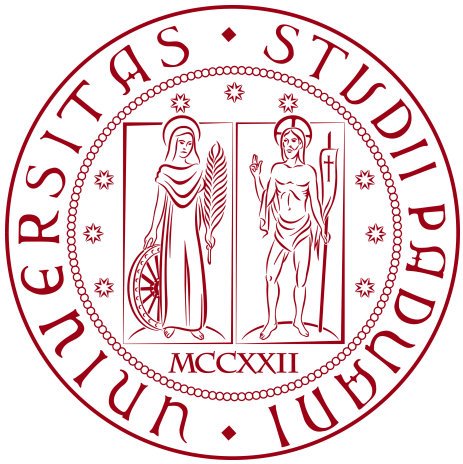
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una libreria per la visualizzazione di dashboard dinamiche.

Tesi di Laurea Triennale

Relatore
Prof. Baldan Paolo

Laureando
Bresolin Gianluca
Matricola 2034316

Anno Accademico 2023-2024

'L'importante non è vincere, è pensare in modo vincente. La vita è fatta per il 10% da quello che succede e per il 90% da come lo affrontiamo.'

Gianluca Vialli.

Ringraziamenti

Desidero esprimere la mia gratitudine al professor Baldan Paolo, mio relatore, per l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Vorrei esprimere il mio più affettuoso ringraziamento ai miei genitori, Marta e Riccardo, e a mia sorella Emma, per il costante sostegno e per i valori che mi hanno trasmesso, i quali mi hanno permesso di diventare la persona che sono oggi e di raggiungere i miei traguardi.

Desidero esprimere un ringraziamento speciale alla mia amata, Rebecca, per il suo supporto, per il suo amore e per la sua costante presenza che mi ha permesso di affrontare con serenità e tranquillità questo percorso di studi, non facendomi mai sentire solo.

Ringrazio infine i miei amici Nicola, Manuel, Leonardo, Matteo e Andrea per la loro compagnia, per i momenti di divertimento e per avermi sempre mantenuto in contatto con la vita e il mondo esterno.

Padova, Luglio 2024 *Bresolin Gianluca*

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 304 ore, dal laureando Gianluca Bresolin presso l'azienda *Datasoil S.r.l.* Lo stage ha previsto la realizzazione di una libreria ReactJS attraverso il refactor e l'ottimizzazione di una libreria utilizzata per la visualizzazione di dashboard parametriche, volta ad ampliare soluzioni e servizi nel contesto di applicazioni SPA e nella realizzazione di interfacce utente, ponendo attenzione ai requisiti di scalabilità, usabilità e performance.
La prima attività condotta è stata l'analisi del codice sorgente della libreria esistente, in modo da poter comprendere il funzionamento e le opportunità di ottimizzazione. Successivamente, sono state individuate le funzionalità da mantenere, quelle da migliorare e quelle da introdurre, per poi procedere con la progettazione e l'implementazione della nuova libreria.
Infine, la libreria realizzata è stata integrata nei prodotti *Datasoil* attraverso un micro servizio dedicato, rendendo disponibile un SDK compatibile con le risposte delle API Datasoil, senza richiedere modifiche lato server ai prodotti esistenti.
Il risultato ottenuto al termine del periodo di stage consiste un SDK per la composizione dinamica di dashboard, dalle dimensioni ridotte rispetto al boundle precedentemente utilizzato dall'azienda e dotato di una maggior robustezza e consistenza grazie ad un uso più opportuno di *TypeScript*, permettendo inoltre di raggiungere una migliore aderenza a quelle che sono le esigenze dei clienti, grazie alla rimozione di grafici inutilizzati e l'introduzione di nuove funzionalità.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici sorgenti

Acronimi e abbreviazioni

API Application Programming Interface. 11, *Glossary:* API

FCP First Contentful Paint. 2, *Glossary:* FCP

SDK Software Development Kit. 2, *Glossary:* SDK

SVG Scalable Vector Graphics. 38, *Glossary:* SVG

TTI Time to Interactive. 2, *Glossary:* TTI

Glossario

API Un'*API (Application Programming Interface)* è un insieme di definizioni e protocolli che permettono a diversi software di comunicare tra loro. Le API forniscono un'interfaccia standardizzata che consente agli sviluppatori di accedere a funzionalità e servizi di un altro software, sistema operativo, libreria o framework senza conoscere i dettagli della loro implementazione . 11

Backend Il *backend* si riferisce alla parte server di un'applicazione, che gestisce la logica, le operazioni di database, l'autenticazione degli utenti e altre funzionalità che non sono visibili direttamente dall'utente finale. 38

Bundle Un *bundle* è un pacchetto che contiene diversi file e risorse che vengono aggregati insieme per essere distribuiti come un'unica unità. In ambito web, un *bundle* può includere file *JavaScript*, *CSS* e immagini. 5

Bundler Un *bundler* è uno strumento utilizzato nello sviluppo web per combinare diversi file e risorse, come *JavaScript*, *CSS* e immagini, in un unico file o in un insieme di file più ridotto. Questo processo migliora l'efficienza del caricamento delle pagine web riducendo il numero di richieste HTTP necessarie. 43

FCP Il *First Contentful Paint (FCP)* è una metrica che misura il tempo dal momento in cui una pagina inizia il caricamento fino al momento in cui qualsiasi parte del contenuto della pagina viene resa visibile sullo schermo . 2

Framework Un *framework* è un insieme di librerie, strumenti e linee guida che forniscono una struttura e un'infrastruttura per lo sviluppo di applicazioni software. I *framework* semplificano il processo di sviluppo fornendo funzionalità comuni, standardizzando le pratiche di programmazione e riducendo la complessità del codice. 5

Frontend Il *frontend* si riferisce alla parte visibile di un'applicazione, che interagisce direttamente con l'utente finale. Questa parte dell'applicazione gestisce l'interfaccia utente, la presentazione dei dati e le interazioni con l'utente. 7

Memoizzato Il termine *memoizzato* si riferisce a una tecnica di ottimizzazione che memorizza il risultato di una funzione o di un calcolo in modo da poterlo riutilizzare in seguito senza doverlo ricalcolare. Questo approccio migliora le presta-

zioni dell'applicazione riducendo il tempo di esecuzione e l'utilizzo delle risorse. 33

Mock Un *mock* è un oggetto simulato o fittizio che viene utilizzato al posto di un oggetto o una funzione reale durante i test software. I *mock* vengono utilizzati per simulare il comportamento di un oggetto o funzione reale, consentendo ai test di verificare il funzionamento del codice in modo controllato e prevedibile, limitando le responsabilità testate alle sole logiche specifiche. 60

Munging Il *munging* è una tecnica di trasformazione dei dati che modifica o oscura i dati in modo da renderli irriconoscibili o incomprensibili per chi non è autorizzato a visualizzarli. Il *munging* viene spesso utilizzato per proteggere i dati sensibili o per nascondere informazioni personali, come indirizzi email o numeri di telefono, sostituendo i caratteri con simboli o codici. Questa tecnica viene inoltre utilizzata per ottimizzare il codice sorgente, riducendo la dimensione dei file. 65

Open-Source Il termine *open-source* si riferisce a un tipo di software il cui codice sorgente è disponibile per l'uso, la modifica e la distribuzione da parte di chiunque. Questo modello promuove la collaborazione e la trasparenza nello sviluppo software. 2

Package Un *package* è un insieme di moduli o classi che vengono raggruppati insieme e distribuiti come una singola unità. I pacchetti facilitano la gestione e la distribuzione del software, includendo tutte le dipendenze necessarie. 37

Peer Dependency Una *peer dependency* è una dipendenza di un modulo che non viene installata automaticamente, ma che deve essere fornita dall'ambiente in cui il modulo stesso è eseguito. Questo tipo di dipendenza è utilizzato per garantire che i moduli condividano una singola istanza di una libreria comune, evitando problemi di incompatibilità e ridondanza.. 40

Refactoring Il refactoring è il processo di ristrutturazione del codice sorgente di un programma senza modificarne il comportamento esterno. L'obiettivo è migliorare la leggibilità, la manutenibilità e la riduzione della complessità del codice. 2

Repository Un *repository* è un archivio centralizzato e strutturato per la conservazione, la gestione e il controllo di versioni di codice sorgente, documentazione e altri file correlati a un progetto software. I repository possono essere ospitati su server locali o remoti e sono spesso gestiti tramite sistemi di controllo versione, consentendo agli sviluppatori di collaborare, tracciare modifiche, gestire diramazioni (*branch*) e fusioni (*merge*), mantenendo una cronologia accurata dello sviluppo del software. 45

Responsive Il termine *responsive* si riferisce a un design che è in grado di adattarsi a diverse dimensioni dello schermo e dispositivi, offrendo un'esperienza utente ottimale, indipendentemente dal dispositivo utilizzato. 39

Runtime Il *runtime* si riferisce al periodo di tempo durante il quale un programma è in esecuzione. Il termine viene anche utilizzato per indicare un ambiente o una libreria che supporta l'esecuzione di un programma, fornendo servizi come la gestione della memoria e l'esecuzione del codice. 39

SDK Un *Software Development Kit (SDK)* è un insieme di strumenti di sviluppo software riuniti in un unico pacchetto installabile. Gli SDK sono progettati per semplificare il processo di sviluppo di applicazioni, fornendo librerie, strumenti di sviluppo, documentazione e esempi di codice . 2

Super-Set Un super-set è un insieme che contiene tutti gli elementi di un altro insieme, detto sottoinsieme. Nel contesto della programmazione, un linguaggio o un *framework* può essere considerato un *super-set* se include tutte le funzionalità di un altro linguaggio o *framework*, aggiungendo ulteriori caratteristiche e capacità.. 41

SVG L'*SVG (Scalable Vector Graphics)* è un formato di file basato su *XML* per descrivere immagini vettoriali bidimensionali. Questo formato permette la rappresentazione di grafica vettoriale con alta precisione, consentendo lo zoom e il ridimensionamento senza perdita di qualità. Gli SVG sono utilizzati ampiamente nel *web design* e nelle applicazioni grafiche grazie alla loro scalabilità e al supporto per interattività e animazioni tramite script e fogli di stile . 38

Tree-Shaking Il *tree-shaking* è una tecnica di ottimizzazione del codice utilizzata dai *bundler* per rimuovere codice inutilizzato dai file JavaScript. Questo processo analizza le dipendenze del codice per identificare e eliminare le parti che non sono effettivamente utilizzate, riducendo la dimensione finale del *bundle* e migliorando le prestazioni delle applicazioni web. 43

TTI Il *Time to Interactive (TTI)* è una metrica che misura il tempo necessario affinché una pagina web diventi completamente interattiva, cioè quando è possibile interagire con tutti gli elementi principali della pagina senza ritardi significativi . 2

Capitolo 1

Introduzione

1.1 L'azienda

Lo stage è stato svolto presso l'azienda *Datasoil S.r.l.* situata nei dintorni della stazione ferroviaria di Padova. Fondata nel 2016, *Datasoil S.r.l.* è un'azienda di prodotto che si dedica allo sviluppo di piattaforme per l'industria 4.0, *smart building* e *smart city*. L'obiettivo dell'azienda è integrare informazioni ed eventi provenienti dai vari livelli aziendali per creare *insight* proattivi in tempo reale, garantendo che le informazioni corrette raggiungano le persone giuste al momento più opportuno. Il punto di partenza è la spazialità aziendale e tutti gli *asset* che risiedono all'interno di essa, da cui provengono tutti i dati raccolti, i quali vengono analizzati trasversalmente grazie a sempre più persone e dispositivi connessi. Da qui il nome dell'azienda, *Datasoil*: fare emergere le informazioni da questo suolo fertile di dati da cui siamo circondati.

Figura 1.1: Logo *Datasoil S.r.l.*

1.2 L'idea

1.2.1 Il contesto applicativo

L'azienda *Datasoil S.r.l.*, essendo un'azienda di prodotto, nasce con l'idea di proporre servizi di monitoraggio e controllo di impianti industriali, *smart building* e *smart city*, offrendo come principale prodotto la piattaforma SYN.

Figura 1.2: Logo SYN

SYN è una piattaforma di monitoraggio e controllo di impianti industriali che raccoglie dati da sensori e dispositivi distribuiti all'interno di un impianto o su più stabilimenti, permettendo di visualizzare in tempo reale lo stato di funzionamento dei vari *asset*, di analizzare i dati raccolti e di attuare azioni di controllo o segnalazione tramite *ticketing*. All'interno della piattaforma, a seconda dei piani attivi del cliente, è possibile visualizzare dashboard dinamiche in merito ad informazioni filtrate e aggiornate *live*, permettendo di raggiungere una panoramica completa dello stato dei vari *asset* monitorati in modo rapido, intuitivo ed efficace, grazie all'utilizzo di molteplici grafici e *widget*.

1.2.2 Il progetto

Il progetto svolto durante lo stage consiste nello sviluppo di una libreria *TypeScript* di componenti per la creazione di dashboard dinamiche. Questo è stato realizzato tramite il refactoring$_G$ e l'ottimizzazione di un *tool* grafico preesistente, integrato nei vari prodotti di *Datasoil S.r.l.*, tra cui *SYN*. La libreria è stata implementata a partire da un modulo *open-source*$_G$ utilizzato in *Redash* (redash.io), una piattaforma per la creazione di dashboard dinamiche tramite interrogazioni sulle fonti di dati configurate all'interno del servizio.
L'esigenza di tale progetto nasce dalla necessità da parte di *Datasoil S.r.l.* di avere una libreria grafica aggiornata alle versioni correnti delle dipendenze utilizzate, migliorando il First Contentful Paint (FCP)$_G$ e il Time to Interactive (TTI)$_G$, raggiungendo una maggiore manutenibilità del codice rispetto alla versione preesistente e riducendo dove possibile le dipendenze esterne e introducendo migliorie grafiche e funzionali.
La libreria sviluppata sotto il nome '*viz-lib*', costituisce, insieme alla già esistente libreria '*dashboard*' (la quale ha subito anch'essa in parte un processo di refactoring per permettere l'integrazione con la nuova libreria e l'introduzione di nuove funzionalità), il nuovo Software Development Kit (SDK)$_G$ '*dsdashbord2*' utilizzato all'interno dei prodotti *Datasoil S.r.l.* per la realizzazione di dashboard dinamiche.

1.3 Principali problematiche

Durante l'analisi iniziale della libreria preesistente in uso all'interno dei prodotti di *Datasoil S.r.l.*, sono emerse alcune problematiche legate alla manutenibilità e alla performance del *tool* grafico a seguito di una attenta revisione del codice sorgente.
Di seguito vengono presentate le principali problematiche riscontrate.

Dipendenze obsolete

La libreria preesistente utilizzava versioni obsolete delle dipendenze esterne, con conseguente degradazione delle prestazioni in termini di utilizzo di spazio e di tempo di caricamente delle risorse, data l'assenza di ottimizzazioni e di aggiornamenti del codice sorgente.
Essendo inoltre la libreria preesistente basata su una versione del modulo di visualizzazioni utilizzato nella piattaforma *Redash*, le tecnologie utilizzate il più delle volte rappresentavano alternative a quelle già utilizzate nei prodotti *Datasoil S.r.l.*, creando dipendenze non necessarie, con conseguente aumento delle dimensioni del SDK utilizzato per la generazione delle dashboard.

Manutenibilità del codice

Il codice sorgente della libreria preesistente, sviluppato in tempistiche rapide a fronte di una esigenza specifica dell'azienda *Datasoil S.r.l.*, risultava essere poco manutenibile e assente di documentazione.
Inoltre, la libreria di visualizzazioni utilizzata da *Redash* su cui si basa l'SDK utilizzato dall'azienda, inizialmente implementata in JavaScript, ha subito solo successivamente un refactoring in TypeScript: questo refactoring non è però avvenuto con l'introduzione di interfacce e tipi, bensì con la semplice aggiunta di tipi *any* per le variabili e per i parametri delle funzioni, introducendo numerosi *@ts-ignore* per ignorare gli errori di compilazione, rendendo il codice poco leggibile e difficile da mantenere.

Componenti inutilizzate

In quanto la libreria preesistente fosse realizzata a partire dal modulo utilizzato dalla piattaforma *Redash*, non tutti i componenti presenti in essa trovavano utilizzo all'interno dei prodotti realizzati da *Datasoil S.r.l.*: questo in gran parte era dovuto ad un'offerta di grafici non adeguati a quello che è il contesto applicativo dell'azienda.

Assenza di funzionalità

La libreria preesistente non soddisfaceva pienamente le esigenze dei clienti di *Datasoil S.r.l.*, mancando di alcune funzionalità fondamentali, quali la possibilità di effettuare il *download* dei dati tabellari o la visualizzazione a pagina piena dei vari *widget*. Queste limitazioni hanno reso necessario un intervento di refactoring per colmare le lacune e migliorare le prestazioni complessive.

1.4 Soluzione scelte

Per risolvere le problematiche emerse durante l'analisi iniziale della libreria preesistente, si è optato per il refactoring e l'ottimizzazione del codice sorgente, individuando le soluzioni presentate di seguito.

Aggiornamento delle dipendenze

Per risolvere il problema delle dipendenze obsolete, è stata effettuata un'analisi delle versioni correnti delle dipendenze utilizzate all'interno della libreria, verificando che le funzionalità utilizzate non fossero deprecate o rimosse, procedendo con l'eventuale refactoring ad un codice compatibile che tenesse inoltre conto delle nuove funzionalità introdotte nelle versioni più recenti.
Grazie ad uno studio accurato delle librerie utilizzate nei prodotti *Datasoil S.r.l.*, è stato possibile sostituire con librerie equivalenti dipendenze preesistenti, riducendo così l'onere di utilizzo dell'SDK all'interno dei prodotti aziendali.

Introduzione di interfacce e tipi
Al fine di migliorare la manutenibilità e la comprensione del codice sorgente, è stato svolto un lavoro di introduzione di interfacce e tipi per le variabili e per i parametri delle funzioni, in modo da rendere il codice più leggibile e permettere controlli statici sul codice sorgente.
Questo lavoro ha permesso di ridurre la presenza di tipi *any* all'interno del codice sorgente, garantendo una maggiore sicurezza e affidabilità del prodotto finale.

Refactoring del codice
Per migliorare la comprensione e la manutenibilità del codice sorgente, è stato intrapreso un lavoro di refactoring su alcune porzioni della libreria preesistente. Questo intervento ha comportato la riscrittura di diverse funzioni e la rimozione di componenti minori utilizzate per eseguire funzionalità semplici, attraverso un codice più leggibile e immediato, eliminando complessità superflue e mantenendo l'efficacia nell'implementazione di operazioni elementari.

Rimozione di componenti inutilizzate
In merito alla presenza di componenti inutilizzate all'interno della libreria preesistente, è stata effettuata un'accurata analisi delle componenti presenti, selezionando quelle che non trovavano un diretto utilizzo all'interno dei prodotti *Datasoil S.r.l.* e procedendo con la loro rimozione.
Questo lavoro ha permesso di ridurre le dimensioni del codice sorgente e di migliorare le prestazioni complessive della libreria, diminuendo inoltre il numero di dipendenze esterne richiesto per il corretto funzionamento dell'SDK prodotto.

Implementazione di nuove funzionalità
Per colmare le lacune riscontrate nella libreria preesistente in merito all'assenza di alcune funzionalità richieste dai clienti di *Datasoil S.r.l.* all'interno dei prodotti forniti, è stato condotto un lavoro di implementazione di nuove funzionalità.
In questo processo, si è cercato di utilizzare dipendenze esterne preferibilmente già in dotazione o che fossero compatibili con il contesto applicativo dell'azienda. Qualora ciò non fosse stato possibile, sono state introdotte nuove dipendenze esterne selezionate a seguito di un'analisi dettagliata che tenesse conto delle implicazioni in termini

di dimensioni finali dell'SDK prodotto.

1.5 Risorse Tecnologiche

La realizzazione della libreria grafica prodotta ha visto l'utilizzo di differenti tecnologie.
Per quanto riguarda l'implementazione dell'SDK in *TypeScript*, è stato utilizzato il *framework*$_G$ *React* per la creazione dei componenti grafici, con il supporto di *PrimeReact, PrimeFlex* e *PrimeIcons*, utilizzando la libreria *Plotly.js* per la generazione dei grafici.
Per la gestione delle dipendenze è stato utilizzato il *package manager npm*, mentre per la generazione e gestione del *bundle*$_G$ è stato impiegato *Rollup*.
In merito al testing del prodotto, è stato integrato il *framework* di testing *Jest* all'interno del progetto per la scrittura ed esecuzione dei test.
Infine, per la documentazione del codice sorgente è stato utilizzato *Confluence* di *Atlassian* mentre, per quanto riguarda la comunicazione interna con il team di sviluppo, è stato impiegato *Slack*.

1.6 Descrizione del prodotto ottenuto

Il prodotto conseguito al termine dello stage consiste in una libreria *TypeScript* di componenti utilizzati nel processo di creazione di dashboard dinamiche, ottenuto a partire dal refactor e l'ottimizzazione della libreria *open-source* utilizzata all'interno della piattaforma *Redash*. La libreria implementata, sotto il nome di *viz-lib*, è stata infatti integrata assieme alla già esistente libreria *dashboard* all'interno del SDK *dsdashboard2*, utilizzato all'interno dei prodotti *Datasoil S.r.l.* per la generazione di dashboard dinamiche. La libreria *viz-lib* offre una vasta gamma di grafici, quali:

- Contatori;
- Grafici ad area;
- Grafici a barre;
- Grafici a bolle;
- Grafici a dispersione;
- Grafici a istogrammi;
- Grafici a linee;
- Grafici a torta;
- Tabelle.

Grazie al refactor e all'ottimizzazione della libreria preesistente, l'SDK prodotto risulta essere più performante, più manutenibile e soprattutto più leggero rispetto alla versione preesistente, con un *bundle* di dimensioni 1.26 MB → 418.03 kB (gzip) rispetto ai 4.11 MB → 1.203 MB (gzip) iniziali.

1.7 Organizzazione del testo

Nel presente capitolo viene presentata l'introduzione della tesi, fornendo una panoramica sull'azienda, sul contesto applicativo, sul progetto, sugli strumenti utilizzati e sul prodotto portato a termine durante lo svolgimento dello stage.
In seguito il documento presenterà la seguente organizzazione:

Il Capitolo 2 descrive la fase di analisi dei requisiti che è stata svolta dall'azienda in fase antecedente all'inizio dell'attività di stage, in modo da permettere una comprensione più profonda di quelle che sono le necessità da soddisfare e gli obiettivi da raggiungere all'interno di questo progetto;

Il Capitolo 3 illustra l'attività di progettazione che è stata svolta in vista dell'implementazione dei grafici prodotti, definendo e individuando le soluzioni implementative che sono state attuate durante la successiva attività di codifica;

Il Capitolo 4 approfondisce l'attività di codifica delle componenti grafiche presentate all'interno della libreria implementata e dei relativi test effettuati per garantire la qualità e la funzionalità del prodotto finale;

Il Capitolo 5 descrive l'attività di rilascio dell'SDK realizzato, presentando le modalità di integrazione all'interno di un prodotto *Datasoil S.r.l.*;

Il Capitolo 6 presenta un epilogo del progetto svolto, includendo un consuntivo finale delle attività svolte, una valutazione del progetto e una sezione dedicata ai possibili sviluppi futuri del prodotto. La sezione infine si conclude con una riflessione finale sull'esperienza di stage svolta.

In merito alla stesura del testo, all'interno del presente documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*$_{\boldsymbol{G}}$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Analisi dei requisiti

In questo capitolo verrà descritta l'attività di analisi dei requisiti inerente al progetto di stage.
In quanto l'SDK in questione è uno strumento *live* sui diversi prodotti *Datasoil S.r.l.*, l'analisi dei requisiti è stata svolta in una fase antecedente all'inizio dello stage da parte del team di sviluppo *frontend*$_G$ dell'azienda.
Per questa ragione, nella seguente sezione verranno riportati i requisiti individuati in modo da poter fornire il contesto necessario alla comprensione del progetto, senza approfondire l'attività di analisi svolta.

2.1 Classificazione dei requisiti

I requisiti presentati all'interno del documento vengono classificati in tre categorie, in base alle esigenze che definiscono, venendo inoltre associati ad un livello di priorità che ne determina l'obbligatorietà o l'opzionalità.

Le categorie di classificazione definite all'interno del progetto di stage sono:

F: *Funzionale*: il requisito definisce le funzionalità che il sistema deve offrire;

A: *Di attributo*: il requisito definisce le caratteristiche che il sistema deve avere;

V: *Di vincolo*: il requisito definisce i vincoli che il sistema deve rispettare.

I livelli di priorità definiti all'interno del progetto di stage sono:

O: *Obbligatorio*: requisito essenziale al fine di concludere il progetto di stage con successo;

OP: *Opzionale*: requisito non essenziale al fine di concludere il progetto di stage con successo, ma che aggiunge valore al prodotto finale.

La nomeclatura utilizzata per la classificazione dei requisiti all'interno del progetto di stage è la seguente:

- **R**: dove R sta per requisito;
- **X.**: dove X indica la categoria di classificazione del requisito;
- **Y.**: dove Y indica il livello di priorità del requisito;
- **Z**: dove Z indica il numero progressivo del requisito. Nel caso di sotto-requisiti, il numero progressivo è composto da più cifre, separate da un punto, dove la cifra antecedente rappresenta il requisito padre e la cifra conseguente rappresenta il sotto-requisito.

2.2 Requisiti

2.2.1 Requisiti funzionali

Nella seguente sezione viene presentata la lista dei requisiti funzionali.

Requisito	Descrizione
R.F.O.1	La libreria deve permettere la renderizzazione di grafici.
R.F.O.1.1	La libreria deve permettere la renderizzazione di grafici a barre (*bar or columns*).
R.F.O.1.2	La libreria deve permettere la renderizzazione di grafici a torta (*pie*).
R.F.O.1.3	La libreria deve permettere la renderizzazione di grafici a scatola (*box-plot*).
R.F.O.1.4	La libreria deve permettere la renderizzazione di grafici a bolle (*bubbles*).
R.F.O.1.5	La libreria deve permettere la renderizzazione di grafici a dispersione (*scatter*).
R.F.O.1.6	La libreria deve permettere la renderizzazione di grafici a mappe di calore (*heatmap*).
R.F.O.2	La libreria deve permettere la renderizzazione di contatori.
R.F.O.2.1	La libreria deve permettere la renderizzazione di contatori con target.
R.F.OP.2.1.1	La libreria deve permettere la renderizzazione di contatori con target, definendo una grafica differente a seconda del raggiungimento o meno del target.
R.F.O.2.2	La libreria deve permettere la renderizzazione di contatori con formattazione personalizzata.
R.F.O.2.3	La libreria deve permettere la renderizzazione di contatori con tooltip con formattazione personalizzata.
R.F.O.3	La libreria deve permettere la renderizzazione di tabelle.
R.F.O.3.1	La libreria deve permettere la renderizzazione di tabelle con ordinamento lungo le varie colonne.
R.F.O.3.2	La libreria deve permettere la renderizzazione di tabelle con filtro globale che agisce lungo tutti i dati.
R.F.O.3.3	La libreria deve permettere la renderizzazione di tabelle con le colonne ordinate secondo l'evenutale ordine fornito dal backend.
R.F.O.4	La libreria deve permettere lo zoom nei i vari grafici.

Continua nella prossima pagina...

Tabella 2.1 – continua dalla pagina precedente	
Requisito	**Descrizione**
R.F.O.5	La libreria deve permettere di effettuare il download in *.png* dei grafici renderizzati.
R.F.O.6	La libreria deve permettere di effettuare il download in *.csv* delle tabelle visualizzate.
R.F.O.7	I widget renderizzati devono adattarsi alla dimensione del contenitore in modo responsivo.
R.F.O.8	La libreria deve permettere di visualizzare i widget in modalità *fullscreen* attraverso l'utilizzo di *Dialog*.

Tabella 2.2: Tabella del tracciamento dei requisiti funzionali.

2.2.2 Requisiti qualitativi

Nella seguente sezione viene presentata la lista dei requisiti qualitativi.

Requisito	**Descrizione**
R.A.O.1	La libreria deve essere facilmente manutenibile, rispettando le convenzioni di codifica attualmente in atto all'interno dell'azienda *Datasoil S.r.l.*
R.A.O.2	La libreria deve essere performante in termini di velocità di rendering dei *widget* forniti.
R.A.O.3	La libreria deve essere ottimizzata per richiedere il minor numero di rendering possibili durante l'interazione dell'utente, eseguendoli in modo efficiente per minimizzare l'uso delle risorse.
R.A.O.3	La libreria deve essere leggera in termini di dimensione di memoria occupata.
R.A.O.3.1	Le dipendenze esterne devono essere ridotte al minimo indispensabile.
R.A.0.3.2	La libreria deve essere composta esclusivamente da codice effettivamente utilizzato per le funzionalità e per i *widget* resi disponibili: qualsiasi componente di codice inutilizzata, superflua o non necessaria deve essere identificata e rimossa.
R.A.O.4	La libreria deve mantenere la compatibilità con i dati in formato *JSON* forniti dal backend, precedentemente utilizzati nel preesistente SDK *dsdashboard*.
Continua nella prossima pagina...	

Tabella 2.3 – continua dalla pagina precedente	
Requisito	**Descrizione**
R.A.O.5	La libreria deve mantenere la compatibilità con le *Application Programming Interface (API)*$_G$ del *backend* esistenti, garantendo che tutte le richieste e le risposte continuino a funzionare correttamente.
R.A.O.6	Le modifiche effettuate ai tipi di dato all'interno della nuova libreria devono essere retrocompatibili con il preesistente SDK *dsdashboard*.

Tabella 2.4: Tabella del tracciamento dei requisiti qualitativi.

2.2.3 Requisiti di vincolo

Nella seguente sezione viene presentata la lista dei requisiti di vincolo.

Requisito	Descrizione
R.V.O.1	La libreria deve essere sviluppata in *TypeScript*.
R.V.O.2	La libreria deve essere sviluppata con *React*.
R.V.O.3	La libreria deve essere sviluppata con l'utilizzo di componenti di *PrimeReact*.
R.V.O.4	La libreria deve essere sviluppata con l'utilizzo di *PrimeFlex*.
R.V.O.5	La libreria deve essere sviluppata con l'utilizzo di *PrimeIcons*.
R.V.O.6	Il codice della libreria deve essere formattato mediante l'utilizzo di *Prettier*.
R.V.OP.7	Il processo di rialscio deve avvenire all'interno del registro privato *Github Packages* di *Datasoil S.r.l.*

Tabella 2.6: Tabella del tracciamento dei requisiti di vincolo.

Capitolo 3

Progettazione

Nel presente capitolo verrà presentata l'attività di progettazione relativa al progetto di stage focalizzato sul refactoring e sull'ottimizzazione dello SDK impiegato dall'azienda *Datasoil S.r.l.* per lo sviluppo di dashboard dinamiche. Verrà analizzato l'approccio adottato per individuare le soluzioni volte al soddisfacimento dei requisiti individuati durante la fase di analisi dei requisiti, proseguendo con la descrizione dell'architettura progettata per la realizzazione della libreria grafica *viz-lib* appartenente al *development kit*, concludendo con la definizione dei tipi e delle interfacce e la progettazione delle singole componenti individuate.

3.1 Metodologia di progettazione

La metodologia di progettazione adottata durante lo svolgimento del progetto di stage ha seguito inizialmente un approccio *top-down*, che permettesse, attraverso uno studio generale della libreria preesistente e della sua architettura, di individuare il flusso di dati tra le varie componenti, fornendo una visione globale delle interazioni e delle dipendenze esistenti. Successivamente, è stato adottato un approccio *bottom-up*, analizzando le singole componenti e le loro funzionalità, in modo da identificare le possibili ottimizzazioni e le modifiche necessarie per il miglioramento delle prestazioni, della leggibilità e della manutenibilità del codice.
L'attività di progettazione è stata condotta in collaborazione stretta con il tutor aziendale, il quale ha fornito supporto e orientamento riguardo alle decisioni progettuali individuate.

3.2 Design dell'architettura

L'architettura progettata per la realizzazione della libreria grafica si basa su un pattern comunemente utilizzato nello sviluppo di *packages* di componenti *frontend* per *React*, ovvero il pattern *Component Library Architecture*: questa architettura si basa sulla creazione di una libreria di componenti riutilizzabili, modulari e indipendenti tra loro.

Tale pattern permette di creare l'architettura ideale per librerie che offrono collezioni di componenti riutilizzabili, permettendo e agevolando la condivisione e il riutilizzo di codice all'interno del progetto, garantendo una maggior manutenibilità del codice. Il modello proposto prevede la seguente struttura, la quale si riflette nella definizione delle *macro-directory* all'interno del progetto:

- **components**: *directory* contenente i componenti della libreria grafica;
- **hooks**: *directory* contenente i *custom hooks* utilizzati all'interno della libreria grafica;
- **utils**: *directory* contenente le funzioni di utilità utilizzate all'interno della libreria grafica;
- **models**: *directory* contenente le definizioni dei tipi e delle interfacce utilizzate all'interno della libreria grafica;
- **index.ts**: file principale della libreria grafica, contenente l'esportazione di tutti i componenti e le funzioni utilizzate;
- **style.css**: file contenente i fogli di stile globali utilizzati all'interno della libreria grafica.

L'*entry point* della libreria grafica è il file *index.ts*, il quale si occupa di esportare il `Renderer`, il componente principale della liberia che si occupa di renderizzare il corretto componente a seconda del tipo di visualizzazione richiesto: tali informazioni necessarie vengono passate come *props* dalle componenti del modulo *dashboard* che utilizzano il `Renderer`, riducendo così la dipendenza tra le componenti di librerie differenti, agevolando la manutenibilità e la scalabilità del codice.

3.3 Progettazione dei tipi e delle interfacce

Nella seguente sezione vengono presentati i tipi e le interfacce definiti durante l'attività di progettazione, associati ad una descrizione dettagliata delle informazioni che rappresentano.
La definizione dei tipi e delle interfacce ha costituito un ruolo fondamentale durante l'attività di progettazione, consentendo di raggiungere una maggior leggibilità e manutenibilità del codice, permettendo di usufruire a pieno dei vantaggi offerti da *TypeScript*.
I tipi e le interfacce verranno presentati in ordine alfabetico.

3.3.1 ChartBaseVisualizationOptions

Interfaccia che definisce i tipi dei dati relativi alle opzioni di visualizzazione comuni a tutti i `Chart`.

```
export interface ChartBaseVisualizationOptions {
      globalSeriesType: string;
      sortX: boolean;
      sortY?: boolean;
      legend: {
      enabled: boolean;
      placement: string;
      traceorder: 'grouped'|'normal'|'reversed'|'reversed+grouped';
      };
      xAxis: {
      labels: {
            enabled: boolean;
      };
      type: string;
      title?: {
            text: string;
      };
      };
      yAxis: [
      {
            type: string;
            rangeMax?: number;
            rangeMin?: number;
            title?: {
            text?: string;
            };
      },
      {
            opposite: boolean;
            type: string;
      },
      ];
      alignYAxesAtZero: boolean;
      error_y: {
      type: string;
      visible: boolean;
      };
      series: {
      error_y: {
            type: string;
            visible: boolean;
      };
```

```
    stacking: string;
    percentValues?: boolean;
    };
    seriesOptions: {
    [x: string]: any;
    };
    valuesOptions: {
    [x: string]: any;
    };
    columnMapping: { [x: string]: string };
    direction: {
    type: string;
    };
    sizemode: string;
    coefficient: number;
    showDataLabels: boolean;
    numberFormat: string;
    dateTimeFormat: string;
    percentFormat: string;
    textFormat: string;
    missingValuesAsZero: boolean;
    onHover?: () => void;
    onUnHover?: () => void;
    reverseX?: boolean;
    reverseY?: boolean;
    showpoints?: boolean;
    heatMinColor?: string;
    heatMaxColor?: string;
    colorScheme: string | (string | number)[][];
}
```

Codice 3.1: Definizione dell'interfaccia `ChartBaseVisualizationOptions`

- **globalSeriesType**: stringa che definisce il tipo di *chart* da renderizzare. I valori possibili sono i seguenti: *area, column, box, bubble, heatmap, line, pie* e *scatter*;
- **sortX**: booleano che definisce se ordinare o meno le serie sull'asse delle ascisse;
- **sortY**: booleano che definisce se ordinare o meno le serie sull'asse delle ordinate;
- **legend**: oggetto che definisce le opzioni della legenda del grafico:
  - **enabled**: booleano che definisce se abilitare o meno la legenda;

  - **placement**: stringa che definisce la posizione della legenda all'interno del grafico;
  - **traceorder**: stringa che definisce l'ordine delle tracce all'interno del grafico.

- **xAxis**: oggetto che definisce le opzioni dell'asse delle ascisse.
  - **labels**: oggetto che definisce le opzioni delle label dell'asse delle ascisse;
  - **type**: stringa che definisce il tipo di dato visualizzato sull'asse delle ascisse;
  - **title**: oggetto che definisce il titolo dell'asse delle ascisse.

- **yAxis**: array che definisce le opzioni dell'asse delle ordinate. Ogni elemento dell'array è un oggetto che definisce le opzioni per una serie di dati.
  Il primo oggetto definisce i valori per la serie principale:
  - **type**: stringa che definisce il tipo di dato visualizzato sull'asse delle ordinate;
  - **rangeMax**: numero che definisce il valore massimo dell'asse delle ordinate;
  - **rangeMin**: numero che definisce il valore minimo dell'asse delle ordinate;
  - **title**: oggetto che definisce il titolo dell'asse delle ordinate.

  Il secondo oggetto definisce i valori per la serie secondaria:
  - **opposite**: booleano che definisce se la serie secondaria è opposta alla serie principale;
  - **type**: stringa che definisce il tipo di dato visualizzato sull'asse delle ordinate.

- **alignYAxesAtZero**: booleano che definisce se allineare le ordinate a zero;

- **error_y**: oggetto che definisce le opzioni degli errori sull'asse delle ordinate.
  - **type**: stringa che definisce il tipo di errore;
  - **visible**: booleano che definisce se visualizzare o meno gli errori.

- **series**: oggetto che definisce le opzioni delle serie.
  - **error_y**: oggetto che definisce le opzioni degli errori sull'asse delle ordinate;
  - **stacking**: stringa che definisce se le serie sono sovrapposte o impilate;
  - **percentValues**: booleano che definisce se visualizzare i valori in percentuale.

- **seriesOptions**: oggetto che definisce le opzioni delle serie;

- **valuesOptions**: oggetto che definisce le opzioni dei valori;

- **columnMapping**: oggetto che definisce il *mapping* delle colonne;
- **direction**: oggetto che definisce tramite *type* la direzione del grafico;
- **sizemode**: stringa che definisce la modalità di dimensionamento;
- **coefficient**: numero che definisce il coefficiente di dimensionamento;
- **showDataLabels**: booleano che definisce se visualizzare o meno le label dei dati;
- **numberFormat**: stringa che definisce il formato dei numeri;
- **dateTimeFormat**: stringa che definisce il formato della data;
- **percentFormat**: stringa che definisce il formato percentuale;
- **textFormat**: stringa che definisce il formato del testo;
- **missingValuesAsZero**: booleano che definisce se i valori mancanti devono essere considerati come zero;
- **onHover**: funzione che definisce l'azione da eseguire al passaggio del mouse;
- **onUnHover**: funzione che definisce l'azione da eseguire al passaggio del mouse;
- **reverseX**: booleano che definisce se invertire l'asse delle ascisse;
- **reverseY**: booleano che definisce se invertire l'asse delle ordinate;
- **showpoints**: booleano che definisce se visualizzare o meno i punti;
- **heatMinColor**: stringa che definisce il colore minimo del grafico *heatmap*;
- **heatMaxColor**: stringa che definisce il colore massimo del grafico *heatmap*;
- **colorScheme**: stringa o array che definisce lo schema dei colori del grafico *heatmap*.

3.3.2 Column

Interfaccia che definisce i tipi dei dati relativi alle colonne della tabella.

```
export interface Column {
      title: string;
      name: string;
      type: string;
      visible: boolean;
      displayAs: string;
      header?: any;
      alignContent?: string;
```

```
    allowHTML?: boolean;
    allowSearch?: boolean;
    booleanValues?: string[];
    highlightLinks?: boolean;
    imageTitleTemplate?: string;
    imageUrlTemplate?: string;
    imageWidth?: string;
    imageHeight?: string;
    linkOpenInNewTab?: boolean;
    linkTextTemplate?: string;
    linkTitleTemplate?: string;
    linkUrlTemplate?: string;
    numberFormat?: string;
    dateTimeFormat?: string;
    order?: number;
}
```

Codice 3.2: Definizione dell'interfaccia `Column`

- **title**: stringa che definisce il titolo della colonna;
- **name**: stringa che definisce il nome della colonna;
- **type**: stringa che definisce il tipo di dato della colonna;
- **visible**: booleano che definisce se la colonna è visibile o meno;
- **displayAs**: stringa che definisce il tipo di visualizzazione della colonna;
- **header**: oggetto che definisce l'header della colonna;
- **alignContent**: stringa che definisce l'allineamento del contenuto della colonna;
- **allowHTML**: booleano che definisce se permettere o meno l'utilizzo di HTML;
- **allowSearch**: booleano che definisce se permettere o meno la ricerca;
- **booleanValues**: array di stringhe che definisce i valori booleani;
- **highlightLinks**: booleano che definisce se evidenziare i link;
- **imageTitleTemplate**: stringa che definisce il template del titolo dell'immagine;
- **imageUrlTemplate**: stringa che definisce il template dell'URL dell'immagine;
- **imageWidth**: stringa che definisce la larghezza dell'immagine;

- **imageHeight**: stringa che definisce l'altezza dell'immagine;
- **linkOpenInNewTab**: booleano che definisce se aprire il link in una nuova scheda;
- **linkTextTemplate**: stringa che definisce il template del testo del link;
- **linkTitleTemplate**: stringa che definisce il template del titolo del link;
- **linkUrlTemplate**: stringa che definisce il template dell'URL del link;
- **numberFormat**: stringa che definisce il formato numerico;
- **dateTimeFormat**: stringa che definisce il formato della data;
- **order**: numero che definisce l'ordine della colonna.

3.3.3 CounterVisualizationOptions

Interfaccia che definisce i tipi dei dati relativi alle opzioni di visualizzazione del componente `Counter`.

```
export interface CounterBaseVisualizationOptions {
      counterLabel: string;
      counterColName: string;
      targetColName?: string;
      rowNumber: number;
      targetRowNumber: number;
      countRows?: any;
      stringDecimal: number;
      stringDecChar: string;
      stringThouSep: string;
      tooltipFormat: string;
      stringPrefix?: string;
      stringSuffix?: string;
      formatTargetValue?: boolean;
}
```

Codice 3.3: Definizione dell'interfaccia `CounterVisualizationOptions`

- **counterLabel**: stringa che definisce l'etichetta del contatore;
- **counterColName**: stringa che definisce il nome della colonna contenente i valori del contatore;
- **targetColName**: stringa che definisce il nome della colonna contenente il valore target del contatore;
- **rowNumber**: numero che definisce il numero di righe da considerare per il calcolo del contatore;

- **targetRowNumber**: numero che definisce il numero di righe da considerare per il calcolo del valore target;
- **countRows**: oggetto che definisce le righe da considerare per il calcolo del contatore;
- **stringDecimal**: numero che definisce il numero di cifre decimali da visualizzare;
- **stringDecChar**: stringa che definisce il separatore decimale;
- **stringThouSep**: stringa che definisce il separatore delle migliaia;
- **tooltipFormat**: stringa che definisce il formato del *tooltip*;
- **stringPrefix**: stringa che definisce il prefisso del contatore;
- **stringSuffix**: stringa che definisce il suffisso del contatore;
- **formatTargetValue**: booleano che definisce se formattare o meno il valore target.

3.3.4 DataLabels

Interfaccia che definisce i tipi dei dati relativi alle label dei dati utilizzati negli *heatmaps*.

```
export interface DataLabels {
  x: any[];
  y: any[];
  mode: string;
  hoverinfo: string;
  showlegend: boolean;
  text: string[];
  textfont: {
    color: string[];
  };
}
```

Codice 3.4: Definizione dell'interfaccia `DataLabels`

- **x**: array che definisce i valori sull'asse delle ascisse;
- **y**: array che definisce i valori sull'asse delle ordinate;
- **mode**: stringa che definisce la modalità;
- **hoverinfo**: stringa che definisce le informazioni del *tooltip*;

- **showlegend**: booleano che definisce se visualizzare o meno la legenda;
- **text**: array di stringhe che definisce il testo;
- **textfont**: oggetto che definisce le opzioni del font del testo:
  - **color**: array di stringhe che definisce i colori del font.

3.3.5 PlotlyOptions

Interfaccia che definisce i tipi dei dati relativi alle opzioni di visualizzazione di *Plotly.js*.

```
export interface PlotlyOptions {
  showLink: boolean;
  displaylogo: boolean;
  displayModeBar?: boolean;
  responsive?: boolean;
  autosize?: boolean;
}
```

Codice 3.5: Definizione dell'interfaccia `PlotlyOptions`

- **showLink**: booleano che definisce se visualizzare o meno il link;
- **displaylogo**: booleano che definisce se visualizzare o meno il logo;
- **displayModeBar**: booleano che definisce se visualizzare o meno la barra di *Plotly*;
- **responsive**: booleano che definisce se rendere o meno il grafico responsivo;
- **autosize**: booleano che definisce se adattare o meno la grandezza del grafico.

3.3.6 RendererProps

Interfaccia che definisce i tipi dei dati passati alla componenti `Renderer`, `Counter`, `Chart` e `Table`.

```
export interface RendererProps {
      type: string;
      visualizationName: string;
      data: {
            [x: string]: any;
      };
      options: VisualizationOptions;
}
```

Codice 3.6: Definizione dell'interfaccia `RendererProps`

- **type**: stringa che definisce il tipo componente da renderizzazre. Può assumere i seguenti valori: *COUNTER*, *CHART* e *TABLE*;
- **visualizationName**: stringa che definisce il titolo di visualizzazione della componente;
- **data**: oggetto che contiene i dati da visualizzare all'interno della componente;
- **options**: oggetto che contiene le opzioni di visualizzazione della componente di tipo *VisualizationOptions*.

3.3.7 Series

Interfaccia che definisce i tipi dei dati relativi alle serie rappresentata nel grafico.

```
export interface Series {
  name: string;
  type: string;
  data?: any[];
  visible?: boolean;
  values?: number[];
  labels?: string[];
  hole?: number;
  marker?: {
    colors?: string[];
    color?: string;
    line?: {
      color?: string;
      width?: number;
    };
    size?: number;
    sizemode?: string;
  };
  hoverinfo?: string | boolean;
  text?: any[];
  textinfo?: string;
  textposition?: string;
  textangle?: number;
  textfont?: {
    size?: number;
    color?: string[];
  };
  direction?: string;
  domain?: {
    x: number[];
```

```
    y: number[];
  };
  sourceData?: Map<string, any>;
  hoverlabel?: {
    font?: {
      color?: string[];
    };
    bordercolor?: string;
    bgcolor?: string;
  };
  hover?: any[];
  x?: any;
  y?: any;
  z?: any;
  yaxis?: any;
  orientation?: number;
  insidetextfont?: {
    size?: number;
    color?: string;
  };
  error_y?: any;
  offsetgroup?: string;
  mode?: string;
  fill?: string;
  boxpoints?: string;
  jitter?: number;
  pointpos?: number;
  colorscale?: string | (string | number)[][];
  xgap?: number;
  ygap?: number;
}
```

Codice 3.7: Definizione dell'interfaccia `Series`

- **name**: stringa che definisce il nome della serie;
- **type**: stringa che definisce il tipo di serie;
- **data**: array che definisce i dati della serie;
- **visible**: booleano che definisce se la serie è visibile o meno;
- **values**: array di numeri che definisce i valori della serie;

- **labels**: array di stringhe che definisce le label della serie;
- **hole**: numero che definisce, nel caso di grafico a torta, il raggio del buco centrale;
- **marker**: oggetto che definisce le opzioni del *marker*;
- **hoverinfo**: stringa o booleano che definisce le informazioni del *tooltip*;
- **text**: array che definisce il testo della serie;
- **textinfo**: stringa che definisce le informazioni del testo;
- **textposition**: stringa che definisce la posizione del testo;
- **textangle**: numero che definisce l'angolo del testo;
- **textfont**: oggetto che definisce le opzioni del font del testo;
- **direction**: stringa che definisce la direzione della serie;
- **domain**: oggetto che definisce il dominio della serie;
- **sourceData**: mappa che definisce i dati sorgente;
- **hoverlabel**: oggetto che definisce le opzioni del *tooltip*;
- **hover**: array che definisce le opzioni del *tooltip*;
- **x**: array che definisce i valori sull'asse delle ascisse;
- **y**: array che definisce i valori sull'asse delle ordinate;
- **z**: array che definisce i valori sull'asse z;
- **yaxis**: array che definisce i valori sull'asse delle ordinate;
- **orientation**: numero che definisce l'orientamento;
- **insidetextfont**: oggetto che definisce le opzioni del font interno;
- **error_y**: oggetto che definisce le opzioni degli errori sull'asse delle ordinate;
- **offsetgroup**: stringa che definisce il gruppo di *offset*;
- **mode**: stringa che definisce la modalità;
- **fill**: stringa che definisce il riempimento;
- **boxpoints**: stringa che definisce i punti del *box*;
- **jitter**: numero che definisce il *jitter* (variazione);
- **pointpos**: numero che definisce la posizione del punto;
- **colorscale**: stringa o array che definisce lo schema dei colori;

- **xgap**: numero che definisce il gap sull'asse delle ascisse;
- **ygap**: numero che definisce il gap sull'asse delle ordinate;
- **colors**: array di stringhe che definisce i colori del *marker*;
- **color**: stringa che definisce il colore del *marker*;
- **line**: oggetto che definisce le opzioni della linea del *marker*;
- **size**: numero che definisce la grandezza del *marker*;
- **sizemode**: stringa che definisce la modalità di dimensionamento del *marker*.
- **font**: oggetto che definisce le opzioni del font del *marker*;
- **bordercolor**: stringa che definisce il colore del bordo del *marker*;
- **bgcolor**: stringa che definisce il colore di sfondo del *marker*;
- **colors**: array di stringhe che definisce i colori delle rappresentazioni (multiple);
- **size**: numero che definisce la grandezza del font;
- **color**: stringa che definisce il colore della rappresentazione (singola).

3.3.8 TableBaseVisualizationOptions

Interfaccia che definisce i tipi dei dati relativi alle opzioni di visualizzazione del componente `Table`.

```
export interface TableBaseVisualizationOptions {
      columns?: Column[];
      itemsPerPage?: number;
      paginationSize?: string;
}
```

Codice 3.8: Definizione dell'interfaccia `TableBaseVisualizationOptions`

- **columns**: array di oggetti che definiscono le colonne della tabella;
- **itemsPerPage**: numero che definisce il numero di elementi per pagina;
- **paginationSize**: stringa che definisce la grandezza della paginazione.

3.3.9 VisualizationOptions

Tipo che definisce le opzioni di visualizzazione delle componenti.

```
export type VisualizationOptions =
      | CounterBaseVisualizationOptions
      | ChartBaseVisualizationOptions
      | TableBaseVisualizationOptions;
```

Codice 3.9: Definizione del tipo `VisualizationOptions`

Questa definizione permette di definire le opzioni di visualizzazione specifiche per ciascun tipo di componente, garantendo una maggiore flessibilità nella definizione di `RendererProps`.

3.3.10 VisualizationSettings

Interfaccia che definisce i tipi dei dati relativi alle impostazioni di visualizzazione delle componenti.

```
interface VisualizationSettings {
      dateFormat: string;
      dateTimeFormat: string;
      integerFormat: string;
      floatFormat: string;
      booleanValues: string[];
      tableCellMaxJSONSize: number;
      allowCustomJSVisualizations: boolean;
      hidePlotlyModeBar: boolean;
      choroplethAvailableMaps: any;
}
```

Codice 3.10: Definizione dell'interfaccia `VisualizationSettings`

- **dateFormat**: stringa che definisce il formato della data;
- **dateTimeFormat**: stringa che definisce il formato della data e dell'ora;
- **integerFormat**: stringa che definisce il formato degli interi;
- **floatFormat**: stringa che definisce il formato dei numeri decimali;
- **booleanValues**: array di stringhe che definisce i valori booleani;
- **tableCellMaxJSONSize**: numero che definisce la grandezza massima delle celle della tabella;
- **allowCustomJSVisualizations**: booleano che definisce se permettere o meno le visualizzazioni personalizzate;
- **hidePlotlyModeBar**: booleano che definisce se nascondere o meno la barra di Plotly;
- **choroplethAvailableMaps**: oggetto che definisce le mappe disponibili per il choropleth.

3.3.11 VisualizationType

Enumerazione che definisce i possibili valori dei tipi di visualizzazione supportati dalla libreria grafica.

```
export enum VisualizationType {
      Counter = 'COUNTER',
      Chart = 'CHART',
      Table = 'TABLE',
      None = 'NONE',
}
```

Codice 3.11: Definizione dell'enumerazione `VisualizationType`

I possibili valori dell'enumerazione sono i seguenti:

- **Counter**: componente che visualizza un contatore;
- **Chart**: componente che visualizza un grafico;
- **Table**: componente che visualizza una tabella;
- **None**: valore di default, utilizzato per la gestione di errori o situazioni non previste.

3.4 Progettazione delle componenti

Nella seguente sezione verranno presentate le componenti individuate durante l'attività di progettazione, con una descrizione dettagliata delle funzionalità offerte e delle possibili ottimizzazioni individuate.
L'attività di progettazione è avvenuta seguendo il *single responsability principle*, invididuando componenti operabili all'interno del design pattern *Component Composition* proprio di *React*, il quale prevede la combinazione di composizioni di componenti più piccole per ottenere componenti più complesse, favorendo il riutilizzo, la modularità e la manutenibilità del codice.
Le componenti verranno presentate in ordine alfabetico, secondo la seguente struttura:

- **Nome della componente**: breve descrizione della componente;
- **Descrizione**: descrizione dettagliata delle funzionalità offerte dalla componente;
- **Props**: definizione delle props utilizzate dalla componente;
- **Ottimizzazioni**: possibili ottimizzazioni individuate per la componente.

3.4.1 Counter

- **Nome della componente**: `Counter`;
- **Descrizione**: La componente `Counter` prevede la visualizzazione di un contatore, il cui valore viene definito a partire dai dati passati come *props* tra le varie componenti e forniti dalla risposta in formato *JSON* ricevuta dai *server Datasoil*.
  La componente permette inoltre di confrontare il valore attuale del contatore con un valore target (opzionale) da raggiungere, anch'esso ricavato dalla risposta in formato *JSON* ricevuta dal *server*.
  Il valore del contatore può essere formattato in base alle preferenze definite lato backend, presentando le seguenti opzioni:
  - Prefisso: stringa da aggiungere prima del valore del contatore;
  - Suffisso: stringa da aggiungere dopo il valore del contatore;
  - Numero di cifre decimali: numero di cifre decimali da visualizzare nel valore del contatore;
  - Separatore delle migliaia: carattere da utilizzare come separatore delle migliaia;
  - Separatore decimale: carattere da utilizzare come separatore decimale.

  (Queste formattazioni sono applicabili anche al valore target e ai *tooltip* visualizzati al passaggio del mouse sul contatore).
  L'implementazione delle formattazioni numeriche prevede l'utilizzo della libreria *numbro.js*.
- **Props**: I *props* della componente `Counter` sono definiti dall'interfaccia `RendererProps`, su cui il componente ne effettua il *picking* (ovvero la selezione) delle informazioni *data, options* e *visualizationName*.

```
export function Counter({
      data,
      options,
      visualizationName,
      }: Pick<RendererProps,
            'data'
            | 'options'
            | 'visualizationName'>);
```

Codice 3.12: Definizione delle *props* della componente `Counter`

- **Ottimizzazioni**:
  - Utilizzo della libreria *numbro.js* a favore della libreria *numeral.js* precedentemente utilizzata, più pesante e non più mantenuta, per la formattazione delle label numeriche all'interno del `Counter`;

  - La progettazione della presente componente prevede l'utilizzo di *hooks* per la gestione del suo stato interno e per il calcolo delle dimensioni del contatore, necessarie per permettere la visualizzazione responsiva della componente: l'utilizzo di tali *hooks* permette di garantire un'implementazione efficiente e performante, riducendo il numero di renderizzazioni e computazioni non necessarie.

3.4.2 Chart

- **Nome della componente**: `Chart`;
- **Descrizione**: La componente `Chart` prevede la visualizzazione di un grafico, il cui tipo e i dati da visualizzare vengono definiti a partire dai dati passati come *props* tra le varie componenti e forniti dalla risposta in formato *JSON* ricevuta dai *server Datasoil*.
  I tipi di grafici che tale componente permette di renderizzare sono i seguenti:
  - *Area Chart*: grafico ad area;
  - *Bar Chart*: grafico a barre;
  - *Box Plot Chart*: grafico a scatola;
  - *Bubble Chart*: grafico a bolle;
  - *HeatMap Chart*: grafico a matrice;
  - *Line Chart*: grafico a linee;
  - *Pie Chart*: grafico a torta;
  - *Scatter Chart*: grafico a dispersione.

  La componente permette inoltre, a seconda del tipo di grafico renderizzato, di:
  - Effettuare il download dell'immagine del grafico in formato *.png*;
  - Effettuare lo zoom sul grafico;
  - Effettuare lo zoom-in sul grafico;
  - Effettuare lo zoom-out sul grafico;
  - Effettuare il pan sul grafico;
  - Effettuare l'autoscale sul grafico;
  - Resettare la dimensione degli assi del grafico.

  A fronte di tali funzionalità, l'implementazione della componente prevede l'utilizzo della libreria *Plotly.js*, la quale permette di renderizzare grafici in modo reattivo.
  Le evenutali label applicate ai grafici possono essere formattate in base alle preferenze definite lato backend, presentando le seguenti opzioni.

  - Label numeriche:
    * Numero di cifre decimali: numero di cifre decimali da visualizzare nel valore della label;
    * Separatore delle migliaia: carattere da utilizzare come separatore delle migliaia;
    * Separatore decimale: carattere da utilizzare come separatore decimale;
    * suffissi: stringa da aggiungere dopo il valore della label, ad esempio '%'.

    L'implementazione delle formattazioni numeriche prevede l'utilizzo della libreria *numbro.js*.

  - Label temporali:
    * Formato della data: formato da utilizzare per la visualizzazione della data;
    * Formato dell'ora: formato da utilizzare per la visualizzazione dell'ora.

    L'implementazione delle formattazioni temporali prevede l'utilizzo della liberia *day.js*.

- **Props**: I *props* della componente `Chart` sono definiti dall'interfaccia `RendererProps`, su cui il componente ne effettua il *picking* delle informazioni *data* e *options*.

```
export function Chart({
      data,
      options,
      }: Pick<RendererProps,
            'data'
            | 'options'>);
```

Codice 3.13: Definizione delle *props* della componente `Chart`

- **Ottimizzazioni**:
  - Utilizzo di un *custom bundle* di *Plotly.js* in modo da ridurre le dimensioni del *bundle* finale, evitando la registrazione di grafici non utilizzati all'interno delle dashboard *Datasoil S.r.l.* Tale ottimizzazione permette inoltre di evitare la registrazione manuale delle *traces*, necessaria nelle versioni rese disponibili da *Plotly.js* a causa di un bug noto non risolto della libreria.
  - Utilizzo della libreria *numbro.js* a favore della libreria *numeral.js* precedentemente utilizzata, più pesante e non più mantenuta, per la formattazione delle label numeriche all'interno dei grafici;
  - Utilizzo della libreria *day.js* a favore della libreria *moment.js* precedentemente utilizzata, in quanto costituisce una *peerdependency* all'interno dei prodotti *Datasoil S.r.l.*, riducendo così le dimensioni del *bundle* finale;

  - Utilizzo di un *custom hooks* per la gestione dello stato interno della componente e per il calcolo delle dimensioni del grafico, necessarie per permettere la visualizzazione responsiva della componente. L'utilizzo dell'*hooks* è volto a garantire un'implementazione efficiente e performante della componente.

3.4.3 Table

- **Nome della componente**: `Table`;
- **Descrizione**: La componente `Table` prevede la visualizzazione di una tabella, i cui dati e la cui struttura (intesa come colonne e tipo di dato visualizzato all'interno di esse) vengono definiti a partire dai dati passati come *props* tra le varie componenti e forniti dalla risposta in formato *JSON* ricevuta dai *server Datasoil*.
  La componente permette inoltre di:
  - Ordinare le colonne della tabella;
  - Filtrare le *entry* della tabella grazie ad un filtro globale;
  - Effettuare il download della tabella in formato *.csv*;

  A fronte di tali funzionalità, l'implementazione della componente prevede l'utilizzo della componente fornita dalla libreria *PrimeReact*.
- **Props**: I *props* della componente `Table` sono definiti dall'interfaccia `RendererProps`, su cui il componente ne effettua il *picking* delle informazioni *data, options* e *visualizationName*.

```
export function Table({
      data,
      options,
      visualizationName,
      }: Pick<RendererProps,
            'data'
            | 'options'
            | 'visualizationName'>);
```

Codice 3.14: Definizione delle *props* della componente `Table`

- **Ottimizzazioni**:
  - Utilizzo della componente `DataTable` fornita dalla libreria *PrimeReact* a favore della componente `Table` di *AntD* precedentemente utilizzata, in quanto la libreria *PrimeReact* offre una maggiore flessibilità e personalizzazione delle tabelle, oltre che a costituire una *peerdependency* all'interno dei prodotti *Datasoil S.r.l.*, riducendo così le dimensioni del *bundle* finale;

- Utilizzo di *hook* per la gestione del filtro globale, in modo da garantire un'implementazione efficiente e performante della componente, a differenza dell'implementazione fornita da *PrimeReact* che comporta un bug noto non risolto.

3.4.4 Renderer

- **Nome della componente**: `Renderer`;
- **Descrizione**: La componente `Renderer` seleziona dinamicamente un sottocomponente tra quelli resi disponibili dalla libreria *viz-lib* in base al tipo di visualizzazione definito nei dati passati come *props*, costituendo l'*entry point* della libreria.

- **Props**: I *props* della componente `Renderer` sono definiti dall'interfaccia `RendererProps`, su cui il componente ne effettua il *picking* delle informazioni *data, options* e *visualizationName*.

```
export function Renderer({
      data,
      options,
      visualizationName,
      }: Pick<RendererProps,
            'data'
            | 'options'
            | 'visualizationName'>);
```

Codice 3.15: Definizione delle *props* della componente `Renderer`

- **Ottimizzazioni**:
  - La componente prevede l'utilizzo di *hooks* per ottimizzare le prestazioni, rendendo il componente memoizzato$_G$, prevenendo i rendering non necessari, garantendo così un'implementazione efficiente e performante.

3.4.5 VisualizationWidgetHeader

- **Nome della componente**: `VisualizationWidgetHeader`;
- **Descrizione**: La componente `VisualizationWidgetHeader` appartiene alla libreria *dashboard*, appartenente sempre al SDK prodotto. Questa componente costituisce l'*header* del *container* per la visualizzazione delle componenti da renderizzare all'interno della dashboard.

- **Props**: I *props* della componente `VisualizationWidgetHeader` sono definiti delle informazioni *name, visualization* e *dataDialog*.

```
function VisualizationWidgetHeader({
      name,
      visualization,
      dataDialog,
      }: {
      name: string;
      visualization: Visualization;
      dataDialog: WidgetData | undefined;
      });
```

Codice 3.16: Definizione delle *props* della componente `VisualizationWidgetHeader`

- **Ottimizzazioni**:
  - Ridefinizione dei *props* della componente in modo da permettere la visualizzazione di un *dialog* contenente la componente visualizzata all'interno del *container*;
  - Ridefinizione della componente in modo da permettere la visualizzazione di un *dialog* contenente la componente visualizzata all'interno del *container*, attraverso l'utilizzo di componenti fornite dalla libreria *PrimeReact*.

Capitolo 4

Realizzazione e testing

Nella seguente sezione verranno presentate le attività di realizzazione e testing effettuate durante lo sviluppo della libreria, con l'obiettivo di fornire una panoramica sulle tecnologie utilizzate, sulle funzionalità implementate e sui test effettuati per garantire la qualità del prodotto sviluppato.

4.1 Strumenti utilizzati

La seguente sezione fornisce gli strumenti utilizzati durante lo svolgimento dello stage per la realizzazione della libreria grafica. Gli strumenti verranno presentati in ordine alfabetico secondo la seguente struttura:

- **Nome strumento**: nome dello strumento utilizzato;
- **Versione**: versione utilizzata nel progetto durante lo stage;
- **Link**: link di riferimento per ulteriori informazioni sullo strumento;
- **Descrizione**: breve descrizione dello strumento e delle sue funzionalità;
- **Vantaggi**: principali vantaggi derivanti dall'utilizzo dello strumento; *(Opzionale)*
- **Svantaggi**: principali svantaggi derivanti dall'utilizzo dello strumento; *(Opzionale)*
- **Alternative Esaminate**: alternative studiate e prese in considerazione durante la scelta dello strumento, con una breve considerazione su di esse e la motivazione per la quale non sono state selezionate. *(Opzionale)*

4.1.1 Tecnologie frontend

4.1.1.1 D3-color

- **Nome strumento**: D3-color

- **Versione**: 3.1.0
- **Link**: d3js.org/d3-color
- **Descrizione**: D3-color è una libreria JavaScript open-source utilizzata per la manipolazione dei colori.
- **Vantaggi**:
  - *D3-color* offre un'ampia serie di funzionalità per la manipolazione dei colori, quali la conversione tra spazi di colori, la manipolazione dei colori e la generazione di scale di colori;
  - *D3-color* è supportato da una vasta e attiva *community*.
- *Alternative Esaminate*:
  - *Chroma-js*: *Chroma-js* è una libreria *JavaScript open-source* utilizzata per la manipolazione dei colori, ma risulta essere più pesante di *D3-color*.

4.1.1.2 D3-scale

- **Nome strumento**: D3-scale
- **Versione**: 4.0.2
- **Link**: d3js.org/d3-scale
- **Descrizione**: *D3-scale* è una libreria *JavaScript* open-source utilizzata per la generazione di scale di colori, di posizioni e di dimensioni.
- **Vantaggi**:
  - *D3-scale* offre un'ampia serie di funzionalità per la generazione di scale di colori, di posizioni e di dimensioni, quali la generazione di scale lineari, logaritmiche e ordinali;
  - *D3-scale* è supportato da una vasta e attiva *community*.

4.1.1.3 Day.js

- **Nome strumento**: Dayjs
- **Versione**: 1.11.7
- **Link**: day.js.org
- **Descrizione**: *Day.js* è una libreria *JavaScript* open-source utilizzata per la manipolazione delle date e degli orari.
- **Vantaggi**:

  - *Day.js* è una libreria molto leggera, con un'ampia serie di funzionalità per la manipolazione delle date e degli orari;
  - *Day.js* viene fornito con dichiarazioni ufficiali di tipo per *TypeScript*;
  - *Day.js* è tutt'ora supportato da una vasta e attiva *community*.

- **Alternative Esaminate**:

  - *Moment.js*: *Moment.js* è una libreria *JavaScript* open-source utilizzata per la manipolazione delle date e degli orari, ma risulta essere più pesante di *Day.js*. *Moment.js* è inoltre considerato *deprecated* a favore di Day.js, che offre una maggiore leggerezza e una maggiore efficienza.

4.1.1.4 Lodash

- **Nome strumento**: Lodash
- **Versione**: 4.14.0
- **Link**: lodash.com
- **Descrizione**: *Lodash* è una libreria *JavaScript open-source* utilizzata per la manipolazione di oggetti e array.
- **Vantaggi**:

  - *Lodash* offre un'ampia serie di funzionalità implementate con efficienza per la manipolazione di oggetti e array, quali la ricerca, la modifica e la rimozione di elementi;
  - *Lodash* è supportato da una vasta e attiva *community*.

- **Svantaggi**

  - *Lodash* è una libreria molto pesante, con un'ampia serie di funzionalità che possono non essere utilizzate all'interno del progetto: per questo motivo in questo progetto sono eseguiti degli `import` sui moduli specifici, riducendo così le dimensioni del *package*$_G$ finale.

4.1.1.5 Numbro

- **Nome strumento**: Numbro
- **Versione**: 2.5.0
- **Link**: numbrojs.com
- **Descrizione**: *Numbro* è una libreria *JavaScript* open-source utilizzata per la formattazione dei numeri.
- **Vantaggi**:

  - *Numbro* offre un'ampia serie di funzionalità per la formattazione dei numeri, quali la formattazione delle cifre decimali, la formattazione delle valute e la formattazione dei numeri in notazione scientifica;
  - *Numbro* è supportato da una vasta e attiva *community*.

- **Alternative Esaminate**:
  - *Numeral.js*: libreria *JavaScript* open-source utilizzata per la formattazione dei numeri, ma risulta non essere più mantenuta attivamente dalla *community*, oltre che ad essere più pesante di *Numbro*.

4.1.1.6 Plotly.js

- **Nome strumento**: Plotly.js
- **Versione**: custom-bundle: 2.33.0
- **Link**: plotly.com/javascript
- **Descrizione**: *Plotly.js* è una libreria *JavaScript open-source*, con supporto per *TypeScript*, utilizzata per la visualizzazione di dati mediante grafici interattivi. Costruito sopra *D3.js*, *Plotly.js* offre un vasto panorama di grafici dinamici in formato *Scalable Vector Graphics (SVG)*$_G$, altamente personalizzabili.
- **Vantaggi**:
  - *Plotly.js* offre funzionalità di interattività avanzate, quali zoom, pan, selezione e salvataggio dei grafici;
  - Le disponibilità di grafici offerte da *Plotly.js* sono molto ampie, permettendo di soddisfare la maggior parte delle esigenze all'interno della libreria grafica;
  - *Plotly.js* è supportato da una vasta e attiva *community*, con ampie serie di esempi disponibili su *CodePen* (codepen.io, una piattaforma di condivisione di codice *online* che permette di visualizzare e modificare codice *HTML, CSS e JavaScript*);
  - *Plotly.js* offre la possibilità di generare *custom bundle* personalizzati, permettendo di registrare le sole *traces* che si vogliono utilizzare, riducendo così le dimensioni del pacchetto finale;
  - *Plotly.js* era già precedentemente utilizzato all'interno della libreria preesistente, non necessitando così di modifiche lato *backend*$_G$ negli editor utilizzati per la generazione delle risposte *JSON* da parte dei server *Datasoil S.r.l.* per la generazione delle dashboard;
  - *Plotly.js* offre una funzionalità, *Plotly.react*, che permette di aggiornare i grafici in modo efficiente, riducendo il tempo di rendering e migliorando le prestazioni complessive della libreria.

- **Svantaggi**:
  - *Plotly.js* è una libreria molto pesante in termini sia di spazio che di rallentamenti *runtime*$_G$, dovuto anche dal fatto che è implementata sopra un *wrapper* proprietario di *D3.js*, impedendo l'ottimizzazione di alcune dipendenze non necessarie in quanto non utilizzate;
  - La documentazione ufficiale di *Plotly.js* è vaga e poco esaustiva;
  - La versione ufficiale di *Plotly.js* presenta degli errori durante la registrazione delle *traces*, impedendo di importare correttamente *Plotly* all'interno dei moduli *TypeScript*, motivo per il quale è stata utilizzata una versione *custom bundle* di *Plotly.js*.
- **Alternative Esaminate**:
  - *D3.js*: *D3.js* è una libreria *JavaScript open-source* utilizzata per la generazione di grafici dinamici e manipolazione dati, la quale dalla sua parte risulta però essere più complessa rispetto a *Plotly.js*, richiedendo una maggiore curva di apprendimento e una maggiore quantità di codice per la generazione di grafici.
  - *Chart.js*: *Chart.js* è una libreria *JavaScript open-source* utilizzata per la generazione di grafici dinamici, la quale risulta essere più leggera di *Plotly.js*, ma offre una minore quantità di grafici disponibili; il suo utilizzo avrebbe inoltre comportato la richiesta di modifiche lato *backend*.

4.1.1.7 PrimeFlex

- **Nome strumento**: PrimeFlex
- **Versione**: 3.3.0
- **Link**: primefaces.org/primeflex
- **Descrizione**: *PrimeFlex* è una libreria *CSS open-source* utilizzata per la creazione di *layout* flessibili e *responsive*$_G$.
- **Vantaggi**:
  - *PrimeFlex* offre un'ampia serie di classi *CSS* per la creazione di *layout* flessibili e responsivi, permettendo di adattare il *layout* in base alla grandezza dello schermo;
  - *PrimeFlex* è una libreria molto leggera;
  - *PrimeFlex* è fortemente integrata con le componenti di *PrimeReact*;
  - *PrimeFlex* può essere integrato a *tailwind*;
  - *PrimeFlex* è supportato da una vasta e attiva *community*.

4.1.1.8 PrimeIcons

- **Nome strumento**: PrimeIcons
- **Versione**: 6.0.1
- **Link**: primefaces.org/primeicons
- **Descrizione**: *PrimeIcons* è una libreria di icone *open-source* utilizzata per la creazione di interfacce utente.
- **Vantaggi**:
  - *PrimeIcons* offre un'ampia serie di icone per la creazione di interfacce utente, le quali sono altamente personalizzabili e facilmente integrabili;
  - *PrimeIcons* è una libreria molto leggera, permettendo di creare interfacce utente performanti e veloci;
  - *PrimeIcons* è fortemente integrata con le componenti di *PrimeReact*;
  - *PrimeIcons* è supportato da una vasta e attiva community.

4.1.1.9 PrimeReact

- **Nome strumento**: PrimeReact
- **Versione**: 10.0.0
- **Link**: primefaces.org/primereact
- **Descrizione**: *PrimeReact* è una libreria di componenti *React open-source* utilizzata per la creazione di interfacce utente.
- **Vantaggi**:
  - *PrimeReact* offre un'ampia serie di componenti *React* per la creazione di interfacce utente, le quali a loro volta sono altamente personalizzabili, con numerose *feature* integrate;
  - *PrimeReact* è una libreria molto leggera, permettendo di creare interfacce utente performanti e veloci;
  - *PrimeReact* costituisce una *peer-dependency*$_G$ all'interno dei prodotti *Datasoil S.r.l.*, permettendo di utilizzare le componenti *PrimeReact* all'interno della libreria grafica senza comportare l'aggiunta di dipendenze esterne, riducendo così le dimensioni del *bundle* finale;
  - *PrimeReact* è supportato da una vasta e attiva *community*.
- **Alternative Esaminate**:
  - *Ant Design*: *Ant Design* è una libreria di componenti *React open-source* utilizzata per la creazione di interfacce utente, la quale non risulta però essere integrata all'interno dei prodotti *Datasoil S.r.l.*

4.1.1.10 React

- **Nome strumento**: React
- **Versione**: 18.0.0
- **Link**: reactjs.org
- **Descrizione**: *React* è una libreria *React open-source* per la creazione di interfacce utente, sviluppata da Facebook.
- **Vantaggi**:
  - *React* offre un'ampia serie di funzionalità per la creazione di interfacce utente, quali il *Virtual DOM*, il *JSX* e *Hooks*;
  - *React* è supportato da una vasta e attiva *community*;
  - *React* è una libreria molto leggera, permettendo di creare interfacce utente performanti e veloci.

4.1.1.11 TypeScript

- **Nome strumento**: TypeScript
- **Versione**: 5.1.6
- **Link**: typescriptlang.org
- **Descrizione**: *TypeScript* è un linguaggio di programmazione open-source sviluppato da *Microsoft*, che fa da *super-set*$_G$ a *JavaScript*.
- **Vantaggi**:
  - *TypeScript* permette di definire tipi ed interfacce per le variabili e i parametri delle funzioni, garantendo una maggiore sicurezza e affidabilità del codice;
  - *TypeScript* permette di effettuare controlli statici sul codice sorgente, riducendo il numero di errori a *runtime*.

4.1.1.12 usehook-ts

- **Nome strumento**: usehook-ts
- **Versione**: 3.1.0
- **Link**: usehooks-ts.com
- **Descrizione**: *usehook-ts* è una libreria *open-source* di *custom hook React*.
- **Vantaggi**:

  - *usehook-ts* offre un'ampia serie di *hook* personalizzati per la creazione di interfacce utente;
  - *usehook-ts* è supportato da una vasta e attiva *community*.

4.1.2 Strumenti per lo sviluppo

4.1.2.1 ESLint

- **Nome strumento**: ESLint
- **Versione**: 8.3.0
- **Link**: eslint.org
- **Descrizione**: *ESLint* è uno strumento *open-source* utilizzato per l'analisi statica del codice sorgente, per identificare pattern problematici o codice che non rispetta le linee guida definite all'interno del progetto.
- **Vantaggi**:
  - *ESLint* permette di definire regole personalizzate per l'analisi del codice sorgente, garantendo la coerenza e la qualità del codice;
  - *ESLint* permette di integrarsi con gli strumenti di *build*, permettendo di eseguire l'analisi statica del codice sorgente durante il processo di *build*.

4.1.2.2 Jest

- **Nome strumento**: Jest
- **Versione**: 29.0.7
- **Link**: jestjs.io
- **Descrizione**: *Jest* è un *framework* di testing *open-source* utilizzato per il testing di codice *JavaScript* e *TypeScript*.
- **Vantaggi**:
  - *Jest* permette di effettuare test unitari, test di integrazione e test *end-to-end*, garantendo la qualità e la stabilità del codice;
  - *Jest* permette di effettuare test in parallelo, riducendo i tempi di esecuzione dei test;
  - *Jest* permette di generare report dettagliati sui test effettuati, permettendo di identificare e correggere eventuali errori.

4.1.2.3 NVM

- **Nome strumento**: NVM
- **Versione**: 1.1.12
- **Link**: github.com/nvm-sh/nvm
- **Descrizione**: *NVM (Node Version Manager)* è uno strumento *open-source* utilizzato per la gestione delle versioni di *Node.js*.
- **Vantaggi**:
  - *NVM* permette di installare e gestire più versioni di *Node.js* all'interno del sistema, permettendo di selezionare la versione corretta per il progetto in corso;
  - *NVM* permette di gestire le versioni di *Node.js* in modo semplice e veloce, permettendo di passare da una versione all'altra con un solo comando.

4.1.2.4 Prettier

- **Nome strumento**: Prettier
- **Versione**: 3.2.5
- **Link**: prettier.io
- **Descrizione**: *Prettier* è uno strumento *open-source* utilizzato per la formattazione del codice sorgente.
- **Vantaggi**:
  - *Prettier* permette di formattare il codice sorgente in modo automatico, garantendo uno stile uniforme all'interno del progetto.

4.1.2.5 Rollup

- **Nome strumento**: Rollup
- **Versione**: 4.18.0
- **Link**: rollupjs.org
- **Descrizione**: *Rollup* è un *bundler*$_G$ di moduli *JavaScript* che permette di risolvere le dipendenze tra i moduli, generando un unico file di output.
- **Vantaggi**:
  - *Rollup* effettua il *tree-shaking*$_G$, rimuovendo le dipendenze non utilizzate all'interno del codice sorgente;

  - Permette di generare bundle in diversi formati, quali *CommonJS* e *ESM*;
  - *Rollup* è noto per la sua velocità ed efficienza nelle *build*, specialmente per progetti più piccoli o librerie. Questo può risultare in tempi di *build* più rapidi e in una migliore esperienza di sviluppo;
  - *Rollup* ha un sistema di *plug-in* molto potente e flessibile che permette di estendere le sue funzionalità;
  - *Rollup* permette di utilizzare *TypeScript* all'interno del progetto, integrando la configurazione dichiarata nel file *tsconfig.json*;
  - *Rollup* permette di generare file di dichiarazione *TypeScript*.

4.1.2.6 Yarn

- **Nome strumento**: Yarn
- **Versione**: 1.22.22
- **Link**: yarnpkg.com
- **Descrizione**: *Yarn* è un *package manager* per *JavaScript*, sviluppato da *Facebook, Google* e *Tilde*.
- **Vantaggi**:
  - *Yarn* è più veloce di *npm*, ha un sistema di cache più efficiente e permette di installare pacchetti in parallelo.
- **Svantaggi**:
  - *Yarn* presenta un *registry* di dimensione minore rispetto a quello di *npm*.
- **Alternative Esaminate**:
  - *npm*: npm è il *package manager* di default per *Node.js*, ma è più lento di *Yarn* e ha un sistema di cache meno efficiente, offrendo inoltre un output meno comprensibile.

4.1.3 Strumenti per la Collaborazione e la Gestione del Progetto

Nella seguente sezione vengono presentati gli strumenti utilizzati per la collaborazione e la gestione del progetto durante lo svolgimento dello stage.

4.1.3.1 Confluence

- **Nome strumento**: Confluence
- **Versione**: Cloud
- **Link**: atlassian.com/software/confluence

- **Descrizione**: *Confluence* è un software di collaborazione sviluppato da *Atlassian*, utilizzato per la creazione e la gestione della documentazione all'interno di un'azienda.

4.1.3.2 Slack

- **Nome strumento**: Slack
- **Versione**: Cloud
- **Link**: slack.com
- **Descrizione**: *Slack* è un software di collaborazione sviluppato da *Slack Technologies*, utilizzato per la comunicazione quotidiana e la collaborazione con il team.

4.2 Realizzazione delle componenti

4.2.1 Ambiente di sviluppo

Durante lo sviluppo della libreria, è stata utilizzata la versione 18.12.1 di *Node.js*, configurata tramite l'utilizzo di *NVM (Node Version Manager)*, un gestore di versioni di *Node.js* che permette di installare e gestire più versioni in modo semplice e veloce. Per quanto riguarda la gestione delle dipendenze, è stato utilizzato *Yarn*, un *package manager* per *JavaScript* che permette di gestire le dipendenze del progetto in modo efficiente e veloce.
Il versionamento del codice è stato gestito tramite *Git*, un sistema di controllo di versione distribuito, configurando e mantenendo un *repository*$_G$ remoto su *Github* interno all'organizzazione dell'azienda.
All'interno del file *package.json* è stata configurata la sezione *scripts* per definire i comandi necessari per l'esecuzione:

- `start-watcher`: avvia il processo di building del codice di `rollup` in modalità *watch*, in modo da monitorare le modifiche effettuate ai file sorgenti e aggiornare automaticamente il bundle prodotto e utilizzato dall'example all'interno del progetto;
- `start-example`: avvia l'esecuztone dell'esempio all'interno del progetto nel server locale, permettendo di visualizzare il funzionamento della libreria all'interno di un'applicazione di test;
- `build-package`: avvia il procetso di building del codice di *rollup*, generando i bundle finali configurati all'interno del file *rollup.config.js*;
- `publish-package`: avvia il protesso di pubblicazione del pacchetto all'interno del registry privato configurato nel *package.json* dei *packages*, permettendo di distribuire la libreria all'interno dell'organizzazione;

- `test`: avvia il processo di tetting del codice tramite *Jest*, effettuando i test definiti all'interno della *repository*.

```
"scripts": {
    "start-example": "cd packages/example && yarn start",
    "start-watcher": "cd packages/dsdashboard2 && yarn rollup-watch",
    "build-package": "cd packages/dsdashboard2 && yarn build",
    "publish-package": "cd packages/dsdashboard2 && yarn publish"
}
```

Codice 4.1: Scripts del file *package.json* di *dsdashboard2*

```
"scripts": {
    "rollup": "rollup -c --bundleConfigAsCjs",
    "rollup-watch": "rollup -c --bundleConfigAsCjs --watch",
    "clean": "rimraf dist",
    "build": "yarn clean && yarn rollup",
    "test": "jest"
}
```

Codice 4.2: Scripts del file *package.json* dei *packages*

Tutti i precedenti comandi vengono eseguiti tramite il comando *yarn* seguito dal nome dello script definito all'interno del file *package.json*.

4.2.2 Componenti

Nella presente sezione viene descritta l'implementazione delle componenti e delle loro ottimizzazioni all'interno della libreria, con l'obiettivo di fornire una panoramica sulle funzionalità offerte, sulle tecnologie utilizzate e sul flusso di dati all'interno del sistema.

4.2.2.1 Counter

La componente `Counter` costituisce un contatore che permette di visualizzare un valore numerico all'interno di un *widget*, associato a un'etichetta rappresentativa dell'informazione visualizzata.
La componente permette inoltre di visualizzare un target, il valore di riferimento da raggiungere, mettendo a disposizione la componente grafica `Knob` della libreria *PrimeReact*, definendo colori differenti per il valore visualizzato a seconda del raggiungimento o meno del target.
La componente offre inoltre la possibilità di visualizzare dei *tooltip*, fornendo informazioni più dettagliate all'utente in merito ai dati visualizzati, attraverso l'utilizzo dell'attributo `title` associati ai `div` contenenti i valori visualizzati.
Le informazioni utilizzate dalla componente vengono elaborate dalla funzione

`getCounterData`, chiamata all'interno dell'`hook useMemo`, in modo da effettuare il ricalcolo dei dati solo in caso di variazione delle *props* passate alla componente. Questa funzione infatti, a partire dai `data`, dalle `optionsCounter` (opzioni di visualizzazione ottenute attraverso la concatenazazione delle opzioni di default e le *options* passate come *props* al componente) e dal `visualizationName` permette di definite i seguenti valori:

- *CounterValue*: valore numerico del contatore;
- *TargetValue*: valore di riferimento da raggiungere;
- *CounterLabel*: etichetta associata al valore numerico;
- *CounterValueTooltip*: testo del *tooltip* associato al valore del contatore;
- *TargetValueTooltip*: testo del *tooltip* associato al valore di riferimento;
- *TrendPositive*: valore booleano che indica se il valore del contatore ha raggiunto o superato il valore target;
- *ShowTrend*: valore booleano che indica se visualizzare o meno il trend del contatore.

Le *optionsCounter* definiscono le formattazioni da applicare ai valori calcolati (§3.4.1 *Counter: descrizione*), attraverso l'utilizzo della libreria *numbro.js*.
La visualizzazione del trend, determinata dalla variabile `ShowTrend`, viene attivata nel caso vi fosse un target da raggiungere e i valori elaborati fossero finiti, modificando il colore del valore del contatore e aggiungendo un'icona in base al raggiungimento o meno del valore di riferimento.
In merito al rendere la componente responsiva, è stato utilizzato l'*hook* `useResizeObserver`, fornito dalla libreria *usehook-ts*, per permettere di aggiornare il fattore di scala utilizzato per ridimensionare il contenuto in modo da adattarlo alle nuove dimensioni del contenitore attraverso la chiamata alla funzione `onScale`, un *wrapper* di una funzione anonima tramite l'*hook* `useDebounceCallback` (fornito sempre dalla libreria *usehook-ts*) utilizzato per evitare chiamate multiple in rapida successione.
Tale scala, gestita come stato interno della componente mediante l'*hook* `useState`, viene passata come parametro alla funzione `getCounterStyles` invocata nella definizione dello `style` del contenitore dei vari elementi che compongono la componente, la quale ritorna un oggetto di proprietà *CSS* necessarie per applicare la trasformazione di scala nei vari browser.

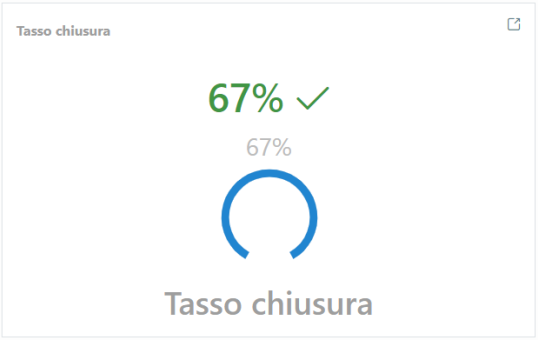


Figura 4.1: Esempio di `Counter` *viz-lib* con target

4.2.2.2 Chart

La componente `Chart` permette di visualizzare i dati passati come *props* alla componente tramite differenti tipologie di grafici.
All'interno del *widget* i *props* `options` vengono elaborati tramite la funzione `getOptions`, la quale si occupa di definire le opzioni di default concatenandole con le opzioni fornite.
In seguito i `data`, le `options` e il `Layout` vengono ulteriormente preparati tramite il *custom hook* `usePlotly`. Questo *hook* si occupa di:

- preparare, a partire dalle *props* `data` e `options`, i dati nel formato richiesto da *Plotly.js*, ovvero sotto forma di un array di oggetti `Series`, ottenuto tramite la funzione `getChartData`, comune a tutte le tipologie di grafici;
- definire attraverso la funzione `prepareData` i dati nel formato richiesto a seconda del tipo di grafico configurato all'interno delle opzioni: questa funzione si occupa di invocare la corretta funzione di preparazione dei dati, wrappata a sua volta all'interno di una funzione specifica per l'aggiornamento delle informazioni;
- aggiornare le dimensioni del *widget* in modo responsivo tramite l'utilizzo dell'hoook `useResizeObserver`, fornito dalla libreria `usehook-ts`, invocando `useDebounceCallback` della medesima libreria per evitare chiamate multiple in rapida successione al controllo di variazione e aggiornamento delle dimensioni del container del grafico;
- preparare, a partire dalle props `options` e dai dati elaborati (comprese le dimensioni del punto precedente), il *layout* del grafico tramite la funzione `prepareLayout`, la quale definisce un oggetto parziale del `Layout` fornito dalla libreria *Plotly.js*. Questa funzione definisce delle opzioni di default ed in seguito si occupa di invocare la funzione opportuna a seconda del grafico configurato;
- aggiornare il `Layout` del grafico in base alle dimensioni del *container*, invocando la funzione `updateChartSize`, la quale si occupa inoltre di stabilire il posizionamento più opportuno per la legenda del grafico a seconda dello spazio disponibile.

- restituire il risultato elaborato costituito dai `plotlyData`, dalle `plotlyOptions` e dal `plotlyLayout` attraverso un `useMemo`, in modo da gestire in modo efficente il ricalcolo dei dati.

Questo *custom hook* permette di gestire le operazioni in modo efficiente, attraverso l'utilizzo di `useEffect` che garantiscono il calcolo e l'aggiornamento delle informazioni solamente nel caso di variazione di dati significativi.
Il risultato restituito dal *custom hook* `usePlotly` viene in seguito utilizzato all'interno di un `useEffect` che lo osserva per effettuare il rendering del grafico tramite una chiamata asincrona alla funzione `plot`, contentente a sua volta la chiamata alla funzione `Plotly.react`, la quale si occupa di renderizzare il grafico all'interno del `div` utilizzato come contenitore e riferito mediante l'utilizzo di un *ref* castato al tipo `PlotlyHTMLElement`, fornito sempre dalla libreria *Plotly.js*.
In seguito vengono configurati i *listeners* dichiarati all'interno delle `options` fornite, permettendo di gestire le interazioni dell'utente di tipo *hover* e *unhover* sul grafico. Tali *listeners* vengono in seguito smontati correttamente all'interno del `return` del `useEffect` per evitare *memory leak* e garantire una corretta gestione della componente.

4.2.2.2.1 Area

Il grafico ad area (*Area*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `AREA`.
La preparazione dei dati avviene in una prima parte invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareAreaSeries`, la quale si occupa di definire le informazioni inerenti alla visualizzazione o meno del testo e la modalità di riempimento dell'area (fino all'asse delle X o fino alla successiva area) a seconda della configurazione della proprietà `fill` definita nelle serie visualizzata.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali ed eventualmente i valori lungo l'asse delle X, ricalcolando i valori cumulativi per ogni tick.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

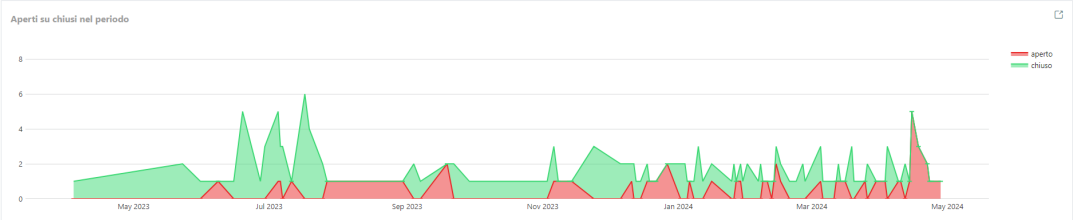


Figura 4.2: Esempio di grafico ad area (*Area*) *viz-lib*

4.2.2.2.2 Bar

Il grafico a barre (*Bar*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `COLUMNS`. La preparazione dei dati avviene in una prima parte invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareBarSeries`, la quale si occupa di definire il tipo (*'bar'*) necessario per *Plotly* per comprendere come interpretare i dati forniti e le informazioni inerenti alla posizione e alla dimensione del testo.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali ed evenutalmente aggiornando i valori lungo l'asse delle X.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

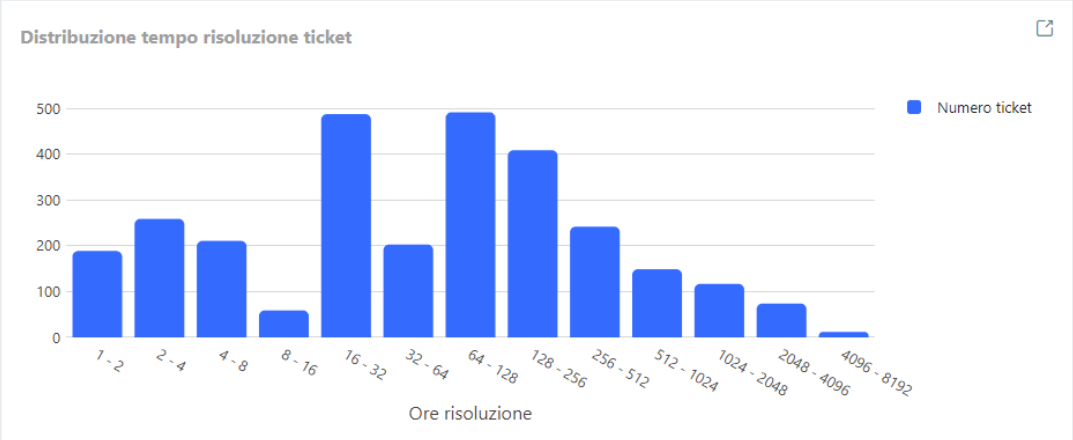


Figura 4.3: Esempio di grafico a colonne (*Bar*) *viz-lib*

4.2.2.2.3 Box

Il grafico a scatole (*Box*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `BOX`.

La preparazione dei dati avviene in una prima parte invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareBoxSeries`, la quale si occupa di definire il tipo (*'box'*) necessario per *Plotly* per comprendere come interpretare i dati forniti e le informazioni inerenti alla visualizzazione dei dati.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali ed evenutalmente aggiornando i valori lungo l'asse delle X.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

4.2.2.2.4 Bubble

Il grafico a bolle (*Bubble*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `BUBBLE`.
La preparazione dei dati avviene in una prima parte invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareBubbleSeries`, la quale si occupa di definire le informazioni inerenti alle bolle, quali le dimensioni di ciascun dato, il coefficiente di ingrandimento e il riferimento sul come interpretare il valore di dimensionamento fornito.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali ed evenutalmente aggiornando i valori lungo l'asse delle X.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

4.2.2.2.5 Heatmap

Il grafico a matrice di calore (*Heatmap*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `HEATMAP`.
La preparazione dei dati avviene invocando la funzione `prepareHeatmapData`, la quale si occupa di definire la scala di colori da utilizzare all'interno del grafico, preparando in seguito le `Series` di dati da visualizzare e le relative `DataLabel` associate, utilizzate

per la visualizzazione delle informazioni durante l'*hover*.
Per quanto riguarda invece la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione.

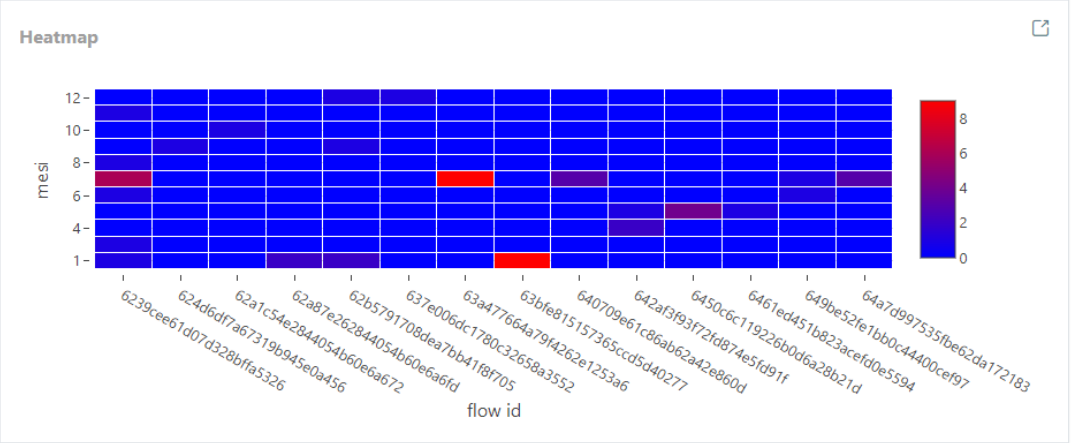


Figura 4.4: Esempio di grafico a matrice di calore (*Heatmap*) *viz-lib*

4.2.2.2.6 Line

Il grafico a linee (*Line*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `LINE`.
La preparazione dei dati avviene in una prima parte invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareLineSeries`, la quale si occupa di definire il tipo (*'line'*) necessario per *Plotly* per comprendere come interpretare i dati forniti e le informazioni inerenti alla posizione e alla dimensione del testo.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali, aggiornando eventualmente i valori lungo l'asse delle X e ricalcolando i valori cumulativi per ogni tick.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

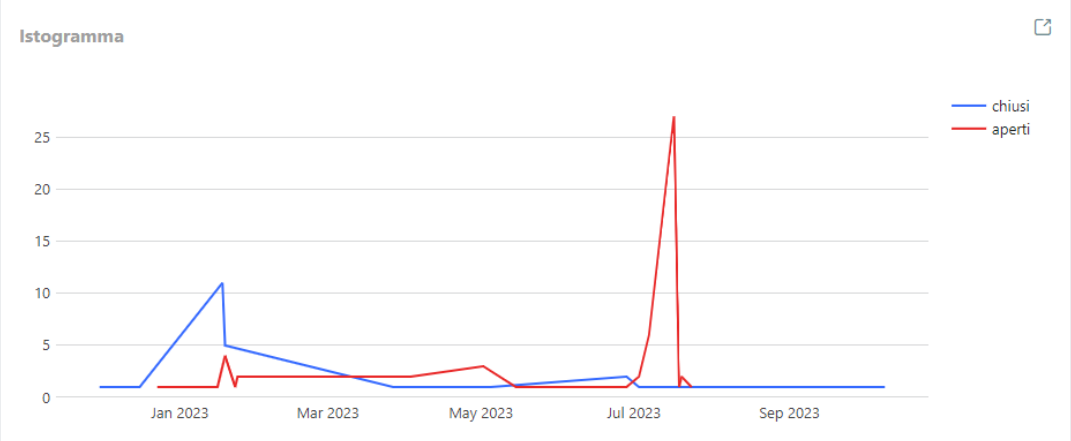


Figura 4.5: Esempio di grafico a linea (*Line*) *viz-lib*

4.2.2.2.7 Pie

Il grafico a torta (*Pie*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `PIE`.
La preparazione dei dati avviene invocando la funzione `preparePieData`, la quale si occupa di definire i colori delle label di testo visualizzate all'interno degli spicchi del grafico: mentre i colori del grafico vengono ricavati dalle opzioni fornite dal backend, i colori delle label vengono determinati a seconda del contrasto maggiore rispetto al colore di sfondo, in modo da aumentare la leggibilità e di conseguenza migliorare l'esperienza utente.
In seguito all'interno della funzione vengono calcolate le percentuali occupate da ciascuna rappresentazione, configurando inoltre i font utilizzati, le dimensioni dei caratteri, le informazioni da visualizzare mediante l'*hover* e la dimensione del raggio dell'*hole* del grafico.
L'aggiornamento dei dati viene gestito mediante la funzione `updatePieData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`.
Per quanto riguarda invece la gestione del `Layout`, viene invocata la funzione `preparePieLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione.

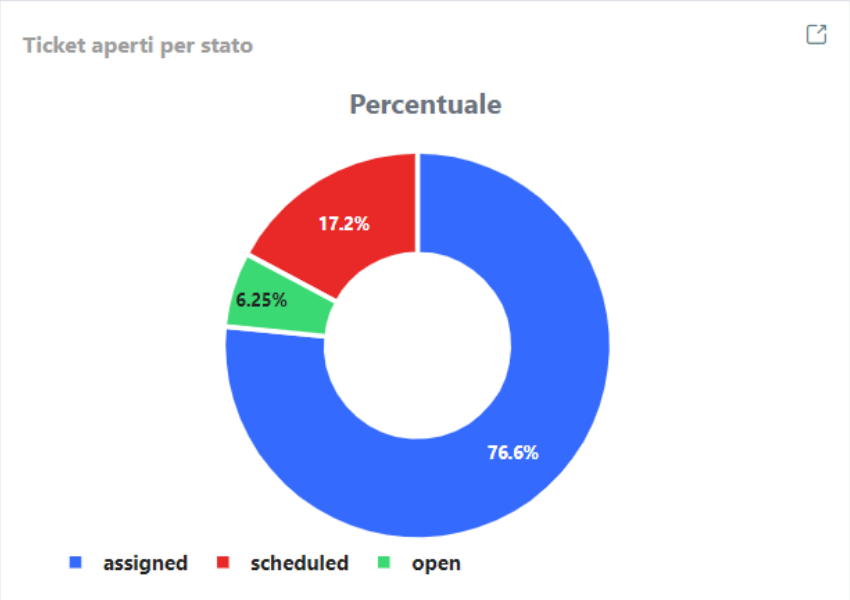


Figura 4.6: Esempio di grafico a torta (*Pie*) *viz-lib*

4.2.2.2.8 Scatter

Il grafico a dispersione (*Scatter*) è una tipologia di grafico visualizzato all'interno della componente `Chart` quando la proprietà `globalSeriesType` è impostata a `SCATTER`.
La preparazione dei dati avviene invocando la funzione `prepareDefaultData`, la quale si occupa di ricavare i valori dei dati da visualizzare e i colori utilizzati, compresi i colori delle label. In seguito, all'interno della stessa funzione, viene invocata la funzione `prepareScatterSeries`, la quale si occupa di definire il tipo (*'scatter'*) necessario per *Plotly* per comprendere come interpretare i dati forniti e le informazioni inerenti alle dimensioni dei punti e alla posizione e dimensione dell'eventuale testo associato.
L'aggiornamento dei dati avviene mediante la funzione `updateDefaultData`, la quale si occupa di aggiornare i testi presenti all'interno delle `Series` visualizzate, invocando la funzione `updateSeriesText`, aggiornando inoltre i valori percentuali ed evenutalmente aggiornando i valori lungo l'asse delle X.
Per quanto riguarda la gestione del `Layout`, viene invocata la funzione `prepareDefaultLayout`, la quale si occupa di definire le informazioni grafiche inerenti alla legenda e al titolo, come posizione, font, colore e dimensione, oltre che a preparare l'asse X e l'asse Y a seconda dei valori da visualizzare.

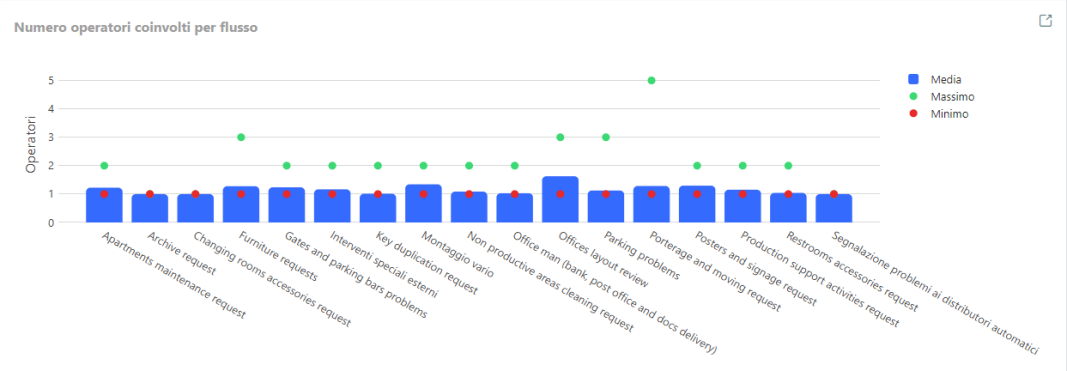


Figura 4.7: Esempio di grafico a dispersione (*Scatter*) e a colonne (*Bar*) *viz-lib*

4.2.2.3 Table

La componente `Table` costituisce una tabella che permette di visualizzare i dati in forma tabellare.
Tale *widget* utilizza `DataTable`, una componente resa disponibile dalla libreria *PrimeReact* che, attraverso la definizione di determinate props, rende disponibili le seguenti funzionalità:

- alternare i colori delle righe della tabella, al fine di aumentare la leggibilità dei dati visualizzati;
- selezionare più righe della tabella, permettendo di effettuare operazioni di *multi-selection* (attraverso l'utilizzo di uno `useState`);
- impostare un filtro globale per la tabella, permettendo di definire su quali colonne effettuare la ricerca dei dati;
- impostare un *empty message* personalizzato, visualizzato nel caso in cui la tabella non contenga dati;
- ordinare i dati all'interno delle singole colonne, permettendo di visualizzare i dati in ordine crescente o decrescente.

Come introdotto nella sezione (§3.4.3 *proprietà della componenti: Table*), la gestione del filtro globale tramite la funzione `onChangeGlobalFilterValue` presenta l'utilizzo di due valori tramite `useState`:

- `globalFilterValueTmp`: stringa utilizzata per memorizzare il valore del filtro impostato all'interno di una componente `InputText` di *PrimeReact*, aggiornata in tempo reale all'input fornito dell'utente;
- `globalFilterValue`: stringa utilizzata per impostare il valore del filtro globale della tabella, aggiornata tramite una chiamata *debounced* di 200 ms per impostare il valore sullo stato.
  (La funzione `debounced` fa uso dell'hook `useDebounceCallback` fornito dalla libreria *usehook-ts*).

Tramite questa implementazione, il recupero dei dati visualizzati all'interno della tabella viene eseguito correttamente, permettendo di risolvere il bug presente all'interno della componente `DataTable`, il quale comportava la perdita di *entry* nel caso in cui il filtro globale subisse più modifiche in rapida successione.
La tabella è resa *scrollable*, impostando una altezza che viene calcolata in modo responsivo a seconda dello spazio disponibile all'interno del *widget* in cui è contenuta.
Attraverso la definizione di un *ref* alla componente `DataTable`, è possibile effettuare il download della tabella in formato *CSV*, tramite l'utilizzo della funzione *exportCSV* resa disponibile dalla libreria *PrimeReact*: tale funzione viene invocata tramite un `Button` posizionato a fianco della componente `InputText` utilizzata per l'input del filtro globale.

Elaborazione dati
I dati renderizzati all'interno della tabella vengono estratti dal *props* `data`, passando a sua volta alla componente `DataTable` l'array `rows` contenente i dati da visualizzare tramite la prop `value`.
Le colonne della tabella sono definite a partire dalle `options` e dai `data` passati come *props* alla componente `Table`. Inizialmente sono ridefinite mediante la funzione `getOptions`, la quale imposta delle opzioni di default e ordina la visualizzazione delle colonne in base all'ordine definito nelle `options`, impostando le proprietà di filtro delle colonne in base alle opzioni indicate, verificando la presenza di dati all'interno dei `data` (nel caso di assenza di riscontro vengono definite le colonne a partire dai `data`).
Successivamente dalle `options` elaborate vengono definite le vere e proprie colonne della tabella (*tableColumns*), attraverso la funzione `prepareColumns`, la quale attua il filtraggio delle colonne e costruisce il loro *header* in base alle opzioni elaborate.
Le `tableColumns` vengono in seguito mappate per creare le componenti `Column` presentate all'interno della tabelle, impostando:

- l'*header* a partire dalle opzioni elaborate;
- il *field* a cui fanno riferimento a partire dalle opzioni elaborate;
- la *props sortable*, utilizzata per permettere l'ordinamento sulla colonna;
- la*props body*, utilizzata per permettere il corretto formattamento e rendering dei dati visualizzati all'interno della singola colonna, invocando la funzione `formatRowValue`.

Le formattazioni dei dati avvengono mediante i *formatter* resi disponibili:

- *string*: formatta il testo in base indicazioni presenti nell'item, eventualmente restituendo del contenuto *HTML*;
- *datetime*: formatta le date in base alle opzioni elaborate mediante l'uso di *day.js*;
- *number*: formatta i numeri in base alle opzioni elaborate mediante l'uso di *numbro*;

- *boolean*: formatta i booleani in base alle opzioni fornite, elaborando anche array;
- *json*: formatta i dati *JSON* in base alle opzioni fornite, permettendo di visualizzare i dati in modo strutturato;
- *image*: formatta le immagini in base alle opzioni fornite, permettendo di visualizzarle all'interno della tabella.

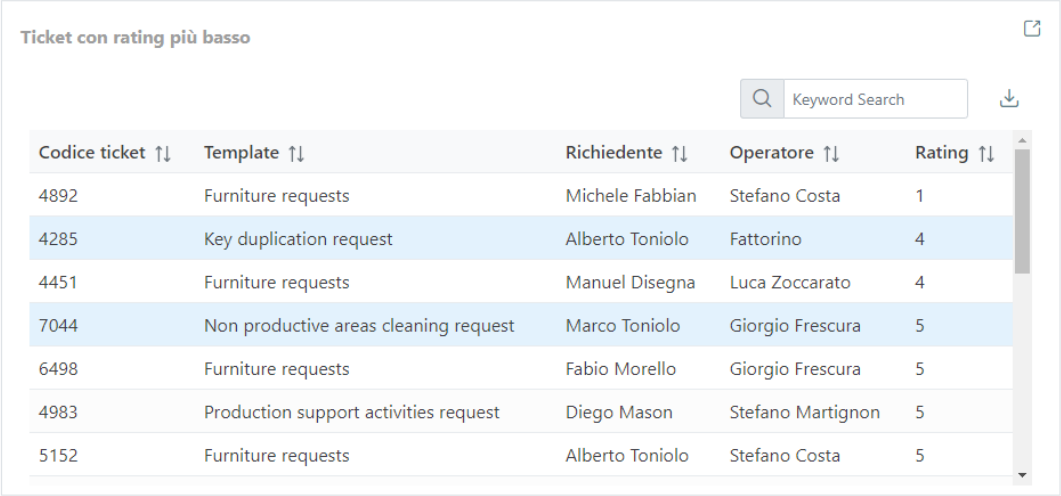


Figura 4.8: Esempio di `Table` *viz-lib*

4.2.2.4 Renderer

La componente *Renderer* costituisce l'*entry point* della libreria, permettendo di renderizzare tutte le componenti presenti all'interno di *viz-lib*: per questo motivo tale componente costituisce l'unico *export* della libreria accessibile all'*index.ts* del SDK. Tale componente costituisce il *wrap* della componente *RendererR*, ottenuto tramite l'utilizzo della funzione `React.memo`.

> `React.memo` è una funzione offerta da *React* per effettuare la *memoization* di una componente funzionale, permettendo di evitare nuove renderizzazioni nel caso in cui il componente padre lo richiedesse, a meno che le sue *props* non abbiano subito modifiche.
> Tale funzione accetta due parametri:
>
> - la componente da *memoizzare*;
> - una funzione che accetta due argomenti: le *props* correnti e le *props* precedenti della componente, restituendo un valore booleano che indica se la componente debba essere renderizzata o meno.
>
> La presenza del secondo parametro permette di effettuare un controllo specifico, andando a definire i criteri di cambiamento delle *props* rilevanti ai fini della nuova renderizzazione della componente.

Nel nostro caso, la funzione `React.memo` utilizza come criterio di controllo tra le *props* la funzione `isEqual` resa disponibile da *lodash*, la quale permette di effettuare un confronto profondo tra due valori per determinare se sono equivalenti.

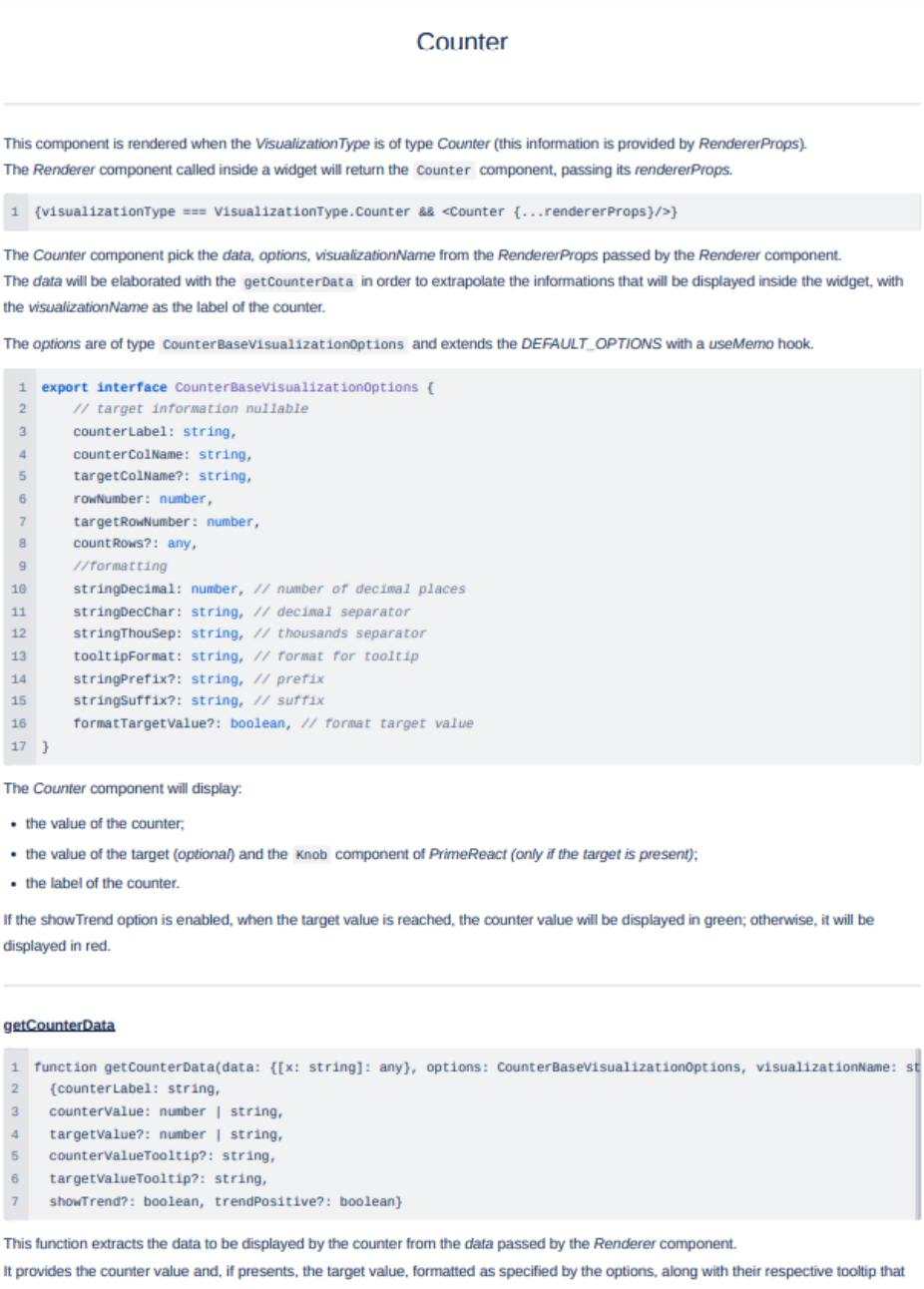
```
const Renderer = React.memo(RendererR, (prev, next) =>
    isEqual(prev.data, next.data)
);
```

Codice 4.3: `React.memo` della componente `Renderer`

Per quanto riguarda la componente `RendereR` in questione, essa ricava dai suoi *props*, tramite l'utilizzo della funzione `getVisualizationType`, il valore dell'enumerazione `VisualizationType`, assegnandolo alla costante `visualizationType`, ai fini di renderizzare la componente corretta in base al tipo di visualizzazione ricevuto come parametro.

4.2.3 Documentazione

L'implementazione della libreria prodotta è stata documentata tramite l'utilizzo di *Confluence*, la piattaforma di gestione della conoscenza e di collaborazione sviluppata da *Atlassian*.
La documentazione, su richiesta dell'azienda, è stata redatta in lingua inglese, attraverso una descrizione dettagliata delle componenti e delle funzionalità offerte dalla libreria.



Counter

This component is rendered when the *VisualizationType* is of type *Counter* (this information is provided by *RendererProps*).
The *Renderer* component called inside a widget will return the `Counter` component, passing its *rendererProps*.

```
{visualizationType === VisualizationType.Counter && <Counter {...rendererProps}/>}
```

The *Counter* component pick the *data*, *options*, *visualizationName* from the *RendererProps* passed by the *Renderer* component.
The *data* will be elaborated with the `getCounterData` in order to extrapolate the informations that will be displayed inside the widget, with the *visualizationName* as the label of the counter.

The *options* are of type `CounterBaseVisualizationOptions` and extends the *DEFAULT_OPTIONS* with a *useMemo* hook.

```
export interface CounterBaseVisualizationOptions {
    // target information nullable
    counterLabel: string,
    counterColName: string,
    targetColName?: string,
    rowNumber: number,
    targetRowNumber: number,
    countRows?: any,
    //formatting
    stringDecimal: number, // number of decimal places
    stringDecChar: string, // decimal separator
    stringThouSep: string, // thousands separator
    tooltipFormat: string, // format for tooltip
    stringPrefix?: string, // prefix
    stringSuffix?: string, // suffix
    formatTargetValue?: boolean, // format target value
}
```

The *Counter* component will display:

- the value of the counter;
- the value of the target (*optional*) and the `Knob` component of *PrimeReact (only if the target is present)*;
- the label of the counter.

If the showTrend option is enabled, when the target value is reached, the counter value will be displayed in green; otherwise, it will be displayed in red.

getCounterData

```
function getCounterData(data: {[x: string]: any}, options: CounterBaseVisualizationOptions, visualizationName: st
  {counterLabel: string,
  counterValue: number | string,
  targetValue?: number | string,
  counterValueTooltip?: string,
  targetValueTooltip?: string,
  showTrend?: boolean, trendPositive?: boolean}
```

This function extracts the data to be displayed by the counter from the *data* passed by the *Renderer* component.
It provides the counter value and, if presents, the target value, formatted as specified by the options, along with their respective tooltip that

Figura 4.9: Esempio documentazione Confluence

La documentazione è stata strutturata in modo da essere facilmente consultabile e comprensibile, con l'obiettivo di fornire un supporto efficace agli sviluppatori futuri che potrebbero dover utilizzare o lavorare sopra l'SDK implementato.
La produzione di una buona documentazione ricopre infatti un ruolo fondamentale ai fini di garantire la manutenibilità del codice e la facilità di comprensione delle funzionalità offerte dalla libreria, in modo da ridurre i tempi di apprendimento e di sviluppo necessari per gli utilizzi futuri del prodotto implementato.

4.3 Testing

Nella presente sezione verranno descritte le attività di testing effettuate durante lo sviluppo della libreria, con l'obiettivo di garantire la qualità del prodotto implementato e la corretta esecuzione delle funzionalità offerte.

4.3.1 Jest

Lo strumento utilizzato e configurato all'interno del progetto per l'esecuzione dei test è *Jest*, un framework di testing per *JavaScript* sviluppato da *Facebook*.
Jest permette di effettuare test su funzioni, classi e moduli, fornendo un'ampia gamma di funzionalità per la scrittura e l'esecuzione dei test.
In particolare, *Jest* offre le seguenti funzionalità:

- **Mocking components**: permette di creare *mock*$_G$ di componenti, in modo da simularne il suo comportamento:

```
jest.mock('percorso.componente', () => ({
    Componente: () => mock_value,
}));
```

Codice 4.4: Esempio di *mock* di una componente

- **Mocking function**: permette di creare *mock* di funzioni, in modo da simularne il suo comportamento:

```
jest.spyOn(file_funzione, 'nomeFunzione')
    .mockReturnValue(mock_value);
```

Codice 4.5: Esempio di *mock* di una funzione

- **Suite di test**: permette di creare *suite* di test, organizzando i test in modo gerarchico:

```
describe('Nome suite di test', () => {
    it('Nome test', () => {
        // Codice del test
    });
});
```

Codice 4.6: Esempio di *suite* di test

- **Expect**: permette di effettuare asserzioni sui valori restituiti dalle funzioni, verificando la correttezza del risultato ottenuto:

```
expect(valore).toBe(valore_aspettato);
```

Codice 4.7: Esempio di `expect` su valore

oppure di verificare la presenza di un elemento all'interno del *DOM*:

```
expect(screen.getByText('Testo')).toBeInTheDocument();
```

Codice 4.8: Esempio di `expect` su elemento del *DOM*

La sua configurazione è stata effettuata all'interno del file *packages.json*, in cui sono state definite le impostazioni di esecuzione dei test e le dipendenze necessarie per il loro corretto funzionamento, come il preset che consente di utilizzare *Typescript*, l'ambiente di test *jsdom* che simula un ambiente browser e i vari formati di file che deve considerare o ignorare.
Di seguito viene riportata la configurazione utilizzata.

```
"jest": {
    "preset": "ts-jest",
    "testEnvironment": "jsdom",
    "moduleFileExtensions": [
        "ts",
        "tsx",
        "js",
        "jsx",
        "json",
        "node"
    ],
    "transform": {
        "^.+\\.(ts|tsx)$": "ts-jest",
        "^.+\\.(js|jsx)$": "babel-jest"
    },
    "transformIgnorePatterns": [
        "node_modules/(?!(d3-color)/)"
    ]
}
```

Codice 4.9: Configurazione *Jest* all'interno del file *packages.json*

4.3.2 Unit testing

Per garantire la correttezza delle funzionalità offerte dalla libreria, è stato effettuato un processo di testing a livello di unità.
Nel presente progetto sono stati implementati test per le singole componenti, *mockando* le dipendenze esterne ed eventuali altre componenti della libreria utilizzate, verificando il corretto funzionamento all'interno della singola unità.

```
jest.mock('./charts/Counter', () => ({
    Counter: () => <div>Counter Component</div>,
}));

jest.mock('./charts/Chart', () => ({
    Chart: () => <div>Chart Component</div>,
}));

jest.mock('./charts/Table', () => ({
    Table: () => <div>Table Component</div>,
}));

describe('Renderer test right components', () => {
    it('renders Counter component when visualizationType is Counter',
        () => {
        jest.spyOn(helper, 'getVisualizationType')
            .mockReturnValue(VisualizationType.Counter);

        const rendererProps: RendererProps = {
            visualizationName: 'Counter',
            type: 'Counter',
            data: {},
            options: {},
        };

        render(<Renderer {...rendererProps} />);

        expect(screen.getByText('Counter Component')).toBeInTheDocument();
        });
// ...
});
```

Codice 4.10: Esempio di *unit test*: `Renderer` component

Come si può osservare nell'esempio di codice riportato, è stato effettuato un test sulla componente *Renderer*, *mockando* le componenti *Counter*, *Chart* e *Table* utilizzate all'interno della componente stessa, verficando che venisse renderizzata la componente corretta in base al tipo di visualizzazione passato come parametro.
Il tipo di visualizzazione a sua volta viene ottenuto dal *mock* della funzione `getVisualizationType`, la quale restituisce il tipo di visualizzazione corretto in base al parametro passato come *props* al `Renderer`: in questo modo viene garantita la correttezza logica della singola componente, senza far affidamento su funzioni o componenti esterne.

La correttezza delle funzioni è stata verificata tramite appositi test di unità, eventualmente *mockando* le dipendenze esterne utilizzate all'interno della funzione stessa, in modo da garantire la correttezza nella logica implementata.

Di seguito viene riportato un esempio di test di unità effettuato sulla funzione `getVisualizationType`.

```
it('should return VisualizationType.Table when type is TABLE', () => {
    expect(
        getVisualizationType('TABLE')
    ).toBe(VisualizationType.Table);
});
```

Codice 4.11: Esempio di *unit test*: `getVisualizationType`

4.3.3 Integration testing

Per garantire la corretta integrazione delle componenti all'interno della libreria, è stato effettuato un processo di testing a livello di integrazione, tramite l'utilizzo di *Jest*.
I test di integrazione permettono di verificare il corretto funzionamento delle componenti all'interno del sistema, testando il comportamento dei singoli moduli all'interno del contesto in cui sono utilizzati.

```
jest.mock('./charts/Counter', () => ({
    Counter: () => <div>Counter Component</div>,
}));

jest.mock('./charts/Chart', () => ({
    Chart: () => <div>Chart Component</div>,
}));

jest.mock('./charts/Table', () => ({
    Table: () => <div>Table Component</div>,
}));

describe('Renderer test right components', () => {
    it('renders Counter component when visualizationType is Counter', () => {

        const rendererProps: RendererProps = {
            visualizationName: 'Counter',
            type: 'Counter',
            data: {},
            options: {},
        };

        render(<Renderer {...rendererProps} />);

        expect(screen.getByText('Counter Component')).toBeInTheDocument();
    });
// ...
});
```

Codice 4.12: Esempio di *integration test*: `Renderer` component

Come si può osservare nell'esempio di codice riportato, è stato effettuato un test di integrazione sulla componente `Renderer`, *mockando* le componenti `Counter`, `Chart` e `Table` utilizzate all'interno della componente stessa, verficando che venisse renderizzata la componente corretta in base al tipo di visualizzazione passato come parametro. Il tipo di visualizzazione a sua volta viene ottenuto internamente alla componente grazie alla funzione `getVisualizationType`, la quale restituisce il tipo di visualizzazione corretto in base al parametro ricevuto come *props* dal `Renderer`: in questo modo viene garantita la corretta integrazione delle componenti all'interno del sistema, testando il comportamento delle unità in relazione tra loro all'interno del contesto in cui sono utilizzate.

Capitolo 5

Rilascio

In questa sezione verrà presentato il rilascio del prodotto finale, concludendo infine con un esempio di integrazione dell'SDK all'interno di un prodotto *Datasoil S.rl.* operativo.

5.1 Rilascio SDK

Al fine di rilasciare un *package*, è necessario effettuare la *build* del prodotto.
Per il processo di *building* del codice è stato utilizzato *Rollup*, un *bundler* di moduli *JavaScript* che permette di:

- creare *bundle* di moduli in formato *ESM* (ECMAScript Module) e *CJS* (CommonJS);
- utilizzare *TypeScript* all'interno del progetto, integrando la configurazione dichiarata nel file *tsconfig.json*;
- risolvere le dipendenze tra i moduli, escludendo le *peerdependency* dal *bundle*;
- produrre *bundle* di dimensioni ridotte grazie alla sua capacità di effettuare il *tree-shaking*;
- supportare plugin per il *terser*, utilizzato per la minimizzazione del codice prodotto attraverso la rimozione dei commenti e degli spazi vuoti, effettuando il *munging* dei nomi delle variabili e introducendo ottimizzazioni per ridurre la dimensione finale;
- supportare *sourcemaps* per facilitare il debug del codice, permettendo di mappare il codice minificato con il codice sorgente originale;
- generare file di dichiarazione *TypeScript*;
- supportare plugin per il calcolo della dimensione del *bundle*, specificando la dimensione di ogni singola dipendenza all'interno del progetto, generando un file di report *html*.

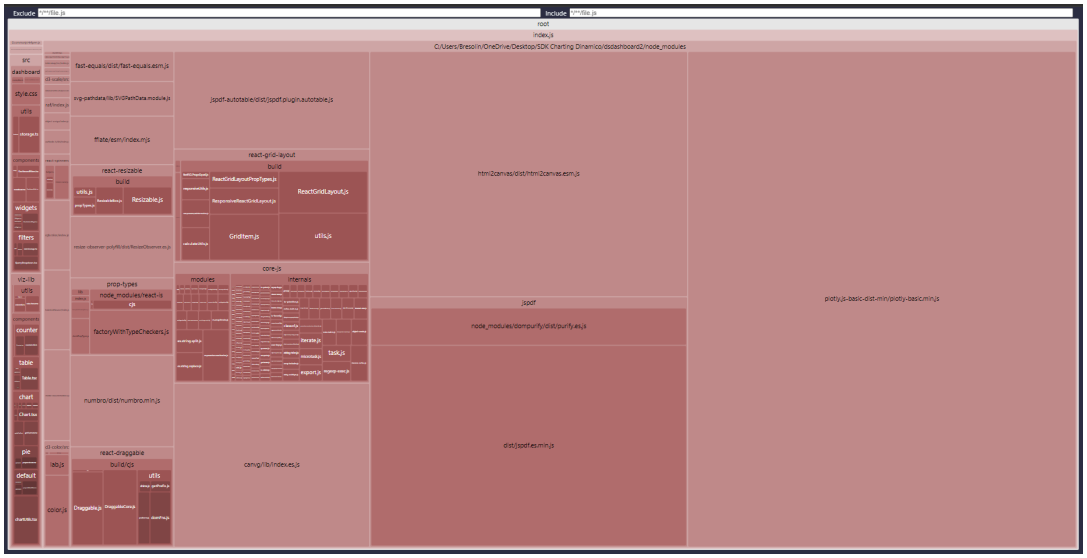


Figura 5.1: Esempio di *bundle visualizer report*

Per effettuare la build del progetto è necessario eseguire il comando *yarn build-package* configurato all'interno del file *package.json* del progetto: tale comando, come mostrato nel listato (§4.1), effettua la *build* del progetto generando i file di output all'interno della cartella *dist*.
Data la configurazione all'interno del file *rollup.config.js*, il comando effettua la *build* del progetto in formato *ESM* e *CJS*.
Per specificare il punto di ingresso per i consumatori dei *package*, sono state definite all'interno del file *package.json* le chiavi *main* e *module* che puntano rispettivamente ai file *dist/cjs/index.js* e *dist/esm/index.esm.js*. Per quanto riguarda il rilascio vero e proprio, utilizzando *Yarn* come *package manager*, il rilascio di un *package* può avvenire principalmente tramite tre modalità differenti:

- **Registro di npm**: è il registro di default di *Yarn*, un registro pubblico che permette a chiunque abbia un account di pubblicare pacchetti. Per impostazione predefinita (ma comunque configurabile), i pacchetti pubblicati su *npm* sono visibili a tutti e di conseguenza tutti possono usufruirne;

- **Registro privato**: è un registro privato che permette di pubblicare pacchetti in un ambiente accessibile solo a chi è autorizzato. Esistono differenti servizi che offrono registri privati, a seconda del contesto operativo e delle esigenze dei prodotti che devono usufruire di tali pacchetti;

- **Github Packages**: è un *software package hosting service* offerto da *Github* che permette di pubblicare pacchetti in modo pubblico o privato all'interno di un repository *Github*, similmente ad un registro *npm*.

Per questo progetto si è optato per l'utilizzo di quest'ultima opzione come registro di pubblicazione dei pacchetti, in quanto all'interno di *Datsoil S.r.l.* i *package* prodotti vengono pubblicati all'interno dello spazio *Github Packages* dell'organizzazione

aziendale: questo servizio è infatti molto utile per chi utilizza *Github* come sistema di versionamento e vuole mantenere i pacchetti all'interno del proprio *repository*. Per pubblicare un pacchetto all'interno di *Github Packages* è necessario configurare il file *package.json* del progetto, specificando il campo *publishConfig* nel seguente modo:

```
{
    "publishConfig": {
        "registry": "https://npm.pkg.github.com/"
    }
}
```

Codice 5.1: Configurazione del campo *publishConfig* all'interno del file *package.json*

In seguito, dopo aver configurato il file *.npmrc* all'interno del progetto, specificando il *token* di autenticazione per l'accesso al registro di *Github Packages*, è possibile eseguire il comando *yarn publish* per pubblicare il pacchetto: una volta avviato il comando, verrà illustrata la versione precedente del *package* e verrà infine richiesto di fornirne una nuova.
Una volta terminato il processo di pubblicazione, il pacchetto sarà disponibile all'interno del registro *Github Packages* dell'organizzazione aziendale.

5.2 Integrazione SDK all'interno di SYNMGR

Durante lo svolgimento dello stage ho avuto la possibilità di integrare l'SDK prodotto all'interno del prodotto *SYNMGR* di *Datasoil S.r.l.*, l'applicazione web di test utilizzata dall'azienda per testare gli sviluppi futuri e le nuove funzionalità che verranno in seguito rilasciate sul prodotto ufficiale *SYN*.
Per utilizzare l'SDK all'interno di *SYNMGR* è stato necessario configurare il file *.npmrc* all'interno del progetto, specificando il *token* di autenticazione per l'accesso al registro di *Github Packages*: utilizzando infatti il registro fornito da *Github*, i package manager come *Yarn* e *npm* andranno a verificare la presenza di tali informazioni e, solo nel caso di riscontro positivo, permetteranno l'installazione dei pacchetti richiesti.
Successivamente è stata sostituito il precedente SDK *dsdashboard* a favore del nuovo SDK *dsdashboard2* all'interno del file *package.json* del progetto, eseguendo infine il comando *yarn install* per l'installazione del pacchetto, permettendo di utilizzare all'interno del prodotto le nuove implementazioni delle componenti inerenti alle dashboard.

```
{
    "dependencies": {
        "@datasoil/dsdashboard2": "^0.1.5"
    }
}
```

Codice 5.2: Configurazione del campo *dependencies* all'interno del file *package.json* di *SYNMGR*

Di seguito, un esempio di una dashboard presente all'interno di *SYNMGR* che utilizza l'SDK *dsdashboard2*.

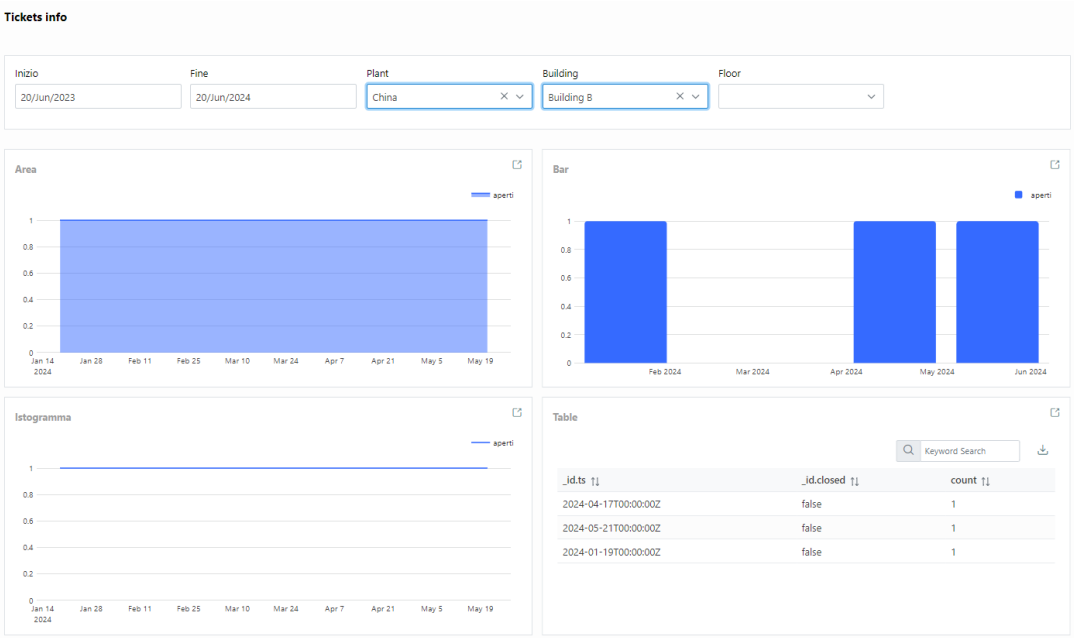


Figura 5.2: Esempio di dashboard all'interno di *SYNMGR*

Capitolo 6

Conclusioni

Nella seguente ed ultima sezione verranno presentate le conclusioni del progetto di stage, valutando i risultati conseguiti, proponendo possibili sviluppi futuri ed esponendo le riflessioni finali maturate.

6.1 Consuntivo finale

Nella seguente tabella vengono riportate le ore effettive svolte durante il progetto di stage, suddivise per attività e per periodo di svolgimento.

Ore	Settimane	Descrizione
54	1, 2	Formazione su tecnologie utilizzate, studio del progetto e del SDK preesistente
68	2, 3, 4	Progettazione SDK: layer di rappresentazione - grafici
90	4, 5, 6	Sviluppo SDK: layer di rappresentazione - grafici
30	6	Progettazione SDK: layer user interaction
52	7, 8	Refactor SDK: layer user interaction
10	8	Deployment dell'SDK prodotto
304	**Totale ore svolte**	

Tabella 6.2: Tabella consuntivo finale

6.2 Valutazione del progetto

Il progetto di stage ha permesso di sviluppare un prodotto finale che rispondesse alle esigenze dell'azienda, soddisfacendo tutti requisiti individuati nel capitolo (§Analisi dei requisiti).
L'SDK *dsdashboard2* è stato implementato con successo, introducendo nuove funzionalità e miglioramenti rispetto al precedente SDK *dsdashboard*, raggiungendo dimensioni

del *bundle* minori (1.26 MB → 418.03 kB (gzip) rispetto ai 4.11 MB → 1.203 MB (gzip) iniziali) e una miglior leggibilità e manutenibilità del codice.
In merito alle tecnologie utilizzate, gli strumenti e le librerie adottate si sono rivelate efficaci e funzionali per lo sviluppo del prodotto, permettendo di implementare un prodotto di qualità. Fa eccezione la libreria *Plotly.js*, il cui utilizzo è risultato vincolato dalle risposte fornite dalle *API* del *backend* per motivi di retrocompatibilità, la quale ha richiesto adattamenti e modifiche sostanziali al codice per garantire un corretto ed efficace funzionamento dell'SDK.
Il progetto nel suo complesso costituisce un *tool* grafico avanzato e robusto, pronto per essere utilizzato all'interno dei prodotti Datasoil S.r.l., sebbene sia da sottolineare la forte dipendenze con le *API* interrogate, le quali costituiscono il principale punto critico del prodotto, ostacolando uno sviluppo flessibile e indipendente.

6.3 Possibili sviluppi futuri

La progettazione dell'SDK *dsdashboard2* è stata effettuata non solo con l'ottica di introdurre nuove funzionalità e miglioramenti rispetto al precedente SDK *dsdashboard*, ma anche con l'intenzione di migliorare la manutenibilità e agevolare l'estendibilità del prodotto.
Possibili sviluppi futuri della libreria potrebbero dunque includere l'introduzione di nuove componenti grafiche, a partire da tutti quei grafici resi disponibili da *Plotly.js* e non ancora implementati all'interno dell'SDK, come ad esempio i grafici a radar o i grafici a proiettile (*bullet chart*).
Sebbene all'interno dell'SDK l'introduzione di nuove componenti non costituisca un problema, rimane da valutare le modifiche lato backend da introdurre all'interno dei prodotti Datasoil S.r.l. per permettere il supporto di tali nuove *widget*: *dsdashboard2* è infatti progettato per essere utilizzato all'interno di un contesto applicativo in cui le dashboard vengono generate a partire da configurazioni fornite lato *backend* e non in un prodotto in cui le componenti vengono montate direttamente nel codice *frontend*.
Altre possibili sviluppi futuri potrebbero consistere nell'introduzione della funzionalità di *drag and drop* all'interno dell'SDK, permettendo all'utente di spostare e ridimensionare le componenti all'interno della dashboard, in modo da poter personalizzare la schermata in base alle proprie esigenze specifiche.
Infine, un ulteriore feature che era stata implementata durante il periodo di stage ma che a causa delle dimensioni in termini di memoria delle librerie utilizzate era stata in seguito rimossa, è la funzionalità di *export* delle tabelle in formato *PDF*: questa funzionalità potrebbe essere reintrodotta all'interno dell'SDK con una implementazione più leggera e performante, permettendo all'utente di esportare i dati in un formato più leggibile e di facile consultazione rispetto al limitato *CSV*.

6.4 Riflessioni finali

Durante lo svolgimento dello stage ho potuto apprendere nuove competenze e conoscenze, applicandomi per la prima volta in un contesto lavorativo reale.
Ho avuto modo di studiare e lavorare con tecnologie che non avevo mai avuto l'opportunità di approfondire, come ad esempio la libreria di componenti *PrimeReact* e la libreria grafica *Plotly.js*, oltre che approfondire le mie conoscenze in ambito di sviluppo web con *React* e *TypeScript*, utilizzando la mia creatività e le mie competenze per sviluppare un prodotto finale che rispondesse alle esigenze dell'azienda.
Ho avuto l'opportunità di lavorare all'interno di un team di sviluppo, confrontandomi con colleghi più esperti e apprendendo costantemente da loro, oltre che collaborare con il mio tutor aziendale, il quale mi ha guidato e supportato durante tutto il percorso di stage.
Durante questa esperienza, è stato quindi fondamentale il saper comunicare con i miei colleghi, affinando le mie capacità di comunicazione e collaborazione, prestando la massima attenzione ai dettagli e consigli ricevuti, cercando di migliorare costantemente il mio *way of working* e le mie competenze.
Il progetto mi ha permesso di acquisire una maggior consapevolezza delle mie capacità, mettendomi alla prova in un ambiente professionale e affrontando problematiche concrete che richiedessero soluzioni immediate ma allo stesso tempo funzionali e ben strutturate: ciò che emerge in modo più evidente da questa esperienza è infatti l'importanza cruciale di valutare attentamente le soluzioni proposte, analizzando i pro e i contro di ogni opzione possibile, poiché ogni decisione avrà ripercussioni significative sul progetto e sul lavoro futuro di manutenzione del prodotto sviluppato.
E' risultato dunque fondamentale il saper sviluppare una valutazione critica che permettesse di individuare le soluzioni migliori e non quelle più immediate, cercando di prevedere possibili problematiche future.
In conclusione, valuto quindi positivamente l'esperienza di stage svoltasi presso l'azienda *Datasoil S.r.l.*, in quanto mi ha permesso di crescere professionalmente e personalmente, acquisendo e affinando competenze e conoscenze funzionali al mio percorso di studi e alla mia futura carriera lavorativa.

Bibliografia

Siti

Component Library Architecture Pattern. URL: https://www.emergeagency.com/insights/detail/how-to-ux-ui-design-system-component-library/.

Composition Component Pattern Felix Gerschau. URL: https://felixgerschau.com/react-component-composition/.

Documentazione Confluence. URL: https://www.atlassian.com/it/software/confluence.

Documentazione D3.js. URL: https://d3js.org/.

Documentazione Day.js. URL: https://day.js.org/.

Documentazione Jest. URL: https://jestjs.io/.

Documentazione Lodash.js. URL: https://lodash.com/.

Documentazione Numbro.js. URL: http://numbrojs.com/.

Documentazione package.json. URL: https://docs.npmjs.com/files/package.json.

Documentazione Plotly.js. URL: https://plot.ly/javascript/.

Documentazione PrimeFlex. URL: https://www.primefaces.org/primeflex/.

Documentazione PrimeReact. URL: https://www.primefaces.org/primereact/.

Documentazione React. URL: https://react.dev/.

Documentazione useHooks. URL: https://usehooks-typescript.com/.

Registro Yarn. URL: https://yarnpkg.com/.

Rollup Building Library. URL: https://rollupjs.org/introduction/.