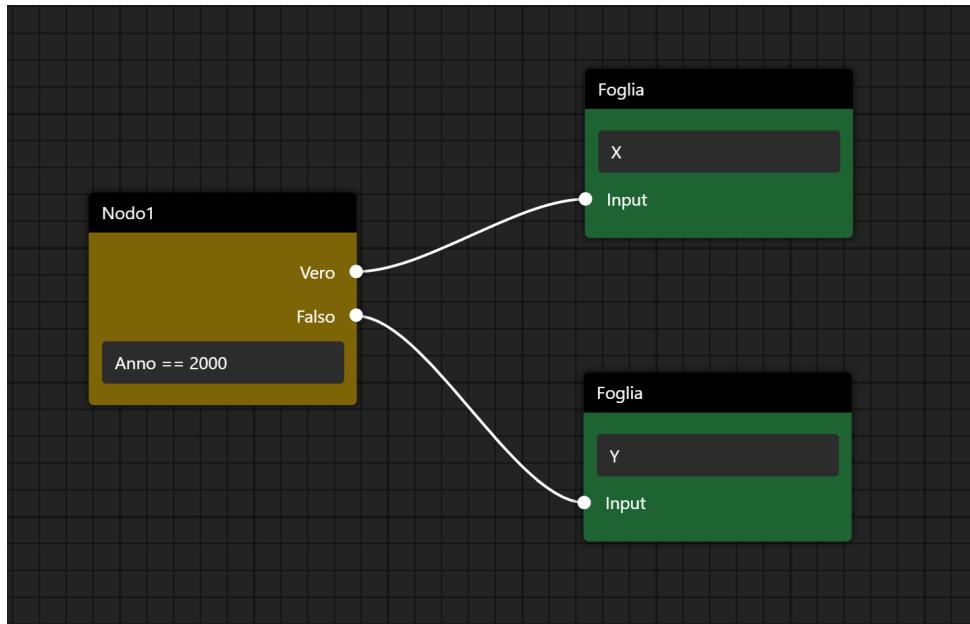


Figura 3.12: Esempio grafo

Listing 3.4: Codice JSON generato

```
1 {
2   "nodes": [
3     {
4       "type": "Nodo1",
5       "id": "node_17122341417480",
6       "name": "Nodo1",
7       "options": [
8         "Condizione", "Anno == 2000"
9       ],
10      "state": {},
11      "interfaces": [
12        ["Vero", {
13          "id": "ni_17122341417481",
14          "value": null
15        }],
16        ["Falso", {
17          "id": "ni_17122341417482",
18          "value": null
19        }]
20      ],
21    },
22    {
23      "type": "Foglia",
24      "id": "foglia_x",
25      "name": "Foglia X",
26      "options": [],
27      "state": {},
28      "interfaces": [
29        {"id": "input_x", "label": "Input", "type": "Input", "value": null}
30      ]
31    },
32    {
33      "type": "Foglia",
34      "id": "foglia_y",
35      "name": "Foglia Y",
36      "options": [],
37      "state": {},
38      "interfaces": [
39        {"id": "input_y", "label": "Input", "type": "Input", "value": null}
40      ]
41    }
42  ]
```

```
21     "position": {
22         "x": 77,
23         "y": 159
24     },
25     "width": 200,
26     "twoColumn": false,
27     "customClasses": ""
28 },
29 {
30     "type": "Foglia",
31     "id": "node_17122341754313",
32     "name": "Foglia",
33     "options": [
34         ["Valore", "X"]
35     ],
36     "state": {},
37     "interfaces": [
38         {"Input", {
39             "id": "ni_17122341754314",
40             "value": null
41         }}
42     ],
43     "position": {
44         "x": 447,
45         "y": 67
46     },
47     "width": 200,
48     "twoColumn": false,
49     "customClasses": ""
50 },
51 {
52     "type": "Foglia",
53     "id": "node_17122341783335",
54     "name": "Foglia",
55     "options": [
56         ["Valore", "Y"]
57     ],
58     "state": {},
59     "interfaces": [
```

```
60      ["Input", {
61        "id": "ni_17122341783336",
62        "value": null
63      }]
64    ],
65    "position": {
66      "x": 446,
67      "y": 293
68    },
69    "width": 200,
70    "twoColumn": false,
71    "customClasses": ""
72  }
73 ],
74 "connections": [
75   {
76     "id": "17122341888649",
77     "from": "ni_17122341417481",
78     "to": "ni_17122341754314"
79   },
80   {
81     "id": "171223419236013",
82     "from": "ni_17122341417482",
83     "to": "ni_17122341783336"
84   }
85 ],
86 "panning": {
87   "x": 315.2400000418017,
88   "y": 45.88125550212162
89 },
90 "scaling": 0.9939734484629387,
91 "colorMap": {
92   "--type-Nodo1": "#7d6406",
93   "--type-Foglia": "#1e6432"
94 }
95 }
```

Nella Figura 3.12, possiamo osservare un esempio di codice JSON che definisce un grafo. Questo codice è utilizzato per rappresentare una struttura dati complessa che consiste in nodi e connessioni, spesso usati per modellare relazioni o strutture complesse in un'applicazione.

Il codice JSON è suddiviso in due parti principali: "nodes" (nodi) e "connections" (connessioni).

1. Nodi (nodes):

- Ogni nodo è un oggetto JSON contenuto nell'array "nodes".
- "type" rappresenta il tipo di nodo, ad esempio, "Nodo1" o "Foglia". Questo campo definisce il ruolo e le caratteristiche specifiche di ciascun nodo.
- "id" è un identificatore unico assegnato a ciascun nodo.
- "name" è il nome del nodo, spesso utilizzato per scopi di riferimento o visualizzazione.
- "options" definisce le opzioni specifiche del nodo. Ad esempio, in "Nodo1", l'opzione "Condizione" ha il valore "Anno == 2000".
- "state" è un oggetto vuoto, che potrebbe essere utilizzato per memorizzare lo stato o le informazioni aggiuntive del nodo.
- "interfaces" rappresenta le interfacce del nodo. Ogni interfaccia ha un nome e un oggetto associato. Ad esempio, "Vero" e "Falso" sono interfacce di "Nodo1".
- "position" definisce la posizione del nodo nel grafo, utilizzando le coordinate (x, y).
- "width" rappresenta la larghezza del nodo.
- "twoColumn" è un valore booleano che potrebbe essere utilizzato per specificare la struttura a due colonne del nodo.
- "customClasses" potrebbe essere utilizzato per applicare classi CSS personalizzate al nodo.

2. Connessioni (connections):

- Le connessioni tra i nodi sono definite nell'array "connections".
- Ciascuna connessione ha un "from" e un "to", che rappresentano le interfacce di partenza e destinazione. Ad esempio, la connessione "17122341888649," va dall'interfaccia "ni_17122341417481" a "ni_17122341417482".

Oltre a nodi e connessioni, il codice JSON include altre informazioni:

- "panning" definisce la posizione di visualizzazione del grafo nel documento.
- "scaling" specifica la scala del grafo.

3. "colorMap": definisce una mappatura dei colori basata sui tipi di nodi.

Questo tipo di codice JSON viene spesso utilizzato per modellare e rappresentare grafi di dati all'interno di applicazioni software, consentendo di visualizzare relazioni complesse tra entità o dati.

3.6 Modalità visualizzazione

Dopo aver creato un grafo che soddisfa le condizioni desiderate, è possibile passare alla modalità di visualizzazione utilizzando il toogle. Ma cosa comporta esattamente questa modalità? In pratica, una volta attivata questa modalità, la prima richiesta è il file CSV a cui si desiderano applicare le regole appena create. Questa flessibilità consente di adattare il processo decisionale a diverse fonti di dati, contribuendo a rendere l'applicazione più versatile e personalizzabile.

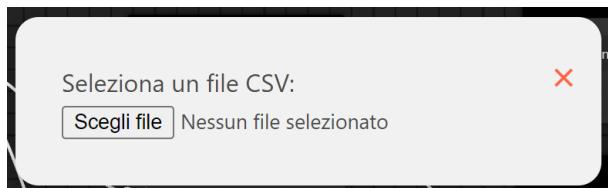


Figura 3.13: Richiesta file CSV

Appena inserito il file parte l'esecuzione di uno script Python, viene creato una copia del file CSV originale utilizzando le funzionalità della libreria Python `tempfile` e `shutil`. Questa copia del file viene utilizzata come base per l'esecuzione delle operazioni successive, garantendo che il file originale rimanga inalterato e che eventuali modifiche siano applicate solo alla copia temporanea. Questo approccio assicura l'integrità e la sicurezza dei dati originali mentre si effettuano le elaborazioni necessarie durante l'analisi del grafo. Una volta inserito il file CSV e atteso il completamento del caricamento, si entra nella modalità di visualizzazione.

Durante il caricamento, il processo sottostante è realizzato utilizzando il framework `Flask`. In questa fase, il framework analizza il JSON generato dal grafo e procede ad applicarlo al file CSV fornito. L'intera operazione viene eseguita attraverso una chiamata `AJAX`, consentendo una comunicazione asincrona tra il client e il server. Tale approccio ottimizza l'esperienza utente, permettendo al processo di svolgersi senza dover ricaricare completamente la pagina web.

CAPITOLO 3. DESCRIZIONE SOLUZIONE

Possono verificarsi due scenari possibili.

Il primo scenario è rappresentato da eventuali incongruenze tra le regole definite e il contenuto del file CSV. Queste incongruenze possono riguardare in particolare i nomi delle colonne o dai tipi degli attributi, i quali possono essere definiti nelle regole ma risultare assenti nel file CSV. Allo stesso modo, potrebbero emergere discordanze riguardo a funzioni utilizzate che non sono state definite nel file delle funzioni fornito inizialmente.

In caso di tale situazione, il processo genererà un errore, ma questo sarà gestito attraverso un segnalatore che evidenzierà in rosso i nodi problematici. Questa evidenziazione lampeggiante segnalerà gli errori nel processo AJAX, consentendo all'utente di identificare facilmente quali nodi stanno impedendo l'esecuzione corretta.

Facendo riferimento al dataset delle monete utilizzato finora, supponiamo di inserire come condizione in un certo nodo "`Nome == 'Andrea'`", considerando che il dataset non contenga l'attributo "Nome". Vediamo come si comporta l'applicazione:

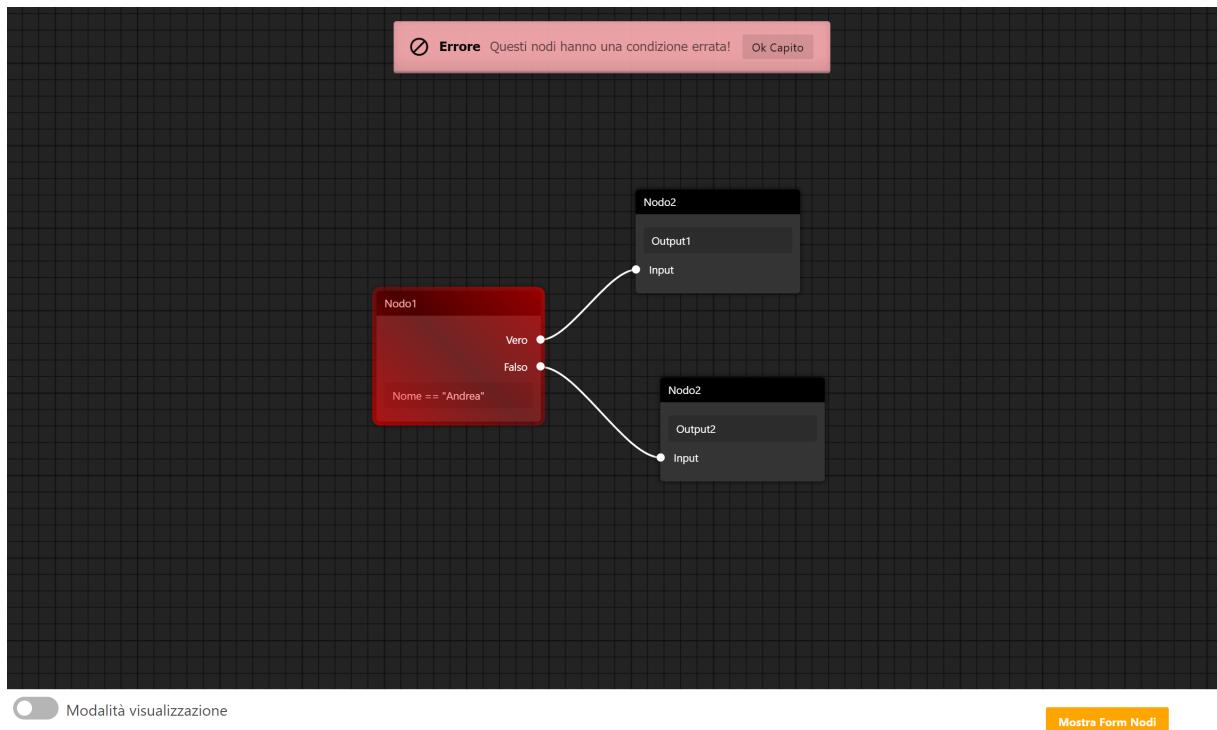


Figura 3.14: Nodo con condizione problematica

Si può notare come il nodo con questa condizione risulti problematico, e il colore rosso lampeggerà su tutti i nodi problematici fino a quando non viene premuto il pulsante "Ok, capito" sull'alert visualizzato. In questo modo, l'editor torna a essere modificabile.

Il secondo scenario, invece, si riferisce alla situazione in cui tutte le condizioni vengono valutate senza problemi durante l'esecuzione della chiamata **AJAX** sul file **CSV**. In questo contesto, dopo il completamento del processo, appare una breve notifica che segnala il passaggio nella modalità di visualizzazione. Durante tale modalità, l'editor non è suscettibile di modifiche; pertanto, non è possibile aggiungere o rimuovere nodi, connessioni, né modificare le condizioni all'interno dei nodi.

In questa modalità, è possibile interagire con i vari nodi per valutare il loro comportamento in relazione alle condizioni che essi contengono.

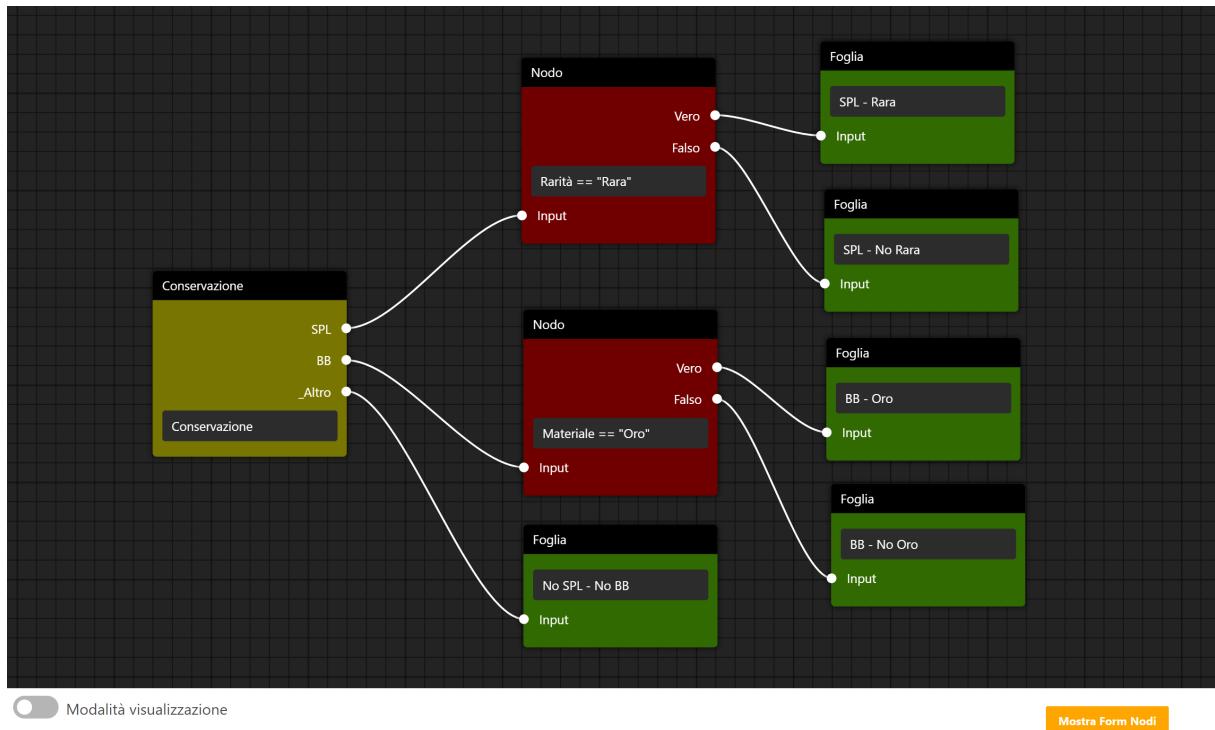
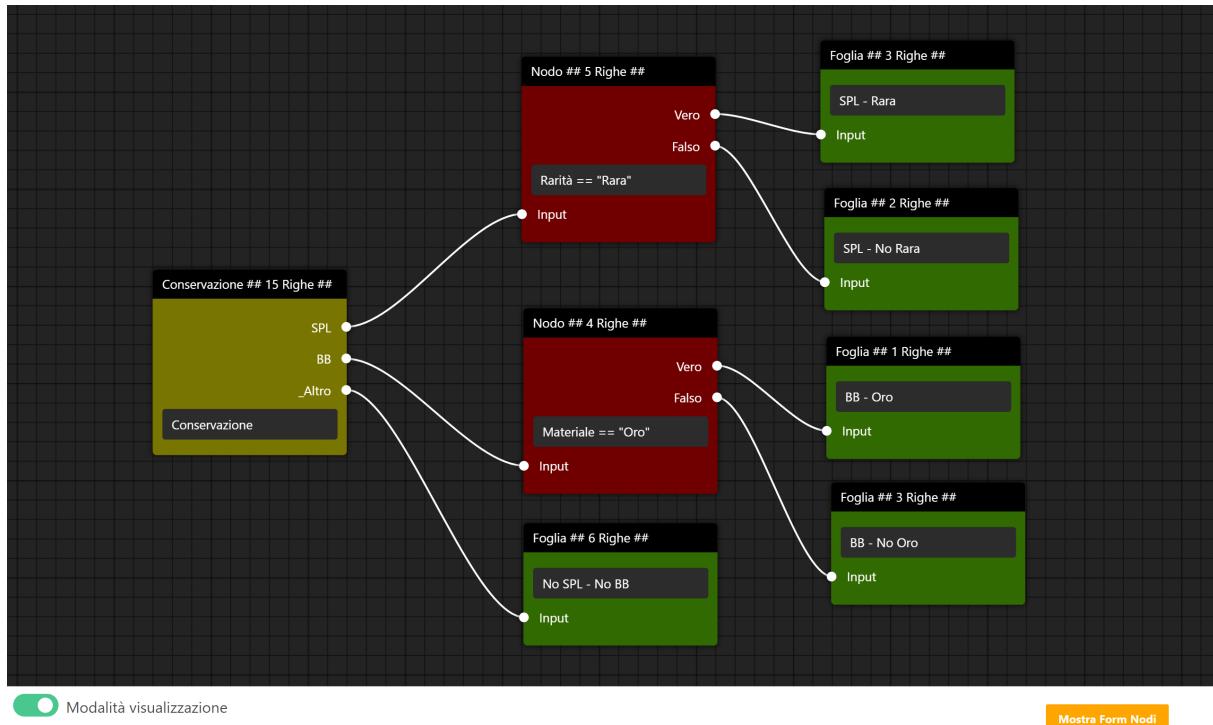


Figura 3.15: Esempio funzionante completo

Nella Figura 3.15, è possibile osservare un grafo completo con diverse condizioni, il quale, durante la transizione alla modalità di visualizzazione, non genera errori. Dopo aver attivato questa modalità, annunciata da un messaggio di successo, è evidente come nei nodi sia comparsa l'indicazione del numero di righe che il nodo, con la sua condizione e quelle dei nodi precedenti applicate alla tabella, genera.

CAPITOLO 3. DESCRIZIONE SOLUZIONE



Modalità visualizzazione

[Mostra Form Nodi](#)

Figura 3.16: Grafo in modalità visualizzazione

In questa modalità, è possibile selezionare un nodo per visualizzare la tabella risultante in base alla condizione specifica del nodo. Naturalmente, selezionando il nodo di "Start", si avrà accesso alla visualizzazione del dataset completo:

	Nome	Paese	Valuta	Anno	Prezzo	Rarità	Peso	Diametro	Materiale	Conservazione	Risultato
0	Aureus di Augusto	Roma	Denarius	14	5000	Rara	7.80	20.0	Oro	SPL	SPL - Rara
1	Tetradramma di Alessandro Magno	Macedonia	Dracma	-323	10000	Raro	17.20	25.0	Argento	BB	BB - No Oro
2	Dinaro di Harun al-Rashid	Baghdad	Dinaro	786	2000	Rara2	2.90	22.0	Oro	MB	No SPL - No BB
3	Mezzo scudo di Ferdinando II	Napoli	Scudo	1836	150	Comune	13.50	31.0	Argento	B	No SPL - No BB
4	1/2 Krugerrand	Sudafrica	Rand	1980	500	Comune	15.50	27.0	Oro	SPL	SPL - No Rara
5	1/2 Corona	Austria	Corona	1915	50	Comune	1.20	19.0	Oro	MB	No SPL - No BB
6	1/2 Dollaro	Stati Uniti	Dollaro	1964	10	Rara	11.34	30.6	Argento	SPL	SPL - Rara
7	1/2 Rublo	Russia	Rublo	1898	100	Rara2	1.60	17.0	Argento	MB	No SPL - No BB
8	1/2 Sovrano	Regno Unito	Sterlina	1911	100	Comune	3.99	19.3	Oro	BB	BB - Oro
9	1/2 Yuan	Cina	Yuan	1991	10	Rara	4.50	22.0	Oro	SPL	SPL - Rara
10	1/2 Zecchino	Venezia	Ducato	1750	200	Raro	3.50	21.0	Oro	MB	No SPL - No BB
11	1/2 Dinaro	Jugoslavia	Dinaro	1970	5	Rara	2.50	18.0	Argento	BB	BB - No Oro
12	1/2 Corona	Cecoslovacchia	Corona	1932	20	Comune	2.50	23.0	Argento	SPL	SPL - No Rara
13	1/2 Corona	Ungheria	Corona	1915	50	Comune	1.20	19.0	Oro	MB	No SPL - No BB
14	1/2 Corona	Romania	Corona	1930	10	Non Comune	2.50	23.0	Argento	BB	BB - No Oro

Modalità visualizzazione

[Mostra Form Nodi](#)

Figura 3.17: Nodo Start selezionato

CAPITOLO 3. DESCRIZIONE SOLUZIONE

Al contempo, selezionando qualsiasi nodo foglia, è possibile filtrare le righe in base al risultato indicato dal nodo:

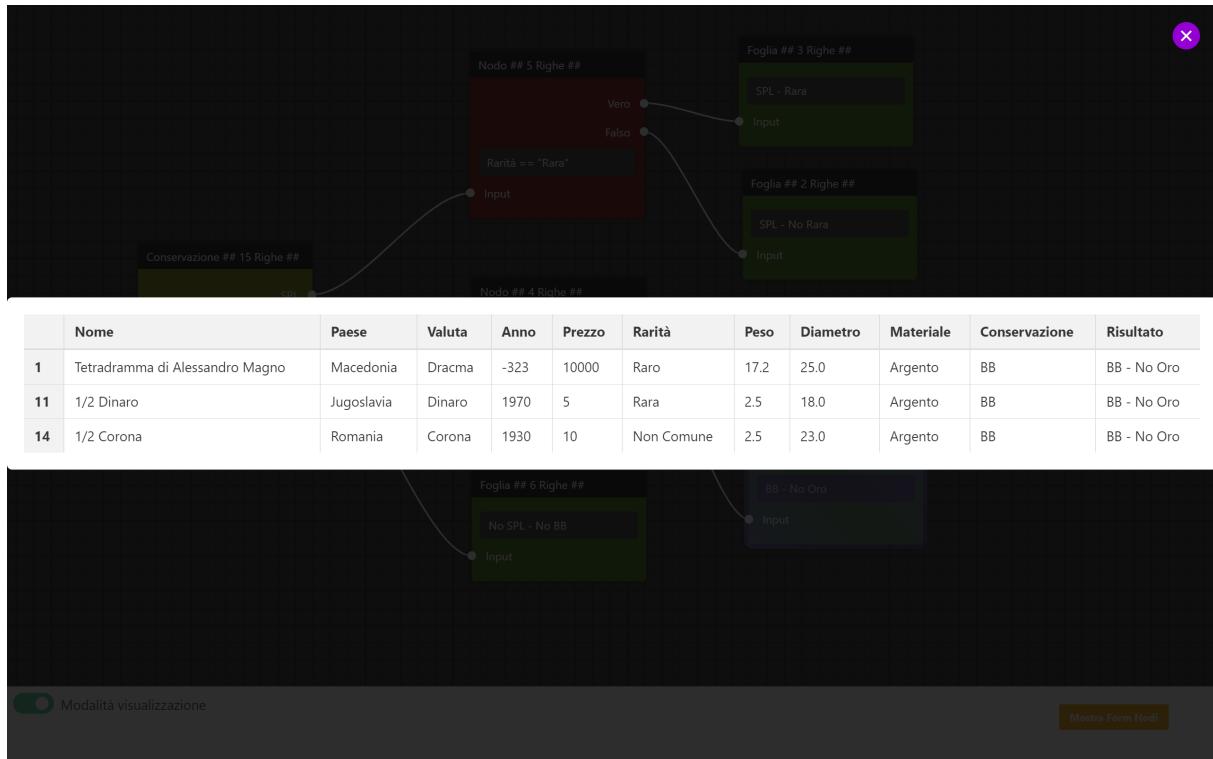


Figura 3.18: Nodo Foglia selezionato

Selezionando un qualsiasi nodo centrale, la tabella sarà filtrata in base alla condizione del nodo, con l'operatore logico **AND** applicato a tutte le condizioni precedenti.

Dopo aver verificato che ciascun nodo si comporta correttamente e essere soddisfatti dell'output ottenuto, è possibile scaricare il grafo nel formato **JSON**.

3.7 Applicazione delle regole

Questa applicazione poggia su una serie di script fondamentali che costituiscono la sua base. Questi script sono principalmente dedicati al controllo dell'applicazione delle regole e risultano cruciali per il funzionamento complessivo dell'applicazione stessa.

3.7.1 Script controllo condizioni

Nel procedimento delineato in precedenza, prima di applicare le regole stabilite, vengono effettuati dei controlli sulle varie condizioni descritte nei nodi del grafo. Questi sono finalizzati a verificare la coerenza con il dataset adottato come base delle regole stesse. Tale precauzione è necessaria poiché l'applicazione delle condizioni al dataset in questione potrebbe generare degli errori in fase di compilazione, nel momento in cui Python tenta di eseguirle. Un ulteriore controllo eseguito riguarda la coerenza e l'accuratezza delle funzioni impiegate nelle suddette condizioni. Di conseguenza, ho deciso di realizzare uno script specificamente dedicato alla valutazione di tali regole.

Per attuare questa verifica, ho sviluppato uno script Python che accetta espressioni Python come input e le valuta dinamicamente rispetto ai dati contenuti nel dataset. Questo processo permette di applicare regole definite come espressioni senza la necessità di modificare direttamente il file sorgente ogni volta che si desidera valutare un nuovo insieme di dati.

Per quanto riguarda la gestione delle funzioni utilizzate nelle espressioni è importante dire che queste funzioni possono essere definite in moduli esterni, il che offre una flessibilità significativa. Utilizzando l'analisi sintattica fornita dal modulo `ast`, lo script individua le chiamate di funzione all'interno delle espressioni dei nodi e cerca le definizioni di queste funzioni nei moduli importati. Questo approccio permette l'utilizzo di funzioni personalizzate senza doverle definire direttamente nel codice principale.

3.7.2 Script applicazione delle regole

È essenziale comprendere il processo di analisi del grafo in relazione alle regole definite. Come spiegato in precedenza, le regole definite all'interno del grafo vengono combinate logicamente mediante l'operatore `AND`.

Ciò significa che tutte le condizioni specificate nei nodi genitori devono essere soddisfatte affinché la condizione associata al nodo figlio possa essere valutata come vera. L'implementazione di questa logica è realizzata attraverso l'utilizzo della funzione `eval` di Python, la quale permette l'interpretazione e l'esecuzione di espressioni Python dinamicamente. Nella fase di valutazione delle condizioni dei nodi genitori, queste sono unite tramite l'operatore `AND` all'interno di una stringa di condizione.

Fino ad ora abbiamo discusso del concetto di grafo anziché di albero, poiché i nodi possono avere più nodi in ingresso provenienti da rami diversi del grafo. Quando ci troviamo in questa situazione, il nodo che riceve più di un input avrà una tabella associata che rappresenta un *OR* delle condizioni dei nodi in ingresso. Questo perché deve tener conto di due possibili scenari distinti.

Tale stringa viene poi passata come argomento alla funzione `eval`, insieme al DataFrame dei dati, al fine di determinare la validità della condizione rispetto ai dati in ingresso.

Questo approccio fornisce un metodo per permettere all'applicazione di valutare le regole definite nel grafo, garantendo una gestione dinamica e flessibile delle condizioni nel contesto dell'analisi dei dati.

Le condizioni booleane vengono applicate a un DataFrame mediante un processo ricorsivo sui nodi. Per essere più precisi, la funzione è eseguita su ogni figlio di un nodo, includendo anche la propria condizione che abbraccia l'intera gerarchia dei suoi "genitori".

Durante l'analisi del grafo, ciascun nodo viene associato a una porzione del DataFrame specifico su quella condizione, in modo da avere i dati pertinenti a quel nodo. Questo consente una buona gestione dei dati in base alle regole definite per ciascun nodo.

Durante l'applicazione delle regole, la valutazione delle condizioni specificate all'interno dei nodi viene eseguita considerando il DataFrame associato a ciascun nodo. Pertanto, ogni nodo agisce come un contesto autonomo per l'applicazione delle proprie regole, garantendo una chiara separazione dei dati e delle operazioni su di essi. Questo approccio favorisce la modularità del sistema, consentendo una gestione più agevole delle regole e dei dati nel contesto complessivo dell'analisi del grafo.

Per ottimizzare l'algoritmo, quando un nodo specifico genera un DataFrame vuoto in base alle regole definite, le regole dei suoi figli non vengono eseguite. Ciò è particolarmente rilevante poiché l'esecuzione di tali regole genererebbe a sua volta un DataFrame vuoto.

Il grafo viene trasmesso, in formato JSON, a Python mediante l'utilizzo di `AJAX` e `Flask`, ma in un formato JSON leggermente diverso da quello precedentemente illustrato. Questo formato è progettato per includere solo le informazioni strettamente necessarie a Python, evitando quindi di inviare un sovraccarico di dati.

Le informazioni essenziali trasmesse a Python includono l'identificativo del nodo, l'identificativo dei suoi figli e la condizione associata. Questa struttura consente un'applicazione più efficiente delle operazioni in Python.

3.8 Esecuzione tramite script

Se si dispone di un grafo completo generato dall'applicazione e si desidera applicarlo a una specifica tabella presente in un file CSV, è possibile eseguire lo script direttamente da terminale. Per farlo, è necessario avviare l'applicazione, la quale aprirà una pagina web.

Il backend di questa applicazione è implementato utilizzando **Flask**. Una volta avviata la pagina, si troveranno due buttoni che consentono di caricare il file JSON contenente il grafo e il file CSV contenente i dati su cui applicare le regole.

Listing 3.5: Esecuzione script

```
1 #!/bin/bash
2 python script.py #(file_funzioni.py)
```

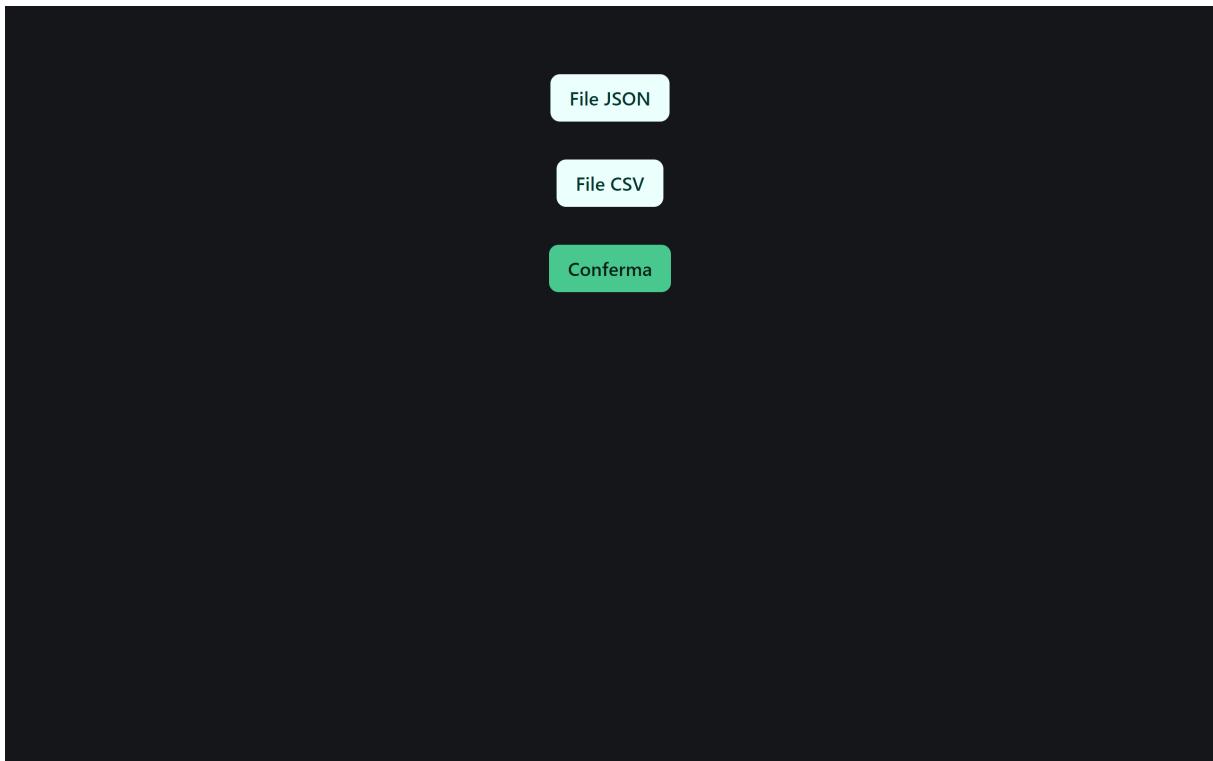


Figura 3.19: Schermata caricamento file

Una volta caricati entrambi i file, l'applicazione procederà ad applicare le regole definite nel grafo alla tabella CSV. Il processo genererà una nuova tabella a partire da quella ricevuta, aggiungendo una colonna risultante che rappresenta l'output delle varie regole presenti nel grafo.

CAPITOLO 3. DESCRIZIONE SOLUZIONE

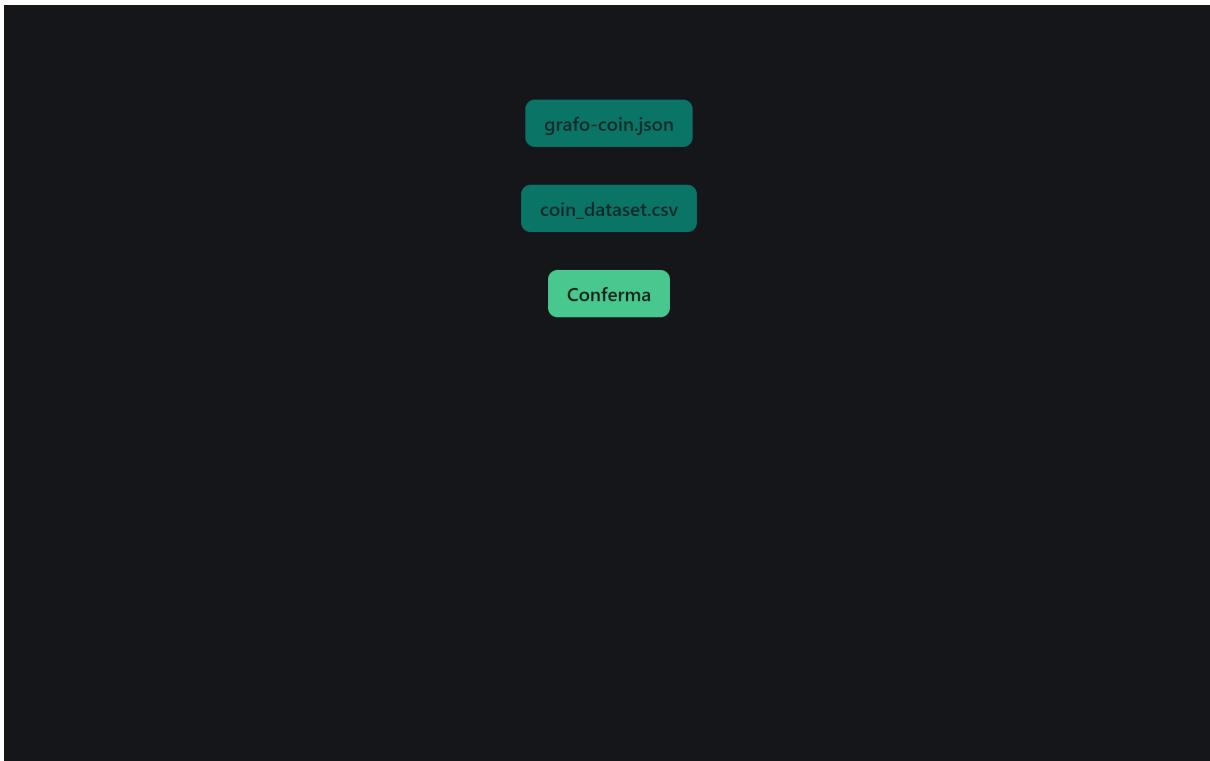


Figura 3.20: File caricati

Se la tabella conferma le aspettative e si vogliono applicare tali regole, premendo il pulsante "Salva", sarà possibile scaricare il file risultante contenente la tabella modificata con le regole applicate.

Risultato del Processo:										
Nome	Paese	Valuta	Anno	Prezzo	Rarità	Peso	Diametro	Materiale	Conservazione	Risultato
Aureus di Augusto	Roma	Denarius	14	5000	Rara	7.8	20.0	Oro	SPL	SPL - Rara
Tetradramma di Alessandro Magno	Macedonia	Dracma	-323	10000	Raro	17.2	25.0	Argento	BB	BB - Non Oro
Dinaro di Harun al-Rashid	Baghdad	Dinaro	786	2000	Rara2	2.9	22.0	Oro	MB	No SPL - No BB
Mezzo scudo di Ferdinando II	Napoli	Scudo	1836	150	Comune	13.5	31.0	Argento	B	No SPL - No BB
1/2 Krugerrand	Sudafrica	Rand	1980	500	Comune	15.5	27.0	Oro	SPL	SPL - No Rara
1/2 Corona	Austria	Corona	1915	50	Comune	1.2	19.0	Oro	MB	No SPL - No BB
1/2 Dollaro	Stati Uniti	Dollaro	1964	10	Rara	11.34	30.6	Argento	SPL	SPL - Rara
1/2 Rublo	Russia	Rublo	1898	100	Rara2	1.6	17.0	Argento	MB	No SPL - No BB
1/2 Sovrano	Regno Unito	Sterlina	1911	100	Comune	3.99	19.3	Oro	BB	BB - Oro
1/2 Yuan	Cina	Yuan	1991	10	Rara	4.5	22.0	Oro	SPL	SPL - Rara
1/2 Zecchino	Venezia	Ducato	1750	200	Raro	3.5	21.0	Oro	MB	No SPL - No BB
1/2 Dinaro	Jugoslavia	Dinaro	1970	5	Rara	2.5	18.0	Argento	BB	BB - Non Oro
1/2 Corona	Cecoslovacchia	Corona	1932	20	Comune	2.5	23.0	Argento	SPL	SPL - No Rara
1/2 Corona	Ungheria	Corona	1915	50	Comune	1.2	19.0	Oro	MB	No SPL - No BB
1/2 Corona	Romania	Corona	1930	10	Non Comune	2.5	23.0	Argento	BB	BB - Non Oro

Figura 3.21: Tabella risultante

Capitolo 4

Conclusione

Il percorso di sviluppo delineato in questo lavoro ha portato alla creazione di un sistema avanzato e innovativo per l'analisi dati basato su grafi di regole. Sebbene l'attenzione iniziale fosse focalizzata sull'applicazione specifica nel campo metabolomico, è cruciale sottolineare che l'approccio adottato offre versatilità e applicabilità più ampia.

Il progetto ha ottenuto risultati significativi, evidenziando l'efficacia del modello basato su grafi nella modellazione e interpretazione di relazioni complesse nei dati. L'automazione di questo processo ha notevolmente accelerato le analisi, consentendo una comprensione più rapida e approfondita dei dati, con possibili impatti positivi su vari settori.

Un elemento cruciale del progetto è stato l'accento posto sull'usabilità e sull'accessibilità. L'interfaccia utente è stata progettata con l'obiettivo di offrire una navigazione intuitiva e una facile comprensione, consentendo a utenti con diverse competenze di sfruttare appieno le potenzialità del sistema.

L'interfaccia completa di questo strumento non solo semplifica le operazioni di analisi dei dati, ma offre anche una visione dettagliata e completa delle informazioni. Gli utenti possono navigare agevolmente tra i diversi elementi del dataset, visualizzare chiaramente il grafo di regole creato e interpretare facilmente i risultati ottenuti dall'analisi.

Questa completezza dell'interfaccia contribuisce a garantire una user experience appagante, consentendo agli utenti di ottenere informazioni dettagliate senza la necessità di passaggi complicati o di dover consultare molteplici schermate.

Si riconosce, tuttavia, che ogni innovazione presenta sfide e limitazioni. La gestione di dataset estremamente vasti potrebbe richiedere ottimizzazioni aggiuntive, e l'adattamento del sistema a contesti diversificati rappresenta un'area in costante evoluzione.

4.1 Sviluppi Futuri

Guardando al futuro, si intende affrontare le limitazioni identificate e ampliare ulteriormente le capacità del sistema. Questo potrebbe coinvolgere sviluppi di algoritmi più sofisticati, l'integrazione di nuove tecnologie o esplorare approcci alternativi per affrontare specifiche sfide.

si prevede, inoltre, di esplorare la possibilità di introdurre nuove tipologie di nodi. Questo potrebbe arricchire ulteriormente le funzionalità del sistema, consentendo una maggiore flessibilità e adattabilità alle esigenze degli utenti. L'inclusione di nuove tipologie di nodi potrebbe aprire la strada a nuove modalità di rappresentazione e analisi dei dati, offrendo agli utenti uno strumento più completo e versatile per esplorare e interpretare le relazioni nei loro dataset

Alcune aggiunte mirate alla personalizzazione potrebbero includere la possibilità di apportare variazioni più ampie all'aspetto dei nodi. Attualmente, l'unica personalizzazione consentita riguarda la modifica del colore del nodo, ma si potrebbe valutare l'introduzione di opzioni supplementari come la personalizzazione dei collegamenti tra i nodi o dei punti di collegamento.

Altri miglioramenti, che si concentrano su aspetti più tecnici, potrebbero riguardare l'ottimizzazione dell'inserimento delle condizioni. Attualmente, le condizioni vengono inserite manualmente, ma un perfezionamento potrebbe consistere nel collegare direttamente il campo di inserimento delle condizioni al dataset, facendo riferimento direttamente alle colonne o persino ai valori specifici delle colonne.

Una prospettiva di miglioramento ulteriore potrebbe contemplare la migrazione alla versione 2 di BaklavaJS, apportando notevoli benefici in termini di efficienza e performance al sistema. La versione aggiornata del framework potrebbe offrire ottimizzazioni e nuove funzionalità che contribuirebbero a potenziare l'applicazione, assicurando un'esperienza ancor più fluida e avanzata agli utenti. Tale evoluzione rappresenterebbe un passo significativo verso l'adattamento alle più recenti tecnologie e al continuo miglioramento delle prestazioni complessive del sistema.

C'è la consapevolezza che, in un progetto di questa portata, la presenza di bug sia inevitabile. Tuttavia, è determinante affrontare eventuali problematiche. Ci si impegna a identificare le cause sottostanti di ciascun problema e ad adottare le misure necessarie per mitigare gli effetti. Pur comprendendo che l'eliminazione totale dei bug potrebbe essere un obiettivo irrealistico, ci si impegna comunque a cercare di risolverli nel miglior modo possibile per garantire la migliore esperienza utente.

Sitografia

- [1] BaklavaJS. <http://v1.baklava.tech>
- [2] BulmJS. <http://bulma.io>
- [3] IziToastJS. <http://izitoast.marcelodolza>
- [4] SweetAlertJS. <http://sweetalert.js.org>
- [5] Flask. flask.palletsprojects.com <https://flask.palletsprojects.com>
- [6] Rivest, Ronald L. "NP-Completezza." <https://people.csail.mit.edu/rivest/HyafilRivest-ConstructingOptimalBinaryDecisionTreesIsNPComplete.pdf>
- [7] Quinlan, J. Ross. "Ottimizzazione." <https://www.sciencedirect.com/science/article/abs/pii/S0020737387800536>
- [8] Zampieri, Nicolo. "Engine ottimizzata." https://cds.cern.ch/record/2688585/files/AA_main.pdf

Bibliografia

- [1] C. Demetrescu, I. Finocchi, G. Italiano, *Algoritmi e strutture dati*, McGraw-Hill, 2008.