Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica

A.a. 2023/2024

Sessione di laurea Ottobre 2024

Web dashboard per l'Analisi Esplorativa e la Gestione di Grandi Volumi di Dati Temporali

Relatore:	Tutor Aziendale:	Candidato:
Luigi De Russis	Pierluigi Di Nunzio	Giovanni Giunta

Ringraziamenti

La stesura di questa tesi rappresenta per me un traguardo significativo, raggiunto grazie al supporto di chi mi ha accompagnato durante questo percorso, e a cui desidero rivolgere i miei più sinceri ringraziamenti.

In primo luogo, vorrei esprimere la mia sincera gratitudine al mio relatore, Prof. Luigi De Russis, per la sua guida, i suoi preziosi consigli e la costante disponibilità durante tutto il percorso di elaborazione di questa tesi. La sua professionalità e attenzione sono state fondamentali per il raggiungimento di questo obiettivo.

Un ringraziamento speciale va anche all'azienda DNDG, che mi ha dato l'opportunità di svolgere il mio lavoro di tesi all'interno della loro realtà, fornendomi gli strumenti e il supporto necessari per portare a termine questo progetto. In particolare, desidero ringraziare i titolari, Gigi Di Nunzio e Federico Di Gregorio, per la loro disponibilità, fiducia e prezioso contributo durante tutte le fasi del mio lavoro.

Grazie per il vostro prezioso contributo, che ha reso possibile il raggiungimento di questo importante traguardo, fondamentale per me e che ha concluso il mio percorso accademico.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Contesto

Nell'era digitale contemporanea, la gestione e l'analisi dei dati rappresentano un pilastro fondamentale per l'innovazione e il progresso in una vasta gamma di settori, dall'economia alla scienza, passando per l'industria e la sanità. In particolare, la capacità di esplorare e gestire grandi volumi di dati temporali in tempo reale è diventata essenziale per prendere decisioni informate, identificare pattern significativi e trarre insight utili. Questo contesto è alimentato da una crescente quantità di dati generati quotidianamente da una varietà di fonti.

L'era dei Big Data ha portato con sé una proliferazione di dati provenienti da sensori IoT, registrazioni finanziarie, dati energetici, climatici e molto altro ancora. Questi dati si presentano spesso sotto forma di serie temporali, con timestamp che registrano i cambiamenti nel tempo. Le serie temporali, essendo sequenze di dati raccolti o registrati a intervalli di tempo specifici, offrono una ricca fonte di informazioni che può rivelare tendenze, stagionalità e anomalie se analizzate correttamente.

La gestione efficace di queste serie temporali pone sfide significative. Richiede strumenti potenti ed efficienti che consentano agli utenti di esplorare i dati in modo dinamico e reattivo. Le tecnologie tradizionali spesso non riescono a gestire la quantità e la velocità dei dati temporali moderni, rendendo necessario lo sviluppo di soluzioni innovative che possano supportare analisi in tempo reale e scalabilità.

In questo scenario, emerge la necessità di una Web Dashboard avanzata per l'analisi esplorativa e la gestione di enormi quantità di dati temporali, cioè un'interfaccia utente visuale accessibile tramite un browser web che consente di raccogliere, visualizzare e monitorare dati provenienti da varie fonti in tempo reale o quasi.

Solitamente, una Web Dashboard aggrega informazioni rilevanti per un determinato contesto o sistema, presentandole in modo comprensibile attraverso grafici,

tabelle, indicatori e altri elementi visivi.

Le Web Dashboard sono ampiamente utilizzate per monitorare KPI (Key Performance Indicators), gestire dati aziendali, analizzare trend o performance di sistemi, e consentono agli utenti di esplorare i dati in maniera interattiva, spesso con la possibilità di applicare filtri, modificare parametri, e generare report personalizzati.

Una dashboard ben progettata può offrire agli utenti un'interfaccia intuitiva e potente per visualizzare, analizzare e interagire con i dati temporali in tempo reale. Le funzionalità di visualizzazione avanzate possono aiutare a trasformare dati complessi in insight comprensibili, facilitando decisioni rapide e basate su dati concreti.

In sintesi, il contesto attuale vede una necessità critica per strumenti che possano gestire efficacemente grandi volumi di dati temporali. La progettazione e lo sviluppo di una Web Dashboard per l'analisi esplorativa e la gestione dei dati temporali rappresentano una risposta a queste esigenze, promettendo di fornire agli utenti un'esperienza utente senza precedenti in termini di prestazioni e funzionalità.

1.2 Obiettivi

L'obiettivo principale di questo progetto è proporre un framework prototipale basato su WebAssembly per la gestione efficiente dei dati, con un focus particolare sull'accumulo progressivo in tempo reale come caso d'uso. Questo framework nasce dall'analisi delle soluzioni esistenti per la creazione di Web Dashboard, con l'intento di offrire una piattaforma performante e flessibile, capace di affrontare le sfide legate alla gestione e visualizzazione di grandi quantità di dati in tempo reale.

Il progetto mira a integrare tecnologie moderne, come Rust e WebAssembly, per ottenere una piattaforma reattiva e capace di eseguire calcoli complessi direttamente nel browser, migliorando così l'esperienza utente e la velocità di elaborazione. In particolare, l'integrazione di Rust, noto per le sue prestazioni e la sicurezza nella gestione della memoria, con WebAssembly, che consente l'esecuzione di codice a velocità quasi nativa, rappresenta uno degli aspetti centrali della proposta.

Un ulteriore obiettivo è quello di offrire uno strumento che permetta l'analisi esplorativa dei dati temporali, facilitando l'interazione dell'utente con set di dati complessi. Questo sarà possibile grazie all'adozione di un'architettura scalabile, in grado di supportare frequenti ricalcoli e aggiornamenti dinamici.

In sintesi, gli obiettivi di questa tesi sono:

- Proporre un framework innovativo basato su WebAssembly per la gestione efficiente dei dati in tempo reale.
- Integrare tecnologie avanzate, come Rust e WebAssembly, per garantire alte prestazioni e sicurezza.

- Sviluppare una Web Dashboard interattiva, che supporti la visualizzazione e l'analisi di grandi volumi di dati temporali.

1.3 Struttura della tesi

Questa tesi è organizzata in diversi capitoli, ciascuno dei quali affronta aspetti specifici dello sviluppo e dell'implementazione del framework prototipale proposto e una Web Dashboard per l'analisi esplorativa dei dati temporali. Ecco una panoramica dei contenuti di ciascun capitolo:

- Il primo capitolo fornisce un'introduzione generale al contesto del progetto. Vengono delineate le motivazioni alla base della necessità di una Web Dashboard avanzata per l'analisi dei dati temporali, sottolineando l'importanza della gestione e dell'analisi dei dati nell'era digitale contemporanea. Viene esplorata la proliferazione dei dati temporali provenienti da diverse fonti e le sfide associate alla loro gestione efficace. Inoltre, vengono descritti gli obiettivi principali del progetto, che includono l'analisi delle soluzioni esistenti e lo sviluppo di una dashboard innovativa e performante.

- Nel capitolo 2, intitolato "Stato dell'arte", viene presentata una revisione approfondita della letteratura esistente riguardante le tecnologie e gli approcci per lo sviluppo di Web Dashboard. Il capitolo analizza due tra i principali tool utilizzati per la gestione dei dati temporali, evidenziando i loro vantaggi e svantaggi in relazione a prestazioni, sicurezza e usabilità. Inoltre, il capitolo affronta il tema dei database, concentrandosi in particolare su due principali architetture: i database organizzati per righe e i database organizzati per colonne.

- Nel capitolo 3, intitolato "Architettura e Sviluppo del Framework Prototipale", vengono descritti l'architettura e lo sviluppo di un framework prototipale che permette la creazione di una Web Dashboard in grado di gestire tanti dati in tempo reale e basato su tecnologie come WebAssembly e Rust. Il capitolo inizierà con una panoramica sui requisiti funzionali e tecnici richiesti per lo sviluppo, proseguirà poi l'illustrazione nel dettaglio delle tecnologie adottate per la realizzazione della soluzione proposta, evidenziando come ciascuna di esse contribuisca al raggiungimento degli obiettivi di performance e interattività prefissati. In primo luogo, vengono introdotte le tecnologie principali WebAssembly e Rust, che sono state selezionate per garantire un'esecuzione efficiente e ad alte prestazioni all'interno del browser. Successivamente, viene trattata l'integrazione di WebSocket, una tecnologia cruciale per abilitare la comunicazione bidirezionale in tempo reale tra il client e il server. Un ulteriore approfondimento riguarda l'utilizzo di React, una delle librerie JavaScript più

diffuse per la creazione di interfacce utente dinamiche e modulari. Viene a questo punto descritto il processo di implementazione che ha portato alla realizzazione della soluzione finale. In questa parte, il capitolo esplora in dettaglio il workflow adottato, le fasi di sviluppo, e le sfide tecniche incontrate durante l'integrazione di queste tecnologie avanzate. Infine, il capitolo concluderà con una guida dettagliata su come utilizzare il framework sviluppato per la creazione di una Web Dashboard.

- Nel capitolo 4, intitolato "Metodologia di Valutazione e Analisi delle Web Dashboard", viene presentata la procedura seguita per la scrittura dei test volti a valutare le performance delle Web Dashboard e quindi, nel nostro caso, anche dei tool utilizzati per la loro creazione e successivamente del framework prototipale proposto. Il capitolo si focalizza inizialmente sul concetto di Performance Testing e nella seconda parte invece verrà introdotta e analizzata la metodologia adottata per progettare e scrivere i test, descrivendo la scelta degli strumenti utilizzati e le tecniche di misurazione delle prestazioni. Viene inoltre illustrato il processo di configurazione dell'ambiente di testing, spiegando come siano stati definiti e implementati i test per analizzare le prestazioni.

- Nel capitolo 5, intitolato "Risultati", viene descritto il processo di esecuzione dei test di performance sulle soluzioni analizzate e sul framework prototipale sviluppato. In particolare verranno prima presentati i risultati ottenuti e successivamente verrà mostrato come il framework prototipale sviluppato utilizzando WebAssembly e Rust abbia apportato miglioramenti tangibili rispetto alle soluzioni esistenti in termini di reattività, gestione dei dati in tempo reale e interattività. L'analisi dei test verrà svolta confrontando le prestazioni della dashboard creata con il framework prototiaple proposto con quelle ottenute da Grafana e Metabase, evidenziando i punti di forza e le aree in cui la soluzione ha portato vantaggi concreti.

- Infine, nel capitolo 6, intitolato "Conclusioni", verranno tratte le considerazioni finali sul lavoro svolto in questa tesi. Questo capitolo offrirà una sintesi completa del percorso intrapreso, ricapitolando i principali passaggi dell'implementazione della soluzione proposta, le tecnologie utilizzate e i risultati ottenuti. Verranno inoltre proposti alcuni lavori futuri che è possibile svolgere partendo da quanto è stato svolto in questo lavoro di tesi.

Capitolo 2

Stato dell'arte

In questo capitolo, ci concentreremo sui principali aspetti legati alla progettazione e all'implementazione delle Web Dashboard, con un'enfasi particolare sulle performance quando si gestiscono grandi quantità di dati in tempo reale. Come affermato in precedenza, infatti, le Web Dashboard sono strumenti fondamentali per l'analisi dei dati temporali, utilizzate in vari settori per monitorare, visualizzare e analizzare informazioni dinamiche e in continua evoluzione.

La capacità di una dashboard di gestire e visualizzare dati in tempo reale è cruciale per molte applicazioni, dall'analisi finanziaria al monitoraggio dei sensori IoT, dalla gestione energetica alla previsione climatica. Tuttavia, questa capacità presenta notevoli sfide in termini di performance. Una dashboard deve essere in grado di aggiornare e visualizzare rapidamente i dati, senza ritardi che potrebbero compromettere l'usabilità e l'efficacia dello strumento.

Inizieremo con un'analisi delle Web Dashboard esistenti, esaminando le tecnologie attualmente disponibili e come queste vengono impiegate per risolvere problemi legati alla visualizzazione e gestione di grandi moli di dati. Questa analisi ci permetterà di identificare le sfide comuni e le esigenze critiche, come la velocità di aggiornamento, la reattività e la capacità di gestire flussi di dati in tempo reale.

Successivamente, passeremo ai tool per la creazione di Web Dashboard, concentrandoci su strumenti come Grafana e Metabase, i quali rappresentano due tra i più popolari tool per la creazione di Web Dashboard. Verranno esaminate le loro funzionalità principali, i loro punti di forza e le limitazioni nell'ambito della gestione di dati temporali in tempo reale. Questo confronto ci aiuterà a comprendere quale tool potrebbe essere più adatto per determinati contesti applicativi, tenendo conto delle specifiche esigenze di performance e di usabilità.

Dopo aver discusso dei tool, esploreremo l'importanza della scelta del database per ottimizzare le performance delle Web Dashboard. Verranno approfonditi i database organizzati per righe e i database organizzati per colonne, evidenziando le differenze fondamentali tra questi due approcci. Il tipo di database utilizzato ha un

impatto significativo sulla capacità della dashboard di gestire e analizzare grandi volumi di dati in tempo reale, ed è quindi cruciale comprenderne il funzionamento per ottenere i migliori risultati.

In sintesi, questo capitolo fornirà una panoramica completa delle tecnologie, dei tool e delle strategie più avanzate per sviluppare Web Dashboard performanti e reattive, capaci di gestire grandi quantità di dati temporali in modo efficiente.

2.1 Analisi delle Web Dashboard esistenti

Attraverso un'analisi dettagliata delle Web Dashboard esistenti e delle tendenze emergenti nel campo della gestione dei dati temporali, questa ricerca si propone di offrire una guida completa e aggiornata a chiunque sia coinvolto nello sviluppo e nell'utilizzo di queste potenti risorse per l'analisi dei dati. Le Web Dashboard sono diventate strumenti indispensabili nell'analisi e nella gestione dei dati, fornendo agli utenti un'interfaccia intuitiva e visivamente accattivante che facilita l'esplorazione e l'interpretazione di informazioni cruciali. Con il crescente volume e la complessità dei dati che le organizzazioni devono gestire, la progettazione e l'implementazione efficaci di queste dashboard sono diventate elementi critici per il successo di molte operazioni aziendali e scientifiche.

Nel panorama tecnologico attuale, diversi aspetti fondamentali determinano il successo e l'efficacia di una Web Dashboard. Tra questi, i seguenti elementi rivestono un'importanza particolare:

2.1.1 Interfaccia Utente e User Experience

L'interfaccia utente (UI) rappresenta il punto di contatto principale tra l'utente e la dashboard. Deve essere progettata con cura per essere intuitiva, offrendo un layout chiaro e una disposizione logica degli elementi che agevolino la comprensione dei dati presentati. Una User Interface ben progettata non solo migliora l'efficienza con cui gli utenti possono interagire con la dashboard, ma anche la loro capacità di interpretare rapidamente i dati. La User Experience (UX), ossia l'insieme delle percezioni, emozioni e reazioni di un utente durante l'interazione con un prodotto, sistema o servizio, deve essere fluida e reattiva, garantendo tempi di risposta rapidi per evitare frustrazioni e mantenere alti livelli di soddisfazione. Una buona esperienza utente dipende dalla capacità di anticipare le esigenze degli utenti e fornire risposte immediate ai loro input, evitando ritardi o interruzioni.

2.1.2 Sicurezza

In un contesto in cui le Web Dashboard gestiscono spesso dati sensibili, la sicurezza diventa un aspetto critico. La protezione dei dati non riguarda solo la prevenzione

di accessi non autorizzati, ma anche la salvaguardia dell'integrità e della riservatezza delle informazioni. È essenziale implementare misure di sicurezza robuste, tra cui l'autenticazione a più fattori, la crittografia dei dati e il controllo granulare degli accessi, per prevenire minacce informatiche e proteggere le risorse aziendali. Oltre a queste misure tecniche, è importante considerare anche la sicurezza durante la progettazione dell'architettura della dashboard, assicurando che i dati siano protetti in ogni fase del processo, dal trasferimento alla memorizzazione.

2.1.3 Responsive Design

Con la crescente diffusione dei dispositivi mobili, è diventato imprescindibile che le Web Dashboard siano progettate con un design responsive cioè adattabile, che garantisca un'esperienza ottimale su una vasta gamma di dispositivi e dimensioni dello schermo. Un design responsive assicura che la dashboard mantenga la sua funzionalità e usabilità indipendentemente dal dispositivo utilizzato, sia esso un desktop, un tablet o uno smartphone. Questo non solo migliora l'accessibilità della dashboard, ma amplia anche la sua utilità, permettendo agli utenti di accedere alle informazioni di cui hanno bisogno ovunque si trovino, in modo rapido e conveniente.

2.1.4 Performance

La performance di una Web Dashboard è uno degli aspetti più critici per garantire un'esperienza utente soddisfacente e produttiva. La capacità di una dashboard di gestire il caricamento e l'aggiornamento dei dati in tempo reale, senza introdurre latenza o rallentamenti, è fondamentale per il suo successo. Ottimizzare le performance significa ridurre al minimo i tempi di attesa degli utenti durante la generazione e la visualizzazione dei grafici, così come garantire che la dashboard possa scalare efficacemente con l'aumento del volume di dati. Questo richiede non solo una progettazione efficiente del software, ma anche l'uso di tecnologie avanzate e l'adozione di strategie di ottimizzazione delle risorse di calcolo.

2.1.5 Integrazione con altri servizi

Per arricchire le funzionalità di una Web Dashboard, è fondamentale che questa sia in grado di integrarsi con altri servizi e piattaforme. L'integrazione consente agli utenti di accedere a una gamma più ampia di dati e funzionalità, potenziando la capacità della dashboard di fornire insight completi e contestualizzati. Le API, i web service e le integrazioni con piattaforme di terze parti sono elementi chiave che permettono alla dashboard di estendere le sue capacità, supportando una visione olistica delle operazioni aziendali e favorendo decisioni informate e tempestive.

2.1.6 Personalizzazione

La personalizzazione è un altro aspetto fondamentale che consente di adattare la Web Dashboard alle esigenze specifiche degli utenti e ai contesti di utilizzo. Offrire opzioni di personalizzazione per la visualizzazione dei dati e l'organizzazione dell'interfaccia può migliorare significativamente l'esperienza utente, rendendo la dashboard non solo più utile, ma anche più rilevante per le diverse funzioni aziendali. La capacità di personalizzare la dashboard in base ai ruoli, alle preferenze individuali e ai requisiti specifici dell'utente finale contribuisce a massimizzare l'efficacia e l'efficienza delle decisioni prese attraverso l'uso della dashboard.

2.1.7 Errori Comuni da Evitare

Nonostante l'importanza di questi aspetti, è essenziale evitare alcuni errori comuni nella progettazione e nello sviluppo di una Web Dashboard. Tra questi, l'uso eccessivo di elementi visivi, che può sovraccaricare l'utente e ridurre la chiarezza delle informazioni; la mancanza di una guida utente adeguata, che può confondere gli utenti e rendere difficoltoso l'uso della dashboard; la scelta inappropriata del tipo di grafico, che può distorcere o rendere difficile l'interpretazione dei dati; e il cattivo uso dei colori, che può compromettere la leggibilità e l'accessibilità della dashboard. Evitare questi errori è cruciale per garantire che la dashboard sia uno strumento efficace per l'analisi dei dati.

Quindi, considerare attentamente tutti questi aspetti durante la progettazione e lo sviluppo di una Web Dashboard è essenziale per garantire un'esperienza utente ottimale e il successo complessivo del progetto. La creazione di una dashboard che sia performante, sicura, intuitiva, e personalizzabile richiede non solo una profonda comprensione delle esigenze degli utenti, ma anche una solida competenza tecnica per implementare soluzioni che rispondano a queste esigenze in modo efficiente e scalabile.

2.2 Tool per creare Web Dashboard

Attualmente, il mercato offre una vasta gamma di soluzioni per la creazione di Web Dashboard dedicate all'analisi e alla gestione dei dati temporali. Queste soluzioni si distinguono per le loro caratteristiche, funzionalità e ambiti di applicazione. Tra le piattaforme più popolari e ampiamente adottate dagli utenti ci sono alcune che hanno saputo guadagnarsi una posizione di rilievo grazie alla loro capacità di fornire strumenti potenti e versatili per visualizzare e interpretare i dati in tempo reale. Di seguito andremo ad analizzare due delle soluzioni più famose.

2.2.1 Grafana

Grafana [1] è una piattaforma open-source rinomata per la visualizzazione e l'analisi dei dati, con un focus particolare sul monitoraggio dei sistemi e sull'analisi dei dati in tempo reale. La sua adozione è diffusa grazie alla capacità di creare dashboard interattive che integrano dati provenienti da una varietà di fonti diverse, offrendo agli utenti un controllo dettagliato e una visione completa delle loro metriche di interesse.

Una delle caratteristiche distintive di Grafana è la vasta gamma di opzioni di visualizzazione dei dati che mette a disposizione. Tra queste, si possono trovare grafici a linee, grafici a barre, mappe geospaziali, e molte altre tipologie di rappresentazioni grafiche. Inoltre, gli utenti hanno la possibilità di inserire del testo esplicativo e di creare variabili da utilizzare nelle diverse visualizzazioni, rendendo le dashboard non solo informative, ma anche altamente personalizzabili e adattabili alle specifiche esigenze di monitoraggio e analisi.

La flessibilità di Grafana è uno dei suoi punti di forza principali. La piattaforma consente una personalizzazione profonda, permettendo agli utenti di creare dashboard su misura che rispondono perfettamente alle loro necessità operative e di business. Questa flessibilità si traduce in un'ampia applicabilità della piattaforma, che viene utilizzata non solo per il monitoraggio dei sistemi e l'analisi dei log, ma anche per l'analisi delle prestazioni delle applicazioni e molte altre applicazioni critiche.

Tuttavia, Grafana presenta anche alcune limitazioni che ne possono ridurre l'efficacia, specialmente per gli utenti meno esperti. La configurazione iniziale e la personalizzazione delle dashboard possono infatti richiedere una curva di apprendimento significativa. Gli utenti devono acquisire familiarità con la piattaforma e con le sue numerose funzionalità, il che può rappresentare una barriera iniziale non trascurabile. Inoltre, l'installazione e la gestione di Grafana su larga scala possono richiedere risorse hardware consistenti e competenze tecniche avanzate, rendendo necessaria la presenza di personale specializzato per mantenere e ottimizzare le prestazioni del sistema.

Un ulteriore aspetto critico riguarda la necessità di conoscere query specifiche per sfruttare appieno alcune delle funzionalità avanzate offerte da Grafana. Questo può rappresentare un ostacolo per gli utenti che non hanno familiarità con i linguaggi di query supportati, limitando l'accesso alle funzionalità più potenti della piattaforma.

Un problema significativo di Grafana, particolarmente rilevante quando si tratta di gestire e analizzare grandi quantità di dati in tempo reale, è legato alla sua architettura di gestione dei dati. In Grafana, tutte le operazioni sui dati vengono effettuate sul server. Questo significa che ogni volta che un utente vuole eseguire un'operazione sui dati o quando è necessario aggiornare i dati da visualizzare, il client deve inviare una richiesta al server. Il server esegue i calcoli necessari e invia i

risultati al client per la visualizzazione. Questo processo aumenta inevitabilmente il tempo di risposta della dashboard all'interazione dell'utente. Quando le operazioni da eseguire sono numerose e i dati da elaborare sono molti, questo meccanismo può causare un significativo rallentamento, fino a portare a uno stallo della dashboard.

Un ulteriore limite di Grafana nell'analisi dei dati in tempo reale è dato dal tempo minimo di aggiornamento automatico dei grafici, che è di 5 secondi. Sebbene questo possa sembrare un intervallo breve, in contesti dove la velocità di aggiornamento è critica, come nelle applicazioni finanziarie o nel monitoraggio di sistemi con requisiti di alta frequenza, questa limitazione può risultare problematica. La necessità di aggiornamenti più frequenti potrebbe richiedere soluzioni alternative o personalizzazioni specifiche che vanno oltre le capacità standard di Grafana.

In conclusione, mentre Grafana offre una potente piattaforma per la visualizzazione e l'analisi dei dati, particolarmente efficace in molte applicazioni di monitoraggio e analisi, presenta anche alcune limitazioni significative. Queste limitazioni devono essere attentamente considerate quando si sceglie una soluzione per l'analisi e la gestione di grandi volumi di dati in tempo reale, poiché possono influenzare significativamente le prestazioni e l'efficacia complessiva della dashboard.

2.2.2 Metabase

Metabase [2] è una piattaforma open-source innovativa progettata per semplificare l'analisi dei dati e la creazione di dashboard. Con un orientamento specifico verso gli utenti non tecnici, Metabase si propone di democratizzare l'accesso all'analisi dei dati, rendendola accessibile a tutti, indipendentemente dalla loro formazione tecnica. La facilità d'installazione, che richiede solo la presenza di Java, e l'intuitività d'uso fanno di Metabase uno strumento ideale anche per coloro che non hanno competenze avanzate in programmazione o gestione dei dati. È importante notare che, oltre alla versione open-source, che è gratuita e altamente flessibile, esiste anche una versione online a pagamento che offre ulteriori funzionalità e supporto.

Metabase offre un'interfaccia utente intuitiva e facile da usare, con una varietà di opzioni per l'analisi dei dati e la creazione di dashboard. Questa interfaccia è progettata per essere accessibile anche a chi non ha esperienza pregressa con strumenti di analisi dei dati, facilitando la creazione di visualizzazioni significative con pochi clic. Il supporto per una vasta gamma di fonti dati, inclusi database SQL, file CSV, query personalizzate e altro ancora, rende Metabase un'opzione versatile per molte organizzazioni. Tuttavia, il supporto per le fonti dati è più limitato rispetto ad alcune soluzioni concorrenti.

La piattaforma è altamente personalizzabile e offre diverse opzioni per la condivisione dei risultati all'interno del team. Questo è particolarmente utile nelle organizzazioni che necessitano di un'analisi dei dati self-service, dove i membri del team possono creare report ad hoc e condividere le loro analisi con altri, favorendo

una cultura aziendale basata sui dati. La possibilità di condividere facilmente dashboard e risultati analitici contribuisce a migliorare la collaborazione e a prendere decisioni più informate e tempestive.

Un altro punto di forza di Metabase è la sua comunità attiva di utenti e sviluppatori, che contribuiscono al continuo sviluppo della piattaforma. Questa comunità fornisce supporto, crea nuovi plugin e migliora costantemente le funzionalità di Metabase, garantendo che la piattaforma rimanga aggiornata con le ultime tendenze e necessità del mercato.

Tuttavia, Metabase presenta anche alcune limitazioni. Potrebbe essere meno flessibile o adattabile per casi d'uso avanzati rispetto ad altre soluzioni più robuste. In particolare, le prestazioni di Metabase possono risultare inferiori quando si tratta di gestire grandi volumi di dati. Come molte altre soluzioni, Metabase esegue le operazioni sui dati sul server e invia i risultati al client solo per la visualizzazione. Questo approccio, come affermato in precedenza, può comportare tempi di risposta più lunghi, soprattutto quando si lavora con dataset di grandi dimensioni o quando si effettuano operazioni complesse.

Un ulteriore svantaggio di Metabase è rappresentato dal tempo minimo di aggiornamento automatico dei dati, che è di un minuto. Questo può essere un problema in contesti dove è necessario avere dati aggiornati in tempo reale o quasi. Inoltre, Metabase non offre la possibilità di visualizzare immediatamente l'ultimo valore caricato nelle serie temporali, il che può limitare l'efficacia della piattaforma in applicazioni che richiedono monitoraggio continuo e aggiornamenti frequenti.

In conclusione, Metabase rappresenta una soluzione potente e accessibile per l'analisi dei dati e la creazione di dashboard, particolarmente adatta per utenti non tecnici e per organizzazioni che necessitano di strumenti di analisi self-service. Tuttavia, per applicazioni avanzate o per la gestione di grandi volumi di dati in tempo reale, potrebbero essere necessarie soluzioni più specializzate o ulteriori personalizzazioni.

2.3 Database

I database rappresentano una componente fondamentale di qualsiasi sistema informativo moderno. La loro funzione principale è quella di archiviare, gestire e recuperare grandi quantità di dati in modo efficiente e sicuro. Nel corso degli anni, i database si sono evoluti significativamente per rispondere alle crescenti esigenze delle applicazioni moderne, che richiedono performance elevate, scalabilità e flessibilità. Tra i vari tipi di database esistenti, quelli relazionali, NoSQL e NewSQL sono i più diffusi, ciascuno progettato per specifici scenari di utilizzo.

Una delle principali differenze nei database relazionali è il modello di organizzazione dei dati, che influisce direttamente sulle prestazioni del sistema in base

al tipo di operazioni richieste. Esistono due modelli fondamentali: i database organizzati per righe (row-oriented) e i database organizzati per colonne (column-oriented). Questi modelli si differenziano per il modo in cui i dati vengono archiviati e recuperati, ciascuno dei quali è ottimizzato per determinati tipi di carichi di lavoro.

2.3.1 Database organizzati per righe

I database organizzati per righe archiviano tutti i dati di una singola riga in un blocco continuo di memoria. Questo li rende particolarmente efficienti per operazioni che accedono frequentemente a molte colonne di una singola riga, come gli aggiornamenti o la consultazione di singoli record. Questi database sono comunemente utilizzati in scenari transazionali, come i sistemi di gestione delle vendite o i sistemi bancari, dove le operazioni CRUD (Create, Read, Update, Delete) quindi creazione, lettura, aggiornamento e cancellazione sono predominanti.

2.3.2 Database organizzati per colonne

D'altro canto, i database organizzati per colonne immagazzinano i dati per ciascuna colonna in blocchi separati. Questa struttura è ideale per operazioni analitiche e per la lettura di grandi volumi di dati che coinvolgono un sottoinsieme di colonne, come le query che eseguono aggregazioni o calcoli su specifici attributi. I database column-oriented sono quindi spesso utilizzati in applicazioni di data warehousing e analytics, dove l'efficienza nella lettura di dati da specifiche colonne è cruciale.

Capitolo 3

Architettura e Sviluppo del Framework Prototipale

In questo capitolo viene descritta l'architettura e lo sviluppo del framework prototipale realizzato nell'ambito di questa tesi. Il framework sfrutta tecnologie avanzate come WebAssembly e Rust per migliorare le prestazioni nella gestione e analisi di grandi volumi di dati temporali in tempo reale, spostando l'elaborazione direttamente sul client.

Il capitolo inizierà con una panoramica sui requisiti funzionali e tecnici richiesti per lo sviluppo della Web Dashboard. Saranno evidenziati i vantaggi di eseguire i calcoli direttamente sul client rispetto al server, soprattutto per quanto riguarda la riduzione della latenza e il miglioramento delle performance, specialmente in contesti di gestione di dati in tempo reale.

Successivamente, verranno approfondite le tecnologie utilizzate nel framework. La sezione seguente si concentrerà su WebAssembly (Wasm)[3], evidenziandone i vantaggi per eseguire codice a velocità quasi nativa nel browser. Si analizzeranno anche i principi di efficienza e portabilità che rendono Wasm ideale per applicazioni web ad alte prestazioni.

Dopo Wasm, sarà trattato il ruolo di Rust, scelto per la sua combinazione di alte prestazioni e sicurezza nella gestione della memoria. Verrà spiegato come il modello di proprietà di Rust e il controllo sicuro della concorrenza abbiano contribuito alla realizzazione di una soluzione robusta e performante, particolarmente adatta all'integrazione con WebAssembly.

Seguirà una sezione dedicata al protocollo WebSocket, utilizzato per garantire comunicazioni in tempo reale tra client e server. Verranno discussi i vantaggi di WebSocket rispetto alle chiamate HTTP asincrone tradizionali e come questo approccio migliora la reattività della dashboard, permettendo aggiornamenti immediati.

Inoltre, sarà analizzata l'integrazione di React, una libreria JavaScript utilizzata per la costruzione di un'interfaccia utente modulare e interattiva. Grazie alla sua gestione efficiente del rendering e al modello basato sui componenti, React si integra perfettamente con WebAssembly e Rust, garantendo un'interfaccia dinamica e scalabile.

Successivamente, verrà descritta l'implementazione concreta del framework, con un'analisi dell'architettura complessiva, delle scelte tecnologiche e delle soluzioni adottate per ottimizzare prestazioni e scalabilità. Verranno forniti dettagli su come le varie tecnologie siano state integrate per creare un sistema capace di gestire grandi volumi di dati temporali in tempo reale, garantendo un'esperienza utente fluida anche in scenari di carico elevato.

Infine, il capitolo concluderà con una guida dettagliata su come utilizzare il framework per la creazione di una Web Dashboard. Saranno illustrate le procedure per sviluppare e configurare una dashboard interattiva, con particolare attenzione alla modularità e alla riusabilità dei componenti.

3.1 Requisiti funzionali e tecnici

Lo sviluppo del framework per la Web Dashboard deve rispondere a una serie di requisiti funzionali e tecnici, che garantiscano prestazioni elevate, facilità d'uso e scalabilità, soprattutto nella gestione di grandi volumi di dati in tempo reale. In questa sezione vengono delineati i requisiti principali che guidano il design e l'implementazione del framework.

3.1.1 Requisiti funzionali

I requisiti funzionali descrivono le capacità che la dashboard deve offrire agli utenti finali. Questi requisiti sono stati definiti per assicurare che la Web Dashboard soddisfi le esigenze di interattività, prestazioni e facilità d'uso.

- **Prestazioni elevate con grandi quantità di dati**: La dashboard deve garantire tempi di risposta rapidi, anche quando gestisce ed elabora grandi volumi di dati in tempo reale. L'obiettivo è evitare rallentamenti, offrendo un'esperienza utente fluida e reattiva.

- **Analisi esplorativa dei dati temporali**: Gli utenti devono poter esplorare i dati in modo dinamico, utilizzando filtri per modificare le visualizzazioni e strumenti di aggregazione per sintetizzare le informazioni rilevanti. La dashboard deve consentire l'analisi da diverse prospettive e livelli di dettaglio.

- **Visualizzazione di tendenze e Rilevazione di anomalie**: Deve essere possibile individuare pattern e anomalie nei dati temporali, supportando la presa di decisioni informate.
- **Aggiornamenti in tempo reale**: La dashboard deve aggiornarsi automaticamente con i dati più recenti, senza che l'utente debba ricaricare la pagina. Questo è particolarmente importante in scenari dove le informazioni in tempo reale sono fondamentali, come il monitoraggio finanziario o la gestione di sensori IoT.
- **Interattività e Personalizzazione**: Gli utenti devono essere in grado di interagire facilmente con la dashboard, modificando parametri come filtri, grafici o impostazioni di visualizzazione in modo rapido e intuitivo.
- **Facilità d'uso**: L'interfaccia deve essere user-friendly, riducendo al minimo la curva di apprendimento per nuovi utenti. Un design pulito e organizzato, insieme a comandi facilmente accessibili, è essenziale per raggiungere questo obiettivo.
- **Scalabilità dei grafici e della dashboard**: La dashboard deve essere in grado di adattarsi a set di dati in crescita senza compromettere le prestazioni, garantendo la fluidità anche con un aumento del carico di dati o della complessità delle visualizzazioni.

3.1.2 Requisiti tecnici

I requisiti tecnici descrivono le tecnologie e gli strumenti necessari per implementare le funzionalità descritte nei requisiti funzionali. Questi requisiti garantiscono che la dashboard sia progettata in modo efficiente e scalabile.

- **Uso di Rust e WebAssembly (Wasm)**: Il framework deve essere costruito utilizzando Rust per le sue prestazioni elevate e la sicurezza nella gestione della memoria. WebAssembly (Wasm) consentirà di eseguire codice nel browser a velocità quasi nativa, migliorando l'efficienza delle operazioni computazionali direttamente sul client.
- **Esecuzione di calcoli complessi nel browser**: Per evitare ritardi dovuti a comunicazioni con il server, la dashboard deve eseguire calcoli intensivi direttamente nel browser, grazie a WebAssembly, riducendo così la latenza e migliorando la reattività.
- **Integrazione con WebSocket per aggiornamenti in tempo reale**: La dashboard deve supportare la comunicazione in tempo reale con il server utilizzando WebSocket, garantendo che i dati più recenti siano sempre visualizzati senza richiedere un intervento manuale dell'utente.

- **Gestione efficiente di grandi flussi di dati**: La dashboard deve essere in grado di processare e visualizzare grandi volumi di dati temporali in modo efficiente. Il framework deve ottimizzare la gestione della memoria e delle risorse per mantenere elevate prestazioni anche in presenza di dataset estesi.

- **Integrazione con React per l'interfaccia utente**: React sarà utilizzato per la creazione di un'interfaccia modulare e interattiva. La gestione efficiente del rendering e la modularità di React consentiranno una maggiore riusabilità dei componenti, facilitando lo sviluppo e la manutenzione della dashboard.

- **Ottimizzazione delle visualizzazioni grafiche**: Le visualizzazioni grafiche dovranno essere ottimizzate per garantire rapidità e reattività, sfruttando tecniche di rendering avanzato per rappresentare dati in tempo reale senza rallentamenti.

3.2 WebAssembly e Rust

WebAssembly è una tecnologia che sta trasformando lo sviluppo delle moderne applicazioni web, permettendo di eseguire codice compilato da linguaggi come C, C++ e Rust direttamente nel browser, con prestazioni vicine a quelle native.[4] Una delle sue principali caratteristiche è l'interoperabilità con JavaScript, che consente di integrare moduli WebAssembly in applicazioni web tradizionali.[5] Nonostante i meccanismi di sicurezza di WebAssembly, come la protezione della memoria, alcune vulnerabilità come gli attacchi side-channel evidenziano l'importanza di una programmazione attenta e della scelta di linguaggi che favoriscono la sicurezza.[6]

Come affermato in precedenza alcuni problemi legati alla sicurezza possono essere risolti a seconda del linguaggio di programmazione scelto tra quelli supportati dalla tecnologia. I tre più diffusi sono C, C++ e Rust.

Facendo un confronto tra questi linguaggi di programmazione attraverso le due tesi di laurea[7][8] è possibile notare che il migliore tra i tre risulta Rust che si distingue nettamente.

Rust è un linguaggio moderno progettato per garantire alte prestazioni e sicurezza senza compromessi. Una delle sue caratteristiche principali è il "borrow checker", un sistema di gestione della memoria che impedisce problemi comuni come i memory leaks e le condizioni di race, offrendo al contempo un'esecuzione efficiente senza il bisogno di un garbage collector. Questo lo rende particolarmente adatto per applicazioni che devono processare grandi quantità di dati in tempo reale, come nel caso delle Web Dashboard interattive.[9]

Rust è inoltre ottimizzato per l'esecuzione parallela, permettendo di sfruttare appieno i moderni processori multi-core, e combina la sicurezza tipica dei linguaggi ad alto livello con le prestazioni dei linguaggi di basso livello come C e C++. Grazie

a queste caratteristiche, Rust non solo riduce gli errori legati alla gestione della memoria, che possono essere critici in WebAssembly, ma offre anche una base solida per la costruzione di software robusto, scalabile e performante.

Per ottimizzare ulteriormente la gestione dei dati, nel nostro progetto utilizziamo Polars, una libreria per DataFrame ad alte prestazioni scritta in Rust.[10]
L'obiettivo di Polars è quello di fornire una libreria DataFrame velocissima che:

- utilizza tutti i core disponibili nella macchina
- ottimizza le query per ridurre inutili allocazioni di memoria
- gestisce dataset con dimensioni maggiori della memoria RAM disponibile sulla macchina

Alcune caratteristiche fondamentali di Polars infatti sono:

- **Velocità**: scritto da zero in Rust, progettato vicino alla macchina e senza dipendenze esterne.
- **API intuitiva**: Polars, internamente, determinerà il modo più efficiente per eseguire la query scritta utilizzando il suo ottimizzatore di query.
- **Out of Core**: l'API di streaming ti consente di elaborare i risultati senza richiedere che tutti i dati siano in memoria contemporaneamente.
- **Parallelizzazione**: utilizza la potenza della macchina su cui è in esecuzione suddividendo il carico di lavoro tra i core CPU disponibili senza configurazioni aggiuntive.
- **Motore di query velocizzato**: utilizza Apache Arrow, un formato di dati colonnare, per elaborare le query in modo vettorizzato e SIMD (Single Instruction Multiple Data) per ottimizzare l'uso della CPU.

3.3 WebSocket

Nel contesto dell'analisi dei dati temporali, garantire una comunicazione in tempo reale tra client e server è cruciale. Per questo utilizziamo WebSocket, un protocollo bidirezionale che consente una connessione persistente e a bassa latenza, ideale per applicazioni che richiedono aggiornamenti continui e immediati.

WebSocket stabilisce un canale di comunicazione TCP bidirezionale che permette al server di inviare dati al client senza esplicite richieste, riducendo la latenza e migliorando l'esperienza utente. Questa caratteristica è particolarmente utile per visualizzare dati in tempo reale, come dati finanziari, metriche di monitoraggio o informazioni da sensori IoT.

Oltre alla sua efficienza nella gestione di grandi volumi di dati, WebSocket riduce l'overhead rispetto ad altre tecnologie, mantenendo una connessione persistente. Inoltre, offre meccanismi di gestione degli errori e riconnessione automatica, garantendo affidabilità anche in presenza di interruzioni di rete.

In sintesi, WebSocket è una soluzione ideale per applicazioni che richiedono comunicazioni in tempo reale e aggiornamenti continui, migliorando sia le prestazioni che l'interattività.

3.4 React

React[11] è una delle librerie JavaScript più popolari per la creazione di interfacce utente dinamiche e reattive. Introdotta da Facebook nel 2013, React ha rivoluzionato lo sviluppo web grazie al suo approccio basato sul Virtual DOM, che consente aggiornamenti efficienti dell'interfaccia utente, riducendo le manipolazioni dirette del DOM e migliorando le performance.

Un concetto chiave di React è la componentizzazione, che permette di suddividere l'interfaccia in componenti riutilizzabili e indipendenti. Questo approccio semplifica lo sviluppo, la manutenzione e la scalabilità delle applicazioni. La riusabilità dei componenti consente inoltre di costruire interfacce complesse in modo più efficiente e coerente.

React si integra facilmente con altre librerie, come D3.js[12], per la visualizzazione di dati dinamici. D3.js offre potenti strumenti per creare grafici interattivi e aggiornabili in tempo reale. La combinazione di React e D3 sfrutta le capacità di aggiornamento del Virtual DOM di React insieme alle funzionalità di visualizzazione di D3, ottimizzando le performance delle Web Dashboard.

Infine, con l'introduzione dei Hooks nella versione 16.8, React ha semplificato la gestione dello stato e degli effetti collaterali nei componenti funzionali, rendendo il codice più modulare e facile da mantenere.

In sintesi, React è ideale per la costruzione di interfacce interattive e performanti, specialmente quando integrato con librerie come D3.js per gestire grandi volumi di dati in tempo reale.

3.5 Implementazione

In questa sezione verrà approfondito il processo di creazione del framework prototipale, evidenziando l'approccio modulare utilizzato per suddividere le diverse componenti del sistema. Si illustreranno i passaggi che hanno portato alla progettazione di una struttura flessibile e scalabile, in grado di essere estesa o modificata con facilità. Particolare attenzione sarà posta all'integrazione tra i moduli per

la gestione dei dati, l'analisi e la visualizzazione, e su come queste componenti interagiscono tra loro in modo sinergico all'interno del framework.

3.5.1 Architettura Modulare

Con l'aumento della complessità delle applicazioni moderne e la necessità di garantire un utilizzo ottimale delle risorse, l'adozione di un'architettura modulare si rivela cruciale. Tale approccio si basa sulla scomposizione delle funzionalità in componenti indipendenti e riutilizzabili, facilitando la manutenzione, l'estensibilità e l'integrazione di nuovi moduli. Inoltre, promuove una migliore collaborazione tra i diversi team di sviluppo, poiché ogni componente può essere sviluppato e gestito separatamente, riducendo il rischio di conflitti.

L'architettura modulare offre numerosi vantaggi che risultano particolarmente utili nelle applicazioni moderne:

- **Manutenibilità**: La suddivisione in moduli facilita l'identificazione e la risoluzione di eventuali problematiche. Se un modulo necessita di correzioni, le modifiche possono essere apportate senza influire sugli altri componenti del sistema.
- **Scalabilità**: Grazie alla modularità, è possibile aggiungere nuove funzionalità senza necessità di riprogettare l'intero framework. I nuovi moduli possono essere integrati senza compromettere quelli esistenti.
- **Testabilità**: Ogni modulo può essere testato separatamente, garantendo che le singole componenti funzionino correttamente prima di essere integrate. Questo approccio migliora la qualità del codice e riduce il rischio di errori.

Nell'architettura modulare l'interoperabilità tra i moduli è garantita da interfacce ben definite che permettono una comunicazione chiara e strutturata. Questo approccio consente ai diversi moduli di operare in modo indipendente, riducendo il livello di accoppiamento tra le parti. Ad esempio, il modulo di visualizzazione è progettato per ricevere i dati elaborati tramite i servizi di comunicazione, senza dipendere direttamente dalle operazioni interne di elaborazione dei dati.

Un'altra caratteristica chiave di questa architettura è la riutilizzabilità dei moduli. Ogni componente può essere utilizzato in diversi contesti e scenari applicativi. Ad esempio, il modulo di visualizzazione può essere esteso per supportare nuovi tipi di grafici senza la necessità di modificare l'intero sistema. Questo approccio favorisce lo sviluppo rapido e l'adattamento a nuovi requisiti.

L'approccio modulare non solo favorisce una struttura più organizzata e gestibile, ma ha anche un impatto positivo sulle prestazioni. Il modulo di elaborazione, sviluppato utilizzando WebAssembly e Rust, è in grado di eseguire operazioni

computazionali complesse direttamente nel browser, garantendo una velocità di esecuzione molto vicina a quella nativa e riducendo drasticamente i tempi di latenza.

Infine, la modularità facilita la gestione delle dipendenze. Ogni modulo può integrare librerie e strumenti specifici senza influire sugli altri moduli. Ad esempio, il modulo di elaborazione può utilizzare librerie come Polars per l'analisi dei dati, mentre il modulo di visualizzazione sfrutta D3.js per la creazione di grafici, garantendo così una struttura flessibile e altamente configurabile.

Nel framework prototipale che abbiamo realizzato, l'architettura modulare, che è possibile vedere anche nella figura 3.1 , è stata articolata in quattro componenti principali:

- **Modulo di Caricamento Dati**: Carica e prepara i dati provenienti da varie fonti. Invia i dati al modulo WASM per l'elaborazione e li passa direttamente ai componenti di visualizzazione.
- **Componenti di visualizzazione**: responsabili della rappresentazione dei dati attraverso grafici interattivi, permettendo una comprensione immediata delle informazioni.
- **Modulo WebSocket**: utilizzando WebSocket per una comunicazione bidirezionale efficiente tra client e server, garantendo aggiornamenti in tempo reale.
- **Modulo WASM**: sfruttando la potenza di WebAssembly e Rust, questo modulo gestisce l'elaborazione efficiente dei dati, migliorando le prestazioni senza compromettere la sicurezza e la scalabilità del sistema.

Questa suddivisione permette al framework di rimanere flessibile e scalabile, facilitando l'aggiunta di nuove funzionalità e miglioramenti futuri senza dover ripensare l'intera architettura.

3.5.2 Modularizzazione dei componenti di visualizzazione

Nel processo di modularizzazione, uno degli aspetti chiave è la separazione della logica di rendering dei grafici dal resto del codice dell'applicazione. In particolare, la libreria D3.js offre un'ampia gamma di strumenti per il rendering di grafici, ma spesso le sue funzionalità vengono integrate direttamente nel codice dell'interfaccia, creando accoppiamenti stretti e limitando la riusabilità.

Nel nostro framework, per ogni componente di visualizzazione, abbiamo adottato un approccio di astrazione, scomponendo le operazioni di D3.js in funzioni specifiche e riutilizzabili. La logica di rendering viene separata in funzioni che eseguono singoli compiti, come:

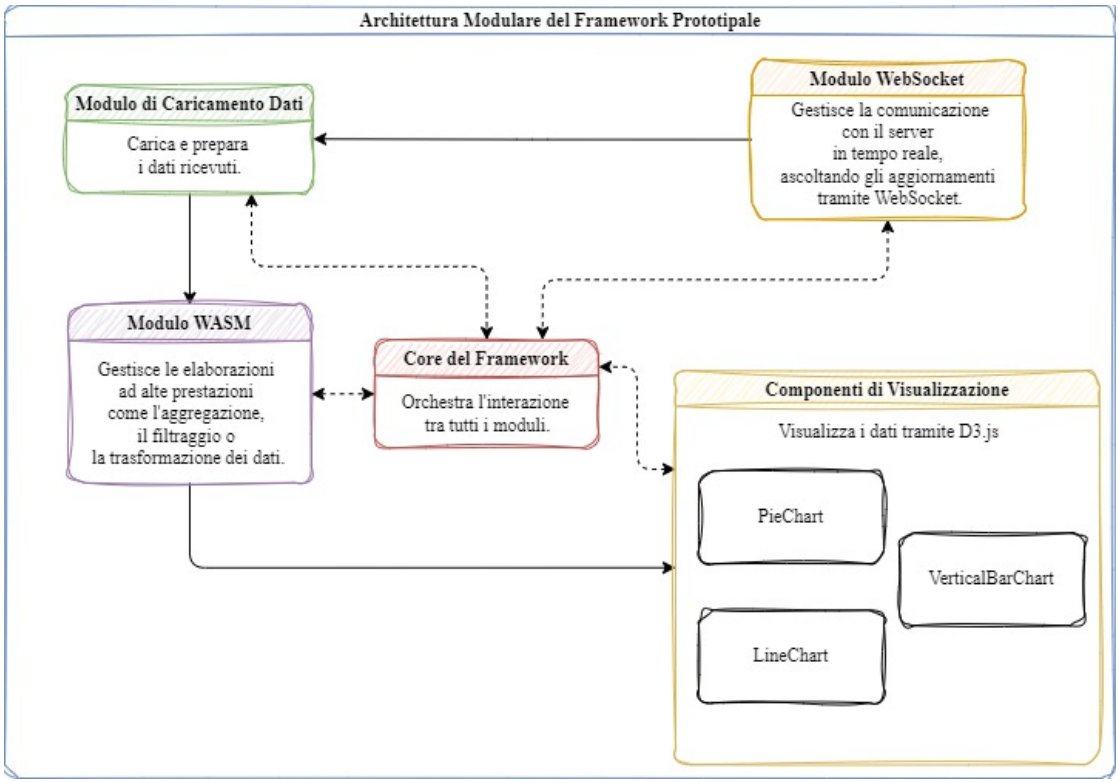


Figura 3.1: Architettura modulare del framework prototipale

- **Creazione degli assi**: Funzioni che generano gli assi (x e y) configurabili in base ai dati ricevuti.
- **Disegno delle linee o delle forme**: Funzioni che tracciano linee o curve su un piano, prendendo in input i dati grezzi.
- **Gestione delle interazioni**: Funzioni per aggiungere interattività, come il tooltips o il comportamento al passaggio del mouse.

Questa separazione rende il codice più pulito e facilmente manutenibile, permettendo di riutilizzare le stesse funzioni in altri grafici con una configurazione minima.

Per garantire la personalizzazione e riusabilità dei componenti di visualizzazione, ogni grafico è stato progettato per essere altamente configurabile tramite parametri dinamici. Abbiamo definito una serie di props (proprietà) che permettono di adattare il comportamento e l'aspetto dei grafici in base alle esigenze specifiche del progetto.

Questa configurabilità assicura che i componenti grafici possano essere facilmente integrati in altri progetti senza dover riscrivere la logica di visualizzazione.

3.5.3 Implementazione dei Servizi di Comunicazione con WebSocket

La gestione della comunicazione in tempo reale tra il client e il server è stata realizzata tramite l'uso di WebSocket. Tuttavia, anziché incorporare direttamente la logica di WebSocket in singoli componenti, è stato creato un servizio dedicato. Questa classe o servizio astrae le operazioni comuni di WebSocket in un'implementazione generica, gestendo la connessione, l'invio e la ricezione dei dati in modo centralizzato.

La classe creata include funzioni che permettono:

- **Connessione al server WebSocket**: Gestisce la creazione della connessione, l'invio delle richieste di apertura e la gestione degli eventi.
- **Ricezione dei messaggi**: Ascolta i messaggi provenienti dal server e invia aggiornamenti ai componenti interessati tramite callback o meccanismi di osservazione.
- **Riconnessione automatica**: Implementa una logica di riconnessione automatica in caso di disconnessione improvvisa, migliorando la resilienza della comunicazione.
- **Invio di messaggi**: Consente l'invio di messaggi al server WebSocket, con una gestione ottimizzata dei formati dati.

In questo modo, viene gestita la connessione WebSocket e viene inoltre fornita un'interfaccia chiara per inviare e ricevere dati, oltre alla gestione degli eventi di riconnessione in modo trasparente. È inoltre possibile registrare diverse callback per gestire messaggi specifici basati su un tipo di evento.

Grazie all'astrazione fornita il sistema diventa altamente flessibile e riutilizzabile. Questa struttura non è legata a una singola applicazione o specifico flusso di dati, il che significa che può essere facilmente adottata in altri progetti che richiedono una comunicazione in tempo reale semplicemente cambiando l'URL del server WebSocket e i callback per i diversi tipi di messaggi.

La flessibilità si manifesta anche nella possibilità di aggiungere facilmente nuovi tipi di eventi o di estendere la logica di gestione, poiché la separazione tra la logica di connessione e la gestione dei dati è ben definita.

La modularizzazione e l'astrazione del servizio WebSocket attraverso questa classe offre notevoli vantaggi in termini di flessibilità, manutenibilità e riusabilità del codice. L'approccio descritto permette a questo servizio di essere facilmente adottato in altre applicazioni, semplificando la gestione della comunicazione in tempo reale e accelerando lo sviluppo di nuove funzionalità.

3.5.4 Integrazione di WebAssembly e Apache Arrow per l'Elaborazione dei Dati

Apache Arrow è una piattaforma di gestione e serializzazione di dati progettata per lavorare con grandi quantità di dati tabellari in modo efficiente, riducendo i colli di bottiglia legati alla manipolazione e alla trasformazione dei dati in memoria.

Nel nostro framework, Apache Arrow è utilizzato per rappresentare i dati in un formato compatto e colonnare, ottimizzato per operazioni di calcolo intensive. Il formato binario di Arrow permette di ridurre l'overhead di serializzazione/deserializzazione durante lo scambio di dati tra il client (browser) e il server, nonché nelle comunicazioni interne con WebAssembly (WASM). Questo approccio consente una maggiore efficienza rispetto ai tradizionali formati basati su stringhe, come JSON o CSV, poiché Arrow è progettato per accedere ai dati direttamente in memoria in modo rapido.

Nel contesto del nostro framework:

- I dati tabellari sono gestiti con Arrow Table, che permette di manipolare e visualizzare grandi dataset con minimi costi di memoria.

- Apache Arrow è compatibile con diversi linguaggi di programmazione e, una volta serializzato il formato Arrow, è facilmente trasferibile e interpretabile in WebAssembly e Rust.

WebAssembly (WASM) gioca un ruolo centrale nel framework per eseguire calcoli intensivi direttamente nel browser, riducendo i tempi di latenza e migliorando le prestazioni. L'integrazione di WASM consente di delegare operazioni computazionali pesanti, come la trasformazione di grandi volumi di dati o il calcolo statistico, a un codice nativo compilato in linguaggi ad alte prestazioni come Rust, mantenendo comunque l'interfaccia con il codice JavaScript esistente.

Nel nostro caso:

- Rust viene utilizzato per scrivere funzioni critiche di elaborazione dei dati, che vengono poi compilate in WebAssembly. Il vantaggio di Rust è la sua compatibilità con WebAssembly, oltre alla sua efficienza in termini di gestione della memoria e sicurezza del calcolo.
- Il codice WASM esegue le trasformazioni sui dataset gestiti da Apache Arrow in modo estremamente rapido, operando direttamente sui buffer di memoria che rappresentano i dati.

Questo approccio permette di caricare nel browser anche dataset di grandi dimensioni senza perdere in termini di reattività o performance. In particolare, grazie alla struttura colonnare di Apache Arrow, le operazioni su specifiche colonne di dati, come il calcolo di medie o aggregati, sono molto più rapide.

Ecco una panoramica di come funziona il flusso che è possibile vedere in modo più approfondito nella figura 3.2:

1. **Prelievo dei dati dal database**: I dati vengono prelevati dal database dal server, pronto per essere inviato al client.
2. **Invio dei dati tramite WebSocket**: Il server invia i dati al client utilizzando una connessione WebSocket, garantendo una trasmissione in tempo reale.
3. **Ricezione e conversione dei dati sul client**: Il client riceve i dati e li converte in formato Apache Arrow tramite il modulo di caricamento dati.
4. **Elaborazione WASM**: Il modulo WebAssembly (WASM) sul client elabora i dati Arrow, eseguendo trasformazioni o calcoli come filtraggio e statistiche.
5. **Visualizzazione dei dati elaborati**: I dati elaborati vengono passati ai componenti di visualizzazione del client, che si occupano di mostrarli tramite grafici interattivi.

L'integrazione di Apache Arrow e WebAssembly con Rust offre un potente strumento per l'elaborazione di grandi quantità di dati in modo efficiente direttamente nel browser. L'utilizzo di Apache Arrow riduce l'overhead associato alla gestione dei dati e consente di eseguire operazioni intensive con WASM senza impattare negativamente sulle performance dell'applicazione o sull'esperienza dell'utente.

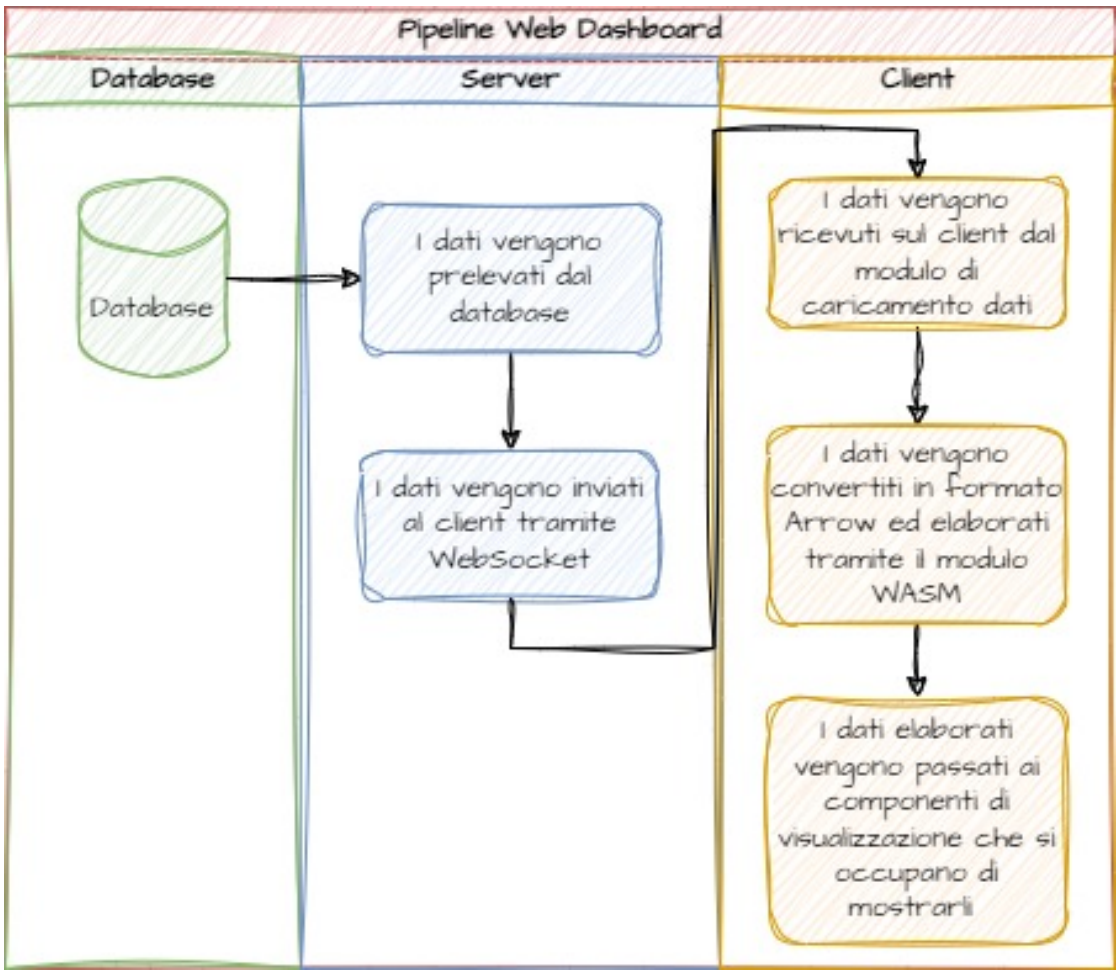


Figura 3.2: Architettura della Pipeline di una Web Dashboard sviluppata con il Framework Prototipale

3.6 Utilizzo del framework per sviluppare una Web Dashboard

La Web Dashboard sviluppata è stata realizzata sfruttando il framework prototipale, con particolare attenzione alla modularità e alla riusabilità dei componenti. Questo framework ha consentito di costruire una soluzione scalabile, con singoli moduli dedicati rispettivamente alla visualizzazione dei dati, alla comunicazione in tempo reale e all'elaborazione di grandi dataset. Vediamo nel dettaglio come questi moduli sono stati usati nell'implementazione della Web Dashboard.

In particolare il framework realizzato è stato utilizzato per la parte relativa al client, per creare i grafici, per la comunicazione con il server e per l'elaborazione dei dati.

Per quanto riguarda la creazione dei grafici sono state sfruttate le API del framework andando a passare delle props specifiche per la realizzazione dei grafici voluti.

Per quanto riguarda la gestione della comunicazione in tempo reale tramite WebSocket nella creazione della Web Dashboard, il WebSocketService è stato il componente centrale per l'integrazione di flussi di dati in tempo reale tra il client (la dashboard) e il server. Questo servizio è stato sviluppato per astrarre tutte le operazioni relative alla comunicazione in tempo reale, semplificando l'implementazione e mantenendo il codice modulare e riutilizzabile.

Il primo passo nella creazione della Web Dashboard è stato configurare la connessione WebSocket. All'interno della dashboard, il WebSocketService è stato istanziato all'avvio dell'applicazione e configurato per connettersi a un endpoint del server in grado di inviare dati in tempo reale attrverso il metodo *connect()* che gestisce l'apertura della connessione con il server WebSocket e assicura che la dashboard sia pronta per ricevere dati in tempo reale non appena il server li invia.

Una volta stabilita la connessione, viene gestita la ricezione dei dati attraverso il metodo *onMessage()*. Ogni volta che il server invia un aggiornamento, il servizio cattura il messaggio e lo invia alla dashboard, che aggiorna automaticamente i grafici e altre visualizzazioni dei dati.

Questa logica ha reso possibile il continuo aggiornamento dei componenti di visualizzazione della dashboard, come grafici a linee o barre, senza dover ricaricare la pagina. Il servizio invia automaticamente i nuovi dati ricevuti ai componenti grafici, che vengono ridisegnati.

Uno degli obiettivi principali della Web Dashboard era fornire grafici interattivi che mostrassero i dati in tempo reale. Grazie all'astrazione della logica di comunicazione WebSocket, l'aggiornamento in tempo reale dei dati è stato semplificato. In questo modo, la dashboard è in grado di visualizzare dati continuamente aggiornati senza la necessità di un'ulteriore interazione dell'utente.

Nel contesto della comunicazione in tempo reale, la stabilità della connessione WebSocket è fondamentale. Per questo motivo, il WebSocketService ha implementato meccanismi di gestione degli errori e di riconnessione automatica. Se la connessione si interrompe o si verifica un errore, il servizio tenta automaticamente di riconnettersi senza intervento manuale, garantendo che i dati in tempo reale continuino a fluire senza interruzioni.

Un altro aspetto importante della gestione della comunicazione in tempo reale è l'aggiornamento dell'interfaccia utente per informare l'utente finale riguardo lo stato della connessione. Il WebSocketService invia notifiche alla dashboard per informare l'utente se la connessione è attiva o se si sono verificati problemi di rete. Queste notifiche migliorano l'esperienza utente, soprattutto in applicazioni dove l'affidabilità dei dati è cruciale.

Grazie all'uso di questo servizio, la dashboard è stata in grado di gestire flussi di dati continui in tempo reale, mantenendo un'interfaccia utente reattiva e interattiva. L'astrazione della logica WebSocket ha inoltre permesso di rendere il codice modulare e facilmente riutilizzabile in altre parti dell'applicazione o per altri progetti che necessitano di comunicazioni in tempo reale.

Questo approccio ha semplificato notevolmente lo sviluppo della dashboard, consentendo di concentrarsi sull'elaborazione e la visualizzazione dei dati ricevuti senza doversi preoccupare della gestione della connessione WebSocket.

L'integrazione di WebAssembly (WASM) e Rust è stata una componente essenziale della creazione della Web Dashboard, poiché ha permesso di eseguire operazioni computazionali intensive direttamente nel browser, mantenendo alti livelli di performance. Questo approccio è stato fondamentale per gestire grandi quantità di dati in modo efficiente e senza dover fare affidamento su server esterni per l'elaborazione, migliorando notevolmente la reattività dell'applicazione.

L'uso di WebAssembly ha consentito di eseguire codice ad alte prestazioni nel contesto del browser. WASM è stato scelto perché permette di eseguire codice compilato, come quello scritto in Rust, direttamente nel browser, superando i limiti prestazionali tradizionali di JavaScript per operazioni complesse.

Nella Web Dashboard, WASM è stato utilizzato principalmente per:

- **Trasformazione dei dati in tempo reale**: Grandi volumi di dati provenienti dai WebSocket dovevano essere filtrati, aggregati e trasformati rapidamente.
- **Esecuzione di algoritmi computazionali**: Alcuni algoritmi di analisi richiedono un'elevata potenza di calcolo che sarebbe stata troppo costosa da eseguire esclusivamente con JavaScript. Attraverso WASM, questi algoritmi sono stati eseguiti molto più rapidamente e in modo efficiente nel browser, senza rallentamenti visibili per l'utente.

La pipeline dati della dashboard è stata ottimizzata per inviare i dati ai moduli WASM scritti in Rust per l'elaborazione, prima di visualizzare i risultati finali.

Nel flusso di lavoro della dashboard quindi i dati provenienti dai WebSocket vengono trasformati in formati compatibili con Apache Arrow, e successivamente elaborati dal modulo WASM compilato da Rust. Una volta completata l'elaborazione, i risultati vengono restituiti a JavaScript per essere visualizzati nei grafici.

Questa pipeline assicura che i dati vengano processati in modo efficiente e senza rallentamenti percepibili dall'utente, anche quando i set di dati diventano molto grandi.

L'integrazione di WebAssembly e Rust nella dashboard ha permesso di ottenere notevoli miglioramenti nelle performance. Grazie all'uso di WASM per eseguire codice ad alte prestazioni e all'efficienza di Apache Arrow nella gestione dei dati tabellari, la dashboard è in grado di gestire flussi di dati complessi e di eseguire elaborazioni in tempo reale senza compromettere l'esperienza utente.

Per garantire un aggiornamento continuo e in tempo reale dei dati all'interno della Web Dashboard sviluppata, è stata implementata una soluzione basata su WebSocket, un protocollo che permette una comunicazione bidirezionale tra client e server. Questo server WebSocket è essenziale per trasmettere dati in tempo reale dal backend (dove i dati sono archiviati e aggiornati) al frontend, dove la dashboard visualizza graficamente queste informazioni. L'utilizzo di WebSocket è stato preferito rispetto ad altre tecniche, come le richieste HTTP periodiche, per garantire un aggiornamento efficiente e reattivo, riducendo il carico di rete e migliorando l'esperienza utente.

Il server WebSocket è implementato utilizzando Node.js ed è responsabile dell'acquisizione dei dati dal database e dell'invio al frontend. La struttura del server consente di gestire due scenari:

- **Integrazione con SQLite**: Per ambienti di test o piccoli dataset, il server invia periodicamente (ad esempio ogni secondo) i dati memorizzati nel database locale al client WebSocket. Questi dati vengono prelevati dal database SQLite attraverso una query SQL, elaborati e inviati in formato JSON al client connesso.

- **Integrazione con Clickhouse**: In questo caso, il server WebSocket esegue query su Clickhouse per ottenere i nuovi dati, li serializza in formato JSON e li trasmette al client.

Il codice gestisce l'intero ciclo della comunicazione, dalla connessione WebSocket alla ricezione e invio dei dati. Ogni volta che il server riceve nuovi dati dal database, questi vengono inviati immediatamente al client connesso, garantendo così che la dashboard visualizzi informazioni costantemente aggiornate.

Questa infrastruttura rappresenta un elemento fondamentale per la Web Dashboard, poiché consente di sfruttare al massimo le potenzialità del framework,

offrendo un flusso continuo di dati senza interruzioni e garantendo un'elevata reattività dell'interfaccia utente. Inoltre, l'architettura flessibile di WebSocket consente di adattare facilmente la soluzione a diversi tipi di applicazioni che richiedono comunicazioni in tempo reale, rendendola altamente riutilizzabile anche in contesti futuri.

Capitolo 4

Metodologia di Valutazione e Analisi delle Web Dashboard

In questo capitolo viene presentata la metodologia utilizzata per progettare e strutturare i test mirati a valutare le performance delle Web Dashboard. L'obiettivo è descrivere il processo di definizione dei test, dalla scelta degli strumenti e delle tecniche di testing fino alla configurazione degli ambienti di prova, senza entrare ancora nel dettaglio delle soluzioni sviluppate.

Verrà fornita una panoramica delle metriche chiave per la valutazione delle prestazioni, come velocità di caricamento, capacità di gestione di grandi volumi di dati e reattività in tempo reale. La metodologia proposta è stata studiata per garantire un confronto oggettivo tra le diverse soluzioni, assicurando la validità dei risultati.

Il capitolo si articola in due sezioni principali. La prima parte offre una panoramica generale sul concetto di Performance Testing, descrivendo le principali tecniche utilizzate per valutare le prestazioni di applicazioni web, come il load testing, lo stress testing e altri approcci. La seconda parte si concentra sulla metodologia specifica adottata per i test di questo progetto, illustrando gli strumenti utilizzati, la configurazione degli ambienti di test e le metriche chiave che verranno applicate nel capitolo successivo per confrontare le prestazioni delle dashboard sviluppate con Grafana, Metabase e il framework proposto.

Inoltre, si spiegherà l'importanza di includere diversi tipi di database e browser nel processo di testing, per valutare l'impatto di queste variabili sulle performance generali delle Web Dashboard.

4.1 Performance Testing

Per valutare in modo oggettivo le reali prestazioni di una Web Dashboard e nel nostro caso quindi fare delle valutazioni oggettive anche sulle prestazioni del tool con cui la dashboard è stata creata, è fondamentale eseguire test automatici mirati. Questi test consentono di identificare con precisione sia i punti di forza che le aree di criticità del software, offrendo una panoramica dettagliata delle sue capacità e limitazioni. Attraverso un'analisi sistematica delle performance, è possibile ottenere dati concreti che guidano eventuali ottimizzazioni e permettono un confronto accurato tra diverse soluzioni.

Oltre alla necessità di individuare punti di forza e criticità di un tool, il concetto di Performance Testing si inserisce in un contesto più ampio di metodologie e tecnologie ormai consolidate per la misurazione e l'ottimizzazione delle prestazioni. Negli ultimi anni, sono state sviluppate numerose tecniche e strumenti specifici che consentono di condurre test automatizzati su diverse piattaforme, con l'obiettivo di analizzare la risposta del sistema in termini di velocità, scalabilità e affidabilità, soprattutto in scenari di gestione di grandi volumi di dati in tempo reale.

Per comprendere a fondo come misurare l'efficacia di un sistema in termini di prestazioni, è necessario fare riferimento a una pratica fondamentale: il Performance Testing o Test delle Prestazioni. Questo approccio consente di valutare parametri cruciali come la velocità, la reattività e la stabilità di un sistema in un contesto operativo reale. Il Performance Testing non si limita a verificare il funzionamento del sistema, ma punta soprattutto a ottimizzare l'esperienza utente, identificando e risolvendo eventuali colli di bottiglia (bottleneck) che possono influire negativamente sulle prestazioni complessive. Nei paragrafi successivi, approfondiremo in dettaglio le tecniche e i test specifici che permettono di analizzare e migliorare continuamente il rendimento di un'infrastruttura IT.

La valutazione delle prestazioni di un tool richiede l'esecuzione di test automatici mirati, che permettano di identificare con precisione tanto i punti di forza quanto le aree di criticità del software. Questi test offrono una panoramica dettagliata delle capacità e delle limitazioni del sistema, fornendo dati concreti che possono guidare l'ottimizzazione e facilitare il confronto tra diverse soluzioni.

Il concetto di Performance Testing si inserisce in un panorama più ampio di metodologie e tecniche consolidate per la misurazione e l'ottimizzazione delle prestazioni. Negli ultimi anni, lo sviluppo di strumenti specifici ha reso possibile condurre test automatizzati su svariate piattaforme, con particolare attenzione alla velocità, scalabilità e affidabilità del sistema. Questi aspetti sono particolarmente critici quando si gestiscono grandi volumi di dati in tempo reale, contesto in cui ogni inefficienza può avere un impatto significativo sull'usabilità e sull'efficienza del sistema.

Il Performance Testing è composto da due metodi di riferimento: il test di

carico (load testing) e il test di picco (stress testing), che a loro volta si articolano in ulteriori sotto test che analizzeremo in seguito, unitamente alle valutazioni orientate nello specifico ad osservare aspetti come la scalabilità e la capacità di carico complessiva dell'applicazione stessa. Un Performance Testing è soggetto alla definizione a monte di alcune metriche, i key performance indicator (KPI), indispensabili per poter avere dei riferimenti utili per confrontare i risultati ottenuti. Oltre ai due tipi di performance testing citati in precedenza ne esistono altri tipi [13], derivanti da queste ultime, quali:

- **Soak Testing (o Endurance Testing)**: è una tipologia di stress test utile a simulare in maniera specifica il costante aumento degli utenti nel tempo, in modo da comprendere la sostenibilità di un sistema anche in un'ottica di lungo periodo.

- **Spike Testing**: costituisce una tipologia di stress test orientata a misurare le prestazioni del sistema quando si verifica un improvviso e significativo aumento degli utenti simultaneamente attivi, con l'obiettivo di comprendere se il sistema è in grado di gestirlo e fino a che punto è in grado di tollerare una situazione di carico decisamente più gravosa rispetto alle condizioni ordinarie. A differenza del soak testing, lo spike testing è generalmente riferito a periodi di esercizio piuttosto brevi, proprio perché si ipotizza che una condizione limite si verifichi soltanto in coincidenza di un picco di attività, molto intenso e concentrato nella durata.

- **Volume Testing**: è orientato alla valutazione delle performance di un'applicazione quando è chiamata ad elaborare una grande quantità di dati e pertanto può essere considerato un caso particolare del load testing, che è predisposto per valutare le performance del sistema in condizioni di carico normali.

- **Capacity Testing**: differisce dal tradizionale stress testing in quanto la sua azione è concentrata a valutare se un'applicazione è in grado di gestire la quantità di traffico per cui è stata originariamente progettata. Il capacity testing mira a conoscere la cosiddetta security zone, entro cui è possibile garantire le massime prestazioni del sistema senza penalizzare in alcun modo la user experience dell'utente finale.

- **Scalability Testing**: coincide in molti casi con il capacity testing ma è focalizzato in maniera specifica sulla capacità dell'applicazione di scalare le performance in funzione dei carichi di lavoro che si verificano, come nel caso di una variazione del numero degli utenti connessi al simultaneamente al sistema.

4.2 Metodologia di valutazione delle Web Dashboard

Nell'ambito del performance testing, seguire una metodologia precisa è essenziale per garantire che i test siano condotti in modo accurato e affidabile. Questo approccio consente di valutare in maniera sistematica le prestazioni delle Web Dashboard, dalla fase di preparazione alla fase di analisi dei risultati, assicurando che ogni aspetto critico venga esaminato.

La metodologia adottata per scrivere ed eseguire i test di performance, che viene mostrata anche nella figura 4.1, si articola nei seguenti passaggi fondamentali:

1. **Identificare l'ambiente e il tool su cui scrivere i test**: Scegliere con attenzione la piattaforma e gli strumenti adeguati per scrivere e gestire i test.
2. **Identificare i criteri e le metriche da testare**: Stabilire quali parametri di prestazione devono essere valutati e definire i riferimenti per ciascuna metrica, così da comprendere se un test è superato o meno.
3. **Pianificare i test**: Decidere quali test scrivere, cosa andranno a misurare e quale sarà il loro scopo specifico.
4. **Configurare l'ambiente di testing**: Preparare l'infrastruttura necessaria per l'esecuzione dei test, assicurandosi che sia adatta ai carichi di lavoro previsti.
5. **Scrivere i test**: Implementare i test seguendo la pianificazione e le metriche stabilite.
6. **Eseguire i test**: Condurre i test sull'ambiente configurato, raccogliendo i dati relativi alle prestazioni.
7. **Analizzare i risultati**: Interpretare i dati raccolti per trarre conclusioni sulle performance del sistema e identificare eventuali aree di miglioramento.

Nei sottoparagrafi seguenti, analizzeremo ciascuno di questi passaggi in dettaglio, spiegando come sono stati applicati nel nostro caso specifico.

4.2.1 Identificazione del tool

L'uso di tool di Performance Testing è una parte fondamentale del processo di sviluppo, in quanto consente di individuare i punti critici del sistema e di ottimizzare le sue prestazioni, garantendo che possa operare in modo affidabile ed efficiente in ambienti ad alta intensità di dati e in tempo reale.

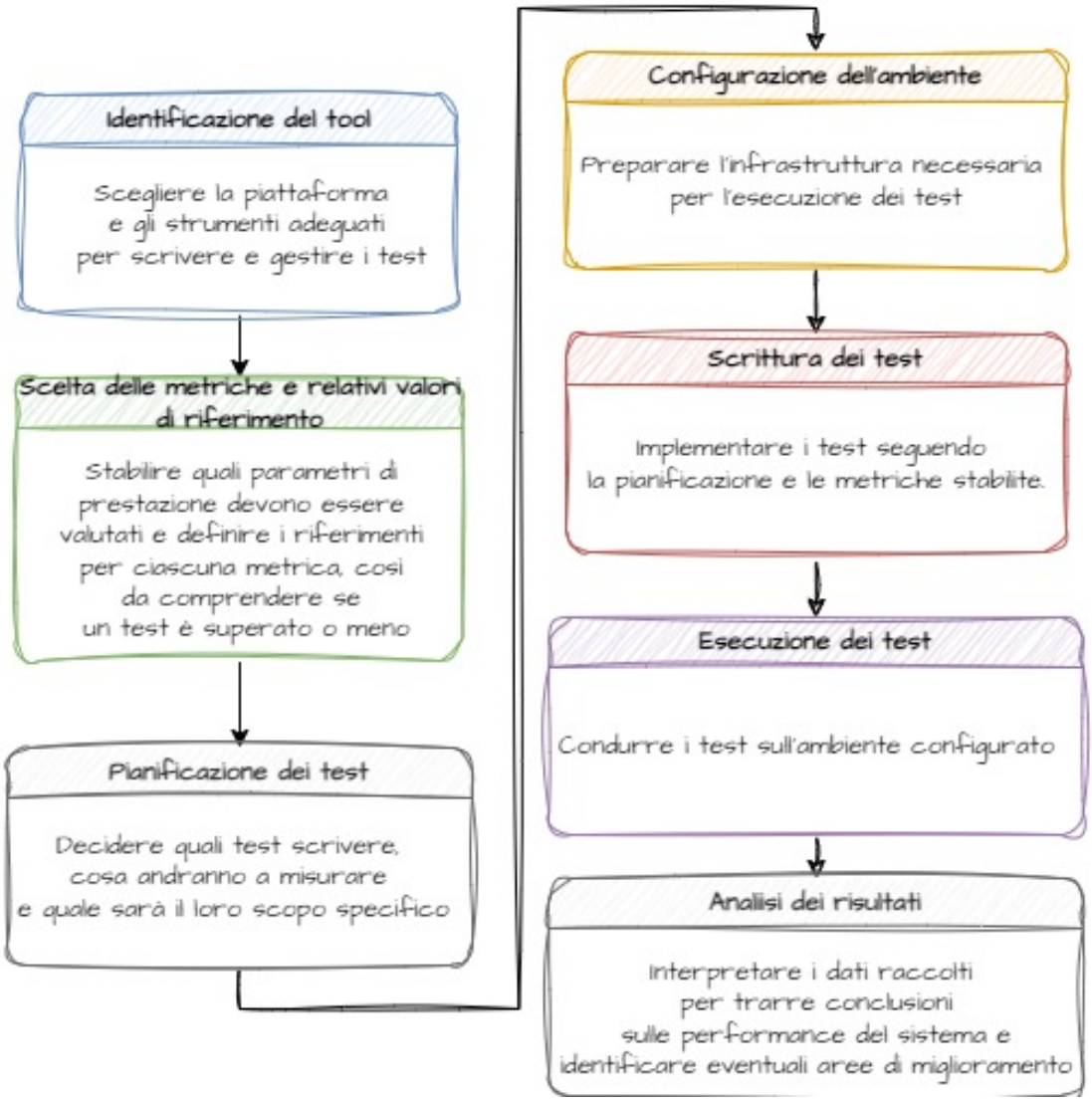


Figura 4.1: Metodologia adottata per Performance Testing

I tool di performance testing sono progettati per simulare scenari di utilizzo reali e permettono di eseguire test approfonditi su più livelli del sistema. Tra gli aspetti chiave che questi tool aiutano a monitorare vi sono:

- **Latenza nelle comunicazioni**: Misurare i tempi di risposta delle chiamate API, inclusi i WebSocket, che sono essenziali per mantenere la comunicazione in tempo reale tra il client e il server. Un'elevata latenza può compromettere l'interattività della dashboard e ridurre l'efficacia delle operazioni di analisi dei dati in tempo reale.
- **Efficienza del rendering delle interfacce**: L'interfaccia utente deve essere in grado di aggiornarsi rapidamente e in modo efficiente quando i dati cambiano, specialmente in contesti ad alta intensità di dati. Un'interfaccia lenta o poco reattiva può deteriorare drasticamente l'esperienza dell'utente, vanificando l'efficacia del sistema.
- **Gestione del carico su lunghe sessioni di utilizzo**: È importante garantire che il sistema possa sostenere prestazioni elevate anche su periodi di utilizzo prolungato, prevenendo eventuali memory leak o cali di prestazioni dovuti a un uso inefficiente delle risorse del sistema.

Alcuni tool di performance testing offrono funzionalità avanzate che vanno oltre la semplice simulazione di scenari di utilizzo di base. Ad esempio, è possibile testare il carico emulando migliaia di utenti simultanei che interagiscono con il sistema, o anche simulare eventi imprevisti, come picchi improvvisi di traffico. Questo permette di capire come il sistema reagisce sotto pressione e di verificare se la gestione delle risorse hardware, come CPU e memoria, è ottimizzata. In ambienti reali, specialmente quando si lavora con dati in streaming o dashboard interattive, il sistema può essere soggetto a fluttuazioni imprevedibili nel traffico e deve essere pronto a gestirle senza degrado significativo delle prestazioni.

Nel corso dello sviluppo, diversi strumenti sono stati considerati per monitorare le prestazioni. Tra questi, tool specializzati per il test delle interfacce web hanno avuto un ruolo centrale nella valutazione della reattività e dell'interattività dell'applicazione.

La scelta degli strumenti da utilizzare per il Performance Testing dipende dalla natura specifica del sistema. Inizialmente, sono stati valutati vari strumenti generali di carico e stress testing, utili per una prima fase di verifica delle capacità complessive del framework. Tuttavia, per garantire che i tool analizzati e il framework proposto siano in grado di soddisfare le esigenze applicative in scenari reali, sarà necessario adottare strumenti specializzati per testare la piattaforma in condizioni di utilizzo più realistiche.

A tal proposito la scelta è ricaduta su Playwright [14], uno strumento open-source creato da Microsoft per soddisfare il bisogno di scrivere test end-to-end su

applicazioni web. Implementato in TypeScript, Playwright [15] supporta anche altri linguaggi di programmazione come JavaScript, Python e Java, rendendolo una scelta flessibile per i team di sviluppo. Un punto di forza di Playwright è il suo supporto multi-browser, che include Chromium, WebKit, Firefox, Google Chrome e Microsoft Edge. Questa caratteristica permette ai test di essere eseguiti su diversi browser e sistemi operativi, tra cui Windows, Linux e macOS, sia localmente che in ambiente CI, con l'emulazione di Google Chrome Mobile e Mobile Safari per il testing su dispositivi mobili.

Playwright offre una API consolidata che semplifica la scrittura di test automatizzati, garantendo che il codice funzioni senza problemi su più browser. Questa API permette agli sviluppatori di navigare, interagire con gli input e validare le asserzioni. Un'altra caratteristica di rilievo è la sua potente capacità di intercettazione delle richieste di rete, che consente di simulare errori di rete, aggiungere header personalizzati o manipolare le richieste, offrendo una versatilità unica per il testing di scenari complessi.

Un'altra funzionalità avanzata di Playwright è il supporto per i test in modalità headless, che permette di eseguire i test senza aprire una finestra del browser. Questo è particolarmente utile negli ambienti di Continuous Integration (CI), come Jenkins o Travis CI, dove l'interfaccia grafica non è disponibile. Playwright si integra facilmente con questi strumenti di CI, permettendo test continui senza interruzioni.

Playwright consente inoltre l'esecuzione isolata dei test tramite "browser contexts", ambienti di test indipendenti che riducono l'interferenza tra test e semplificano il debug. A questo si aggiungono funzionalità come la registrazione video e la cattura di screenshot, nonché la memorizzazione di log della console e del traffico di rete, che semplificano la revisione dei test e l'identificazione dei problemi.

Rispetto ad altri strumenti come Selenium e Cypress, Playwright si distingue per la sua capacità di gestire test complessi su più browser con un'unica API, oltre a offrire un'integrazione più completa con gli strumenti di rete e CI. Inoltre, il supporto nativo per il mobile emulation e la sua capacità di intercettare e modificare richieste di rete lo rendono una scelta più potente per molte esigenze di testing avanzato.

Infine, Playwright si presta anche per scrivere test volti a valutare le performance delle applicazioni web o delle Web Dashboard, sfruttando le sue API per misurare metriche chiave e rilevare eventuali colli di bottiglia, come la latenza, la scalabilità e la capacità di carico.

4.2.2 Scelta delle metriche

Per comprendere appieno l'efficacia di una Web Dashboard, è fondamentale utilizzare metriche precise e standardizzate. Questi indicatori forniscono una base

oggettiva per valutare le prestazioni della dashboard in termini di usabilità, velocità e capacità di gestione dei dati, consentendo di identificare le aree di miglioramento e ottimizzare l'esperienza utente.

Le metriche per valutare le Web Dashboard quindi sono fondamentali per garantire che queste risorse siano non solo visivamente accattivanti, ma anche efficienti e utili per l'utente finale infatti l'efficacia di qualsiasi sistema di analisi dei dati temporali dipende non solo dalla sua funzionalità e dalle sue capacità, ma anche dalle sue prestazioni in termini di velocità, reattività e scalabilità.

Per valutare una Web Dashboard, è essenziale utilizzare un insieme di metriche ben definite che possano fornire un quadro completo delle sue prestazioni. La selezione accurata di queste metriche è fondamentale per garantire un'analisi approfondita e affidabile della dashboard e per identificare eventuali aree di miglioramento.

Per capire quali metriche scegliere per testare le Web Dashboard siamo andati ad analizzare quali potevano essere gli aspetti più importanti per un utente che deve usare una Web Dashboard e quali sono le caratteristiche di una dashboard concentrandoci maggiormente su quali caratteristiche dovesse avere una Web Dashboard per essere maggiormente usata da un utente. Poiché è essenziale che una Web Dashboard che deve elaborare una quantità abbastanza cospicua di dati in tempo reale sia altamente veloce e scalabile, abbiamo determinato che le metriche utili per valutarne la velocità e la scalabilità sono le seguenti facendo anche riferimento alle metriche definite dagli sviluppatori di Chrome [16]:

- **tempo del caricamento iniziale della dashboard (Page Load Time)**: misura il tempo necessario affinché la dashboard venga completamente renderizzata nel browser dell'utente ed è utile in quanto se la Web Dashboard risulta molto lenta nel caricare i dati l'utente potrebbe ritenere la dashboard troppo lenta e quindi abbandonare l'uso di essa. Questa metrica potrebbe anche essere estesa anche ai singoli grafici presenti nella dashboard per capire il tempo di caricamento di ogni singolo grafico visto che questi dati poi influenzano il caricamento dell'intera pagina.

- **tempo di arrivo del primo byte della risposta (First Contentful Paint)** [**17**]: misura il tempo che intercorre tra il momento in cui l'utente ha raggiunto per la prima volta la pagina e il momento in cui una parte dei contenuti della pagina viene visualizzata sullo schermo. Per questa metrica, "contenuti" si riferisce a testo, immagini (incluse le immagini di sfondo), elementi <svg> o elementi <canvas> non bianchi. Questa metrica incide linearmente sulla precedente in quanto più tempo ci mette ad arrivare il primo byte della risposta e più alto sarà il tempo di caricamento della pagina.

- **tempo di caricamento dell'elemento più grande presente nella dashboard (Largest Contentful Paint)** [**18**]: indica il tempo di rendering

del blocco di immagini o di testo più grande visibile nell'area visibile, in relazione al momento in cui l'utente ha raggiunto per la prima volta la pagina. Anche questa metrica incide sul caricamento della pagina in quanto più tempo serve per caricare l'elemento più grande e più tempo serve per caricare completamente la pagina.

- **risposta della dashboard ad alcune interazioni dell'utente (Interaction to Next Paint) [19]**: misura il tempo che impiega la Web Dashboard ad aggiornare i dati dei grafici a seguito di una interazione dell'utente quale, ad esempio, un click, una selezione, l'introduzione di un filtro,... Risulta anche questa metrica molto utile per il in quanto se l'utente svolge un'azione sulla dashboard quest'ultima deve rispondere in modo veloce per soddisfare le aspettative dell'utente.

- **tempo delle operazioni sui dati e della risposta del server (Server Response Time) [20]**: questa metrica influenza le metriche precedenti. Qualora le operazioni sui dati siano svolte sul server entrambi questi parametri dipendono dal server, mentre se le operazioni vengono svolte sul client il tempo di risposta del server sarà più breve e la velocità dei calcoli dipende dal dispositivo su cui viene eseguita la dashboard.

- **long task detection e Total Blocking Time**: misura il tempo totale dopo il First Contentful Paint (FCP) in cui il thread principale è stato bloccato per un tempo sufficientemente lungo da impedire la reattività dell'input. Il thread principale è considerato "bloccato" ogni volta che c'è un'attività lunga cioè un'attività che viene eseguita sul thread principale per più di 50 millisecondi. Diciamo che il thread principale è "bloccato" perché il browser non può interrompere un'attività in corso. Pertanto, se un utente interagisce con la pagina nel corso di un'attività lunga, il browser deve attendere il completamento dell'attività prima di poter rispondere. Se l'attività è abbastanza lunga (più di 50 millisecondi), è probabile che l'utente noterà il ritardo e considererà la pagina lenta o non funzionante.

Altre metriche che possono essere prese come riferimento sono:

- **la velocità e la frequenza di aggiornamento dei dati presenti sulla dashboard**: queste due caratteristiche sono utili a capire quanto una Web Dashboard, in presenza di tanti dati sia performante. La velocità di aggiornamento in particolare indica quanti dati si riescono ad elaborare in un secondo mentre la frequenza di aggiornamento dipende dalla Web Dashboard.

- **la massima quantità di dati per garantire un tempo di aggiornamento ottimo**: questa metrica è utile a capire al di sopra di quale soglia di dati

la dashboard inizia ad essere meno performante. Fornendo questa metrica è possibile capire per l'utente se la Web Dashboard è adatta ad analizzare o meno i dati in suo possesso.

Per quanto riguarda le prime metriche è possibile prendere dei valori di riferimento con i quali capire le performance della Web Dashboard, mentre le altre metriche, quali la frequenza di aggiornamento dei dati e la massima quantità di dati per avere i valori delle altre metriche accettabili sono delle informazioni utili all'utente per capire se la Web Dashboard può fare al proprio caso oppure no.

Andiamo quindi metrica per metrica ad analizzare come queste sono fondamentali nel nostro contesto:

- **Page Load Time (PLT)**: questa metrica riveste un'importanza significativa nel nostro contesto poiché influisce direttamente sulla User Experience. Infatti, il tempo di caricamento della pagina rappresenta un fattore cruciale: un tempo di caricamento più elevato può compromettere notevolmente l'esperienza dell'utente all'interno della Web Dashboard. Un caricamento lento non solo può causare frustrazione e rallentare l'efficienza operativa dell'utente, ma può anche portare a una percezione negativa dell'interfaccia, influenzando così l'intero processo di interazione con la dashboard. Pertanto, ridurre il tempo di caricamento è essenziale per garantire un'esperienza utente fluida e soddisfacente.

- **First Contentful Paint (FCP)**: come discusso nel capitolo precedente, questa metrica influisce direttamente su quella precedente poiché rappresenta il tempo necessario affinché il primo byte di dati arrivi al browser dell'utente. In altre parole, il tempo di arrivo del primo byte è un indicatore cruciale che contribuisce al tempo complessivo di caricamento della pagina. Poiché un tempo di arrivo del primo byte più lungo si traduce in un incremento del tempo totale di caricamento, questa metrica ha un impatto simile alla metrica precedente, sottolineando l'importanza di una risposta rapida del server per migliorare l'efficienza e la velocità di caricamento della Web Dashboard.

- **Largest Contentful Paint (LCP)**: analogamente alla metrica precedente, anche questa influisce sul tempo totale di caricamento della pagina, poiché misura il tempo necessario per caricare l'elemento più grande visibile nella dashboard. Il tempo di caricamento di questo elemento chiave contribuisce significativamente al tempo complessivo di rendering della pagina; quindi, un aumento del tempo di caricamento dell'elemento più grande si traduce in un aumento del tempo complessivo necessario per visualizzare completamente la dashboard. Questo riflette l'importanza di ottimizzare il caricamento degli elementi principali per migliorare la velocità e l'efficienza complessiva della Web Dashboard.

- **Interaction to Next Paint (INP)**: questa metrica condivide una motivazione simile a quella del tempo di caricamento della pagina, poiché riflette il tempo necessario per aggiornare la dashboard in risposta alle interazioni dell'utente, come filtri, clic e selezioni. Se la risposta della dashboard a tali interazioni è lenta, può compromettere notevolmente l'esperienza dell'utente, provocando frustrazione e potenzialmente portando all'abbandono della dashboard. In altre parole, una reattività insufficiente può influire negativamente sull'usabilità e sull'accettabilità della dashboard, rendendo essenziale ottimizzare questo aspetto per mantenere un'esperienza utente positiva e coinvolgente.

- **Server Response Time (SRT)**: anche il tempo di risposta del server ha un impatto significativo sulle metriche precedenti. Infatti, quanto più il server impiega a rispondere alle richieste del client, tanto più lungo sarà il tempo necessario alla dashboard per caricare i dati iniziali e per rispondere alle interazioni dell'utente. In altre parole, un aumento del tempo di risposta del server si riflette direttamente nei tempi di caricamento della pagina e nella reattività della dashboard, influenzando negativamente l'esperienza complessiva dell'utente.

- **Long Task Detection e Total Blocking Time**: questa metrica misura la presenza di attività che bloccano l'esecuzione del thread principale, il che può influire negativamente sulle metriche precedenti. Il Total Blocking Time (TBT) fornisce una misura della durata complessiva in cui il thread principale è stato bloccato, impedendo l'interazione fluida con la dashboard. Un thread principale bloccato comporta ritardi nelle operazioni di rendering e nelle risposte alle interazioni dell'utente, amplificando i tempi di caricamento della pagina e riducendo la reattività complessiva della dashboard.

Una volta scelte le metriche, è fondamentale stabilire i valori di riferimento o benchmark per ciascuna di esse. Questi valori aiutano a determinare se le prestazioni ottenute dai test sono eccellenti o se necessitano di miglioramenti. Per definire tali valori di riferimento, ci siamo basati sulle linee guida fornite dagli sviluppatori di Google. Essi non solo hanno identificato le metriche cruciali per valutare l'efficienza di una Web Dashboard, ma hanno anche stabilito dei valori di riferimento per ciascuna metrica.

Questi valori di riferimento sono suddivisi in tre fasce:

- **Valore Ideale**: questa fascia è delimitata da un valore ottimale, al di sotto del quale le prestazioni della dashboard sono considerate eccellenti. In questa fascia, l'esperienza utente è ideale e la dashboard funziona in modo molto efficiente.

- **Possibili Miglioramenti**: questa fascia è delimitata tra il valore ideale e quello non accettabile. Indica prestazioni soddisfacenti ma che potrebbero essere migliorate. In questa zona, la dashboard è utilizzabile, ma non offre un'esperienza ottimale.
- **Valore Non Accettabile**: rappresenta il valore al di sopra del quale le prestazioni sono considerate inadeguate. In questa fascia, le prestazioni della dashboard sono insufficienti e potrebbero compromettere gravemente l'esperienza utente, suggerendo la necessità di interventi correttivi urgenti.

La tabella 4.1 riporta i valori di riferimento specifici per ciascuna metrica, suddivisi nelle tre fasce sopra descritte. Questi benchmark ci permettono di analizzare e valutare le prestazioni della Web Dashboard in modo preciso e oggettivo, facilitando l'individuazione delle aree di miglioramento e garantendo una migliore esperienza utente.

Metrica	Ideale	Possibili Miglioramenti	Non Accetabile
PLT	<= 2.5 s	tra 2.5 e 4 s	> 4 s
FCP	<= 1.8 s	tra 1.8 e 3 s	> 3 s
LCP	<= 2.5 s	tra 2.5 e 4 s	> 4 s
INP	<= 200 ms	tra 200 e 500 ms	> 500 ms
SRT	<= 800 ms	tra 800 ms e 1.8 s	> 1.8 s

Tabella 4.1: Benchmark metriche

4.2.3 Pianificazione dei test

Una volta selezionate le metriche da testare e stabiliti i valori di riferimento, il passo successivo è pianificare i test da eseguire. Abbiamo optato per la creazione di un test unico per valutare tutte le metriche selezionate. Tuttavia, per ciascuna soluzione utilizzata, sono stati progettati due test distinti: uno per ciascuna dashboard che utilizza un tipo di database diverso. Questa scelta ci ha permesso di analizzare come le diverse soluzioni gestiscono le operazioni in relazione ai database utilizzati. I database scelti per questa valutazione sono stati SQLite, un database organizzato per righe, e ClickHouse, un database organizzato invece per colonne.

Per la progettazione dei test, abbiamo fatto riferimento a documenti forniti dal Web Performance Working Group, un gruppo che opera sotto l'egida del World Wide Web Consortium (W3C). In particolare, abbiamo esaminato i documenti riguardanti le metriche che abbiamo selezionato, tra cui Navigation Timing, User Timing, High Resolution Timing, Paint Timing, Server Timing e Performance Timeline. Questi documenti ci hanno fornito le linee guida necessarie per garantire

che i test fossero accurati e conformi agli standard di valutazione delle prestazioni web.

In questo sottoparagrafo verrà illustrato in dettaglio come ciascun documento del Web Performance Working Group ha influito sulla scrittura dei test specifici per ogni metrica. Analizzeremo come le definizioni e le raccomandazioni contenute in questi documenti abbiano guidato la nostra progettazione dei test e come abbiamo applicato queste indicazioni nella pratica per garantire una valutazione precisa delle performance delle Web Dashboard.

Il primo documento che siamo andati ad analizzare è stato Performance Timeline [21] il quale specifica le primitive della Timeline delle Prestazioni che consentono agli sviluppatori web di misurare accuratamente le caratteristiche delle prestazioni delle applicazioni web ed essenzialmente, fornisce un modo per accedere, strumentare e recuperare varie metriche di prestazione dall'intero ciclo di vita di un'applicazione web. Il documento afferma inoltre che gli sviluppatori possono accedere alla Timeline delle prestazioni attraverso l'interfaccia "Performance". L'uso di PerformanceObserver è incoraggiato perché risolve alcune limitazioni dell'approccio basato su buffer. Consente di evitare il polling della timeline per rilevare nuove metriche, elimina la logica costosa di deduplicazione per identificare nuove metriche e risolve le condizioni di concorrenza con altri consumatori che potrebbero voler manipolare il buffer. Inoltre, nuove API e metriche di prestazione potrebbero essere disponibili solo tramite l'interfaccia PerformanceObserver. Il documento definisce anche metodi e attributi aggiuntivi per l'oggetto Performance, come getEntries(), getEntriesByType() e getEntriesByName().

Ora andiamo ad analizzare per ogni metrica come siamo arrivati alla scrittura del test analizzando anche come i documenti del Web Performance Working Group del W3C ci sono stati utili nella scrittura dei vari test.

Page Load Time

Per calcolare il tempo del caricamento iniziale dell'intera Web Dashboard e dei suoi componenti siamo andati ad analizzare il documento Navigation Timing [22] il quale ci ha indirizzato per scrivere il test per analizzare questa caratteristica importante per una Web Dashboard.

Il documento Navigation Timing è una specifica tecnica che definisce un'interfaccia per consentire alle applicazioni web di accedere a informazioni dettagliate sulle tempistiche di navigazione di un documento e può essere adattato al nostro scopo di analisi su una Web Dashboard. L'interfaccia, chiamata PerformanceNavigationTiming, è progettata per fornire metriche ad alta risoluzione sulle prestazioni relative al caricamento di una pagina web e aggiornare una vecchia versione costituita dalle due interfacce chiamate PerformanceTiming e PerformanceNavigation. Questo include informazioni come il tempo di caricamento effettivo della pagina,

il tempo di inizio e fine di eventi come il caricamento del DOM e l'esecuzione di script, e il numero di reindirizzamenti che si sono verificati durante la navigazione. Inoltre, sono inclusi dettagli su considerazioni importanti relative alla privacy e alla sicurezza.

Sulla base di quanto ricavato dallo studio di questo documento, abbiamo usato le due interfacce per scrivere la parte dei test relativi al caricamento della pagina complessiva.

First Contentful Paint

Per calcolare il tempo relativo al First Contentful Paint invece abbiamo fatto riferimento al documento chiamato Paint Timing [23] il quale invece specifica come misurare i tempi di rendering di una pagina web, che nel nostro caso è una Web Dashboard, concentrandosi sui momenti chiave come il primo paint (FP) e, appunto, il primo paint di contenuti significativi (FCP) introducendo una interfaccia chiamata PerformancePaintTiming che estende l'interfaccia PerformanceEntry e consente di ottenere informazioni sul paint timing includendo il tipo di paint (First Paint o First Contentful Paint), il timestamp di inizio e la durata.

Largest Contentful Paint

Per calcolare il tempo relativo al Largest Contentful Paint abbiamo fatto riferimento proprio al documento Largest Contentful Paint [24] che presenta l'API Largest Contentful Paint la quale si basa sui concetti definiti nella specifica Paint Timing e offre una misurazione robusta e automatica per identificare il contenuto visivamente più rilevante per l'utente durante il caricamento della pagina. L'algoritmo dell'API tiene traccia del contenuto più grande renderizzato fino a quel momento e anche se un contenuto viene rimosso, rimane comunque rilevante per l'algoritmo il quale si arresta quando avvengono eventi di scroll o input da parte dell'utente, poiché questi potrebbero introdurre nuovi contenuti.

L'API è basata su euristiche e presenta alcune problematiche:

- Se l'utente interagisce con la pagina prima del completamento del caricamento, l'algoritmo potrebbe restituire risultati errati o incompleti.
- I contenuti rimossi vengono comunque considerati, il che può creare problemi con elementi come splash screen.

Interaction to Next Paint

Per calcolare il tempo che ci mette la Web Dashboard a rispondere con i dati aggiornati ad una interazione dell'utente quale un click, un cross-filter, etc abbiamo

preso come riferimento il documento User Timing [25] che ci ha indirizzato alla scrittura del relativo test.

L'articolo introduce l'API User Timing, sviluppata per consentire agli sviluppatori web di valutare le prestazioni delle proprie applicazioni e nel nostro caso di una Web Dashboard. Il documento fa inizialmente riferimento all'imprecisione del metodo Date.now(), analizzato ulteriormente dal documento High Resolution Time, di JavaScript per ricavare il timestamp in base ai diversi agenti utente per poi definire le interfacce PerformanceMark e PerformanceMeasure per esporre un timestamp ad alta precisione e monotonicamente crescente.

Come accennato prima abbiamo inoltre analizzato perché il metodo Date.now() non è molto preciso attraverso il documento High Resolution Time [26] il quale afferma che l'oggetto Date è molto limitante nella misurazione accurata del tempo a causa dello scarto dell'orologio di sistema e della mancanza di risoluzione sub-millisecondo. Le specifiche definiscono capacità come orologi stabili e monotonici con risoluzione sub-millisecondo, essenziali per compiti come la misurazione delle prestazioni, la determinazione del frame rate delle animazioni e la sincronizzazione tra contesti diversi come i Workers e il thread principale.

Server Response Time

Per calcolare il tempo che ci mette il server ad eseguire le operazioni richieste sui dati per poi mostrarli nella Web Dashboard e mandare la risposta al client a seguito di una richiesta abbiamo analizzato il documento Server Timing [27] che ci ha indirizzato nella scrittura del relativo test. Questo documento introduce l'interfaccia PerformanceServerTiming, che consente ai server di comunicare al browser metriche di performance relative al ciclo richiesta-risposta. Questa interfaccia rappresenta una singola metrica di performance comunicata dal server al browser definendo gli attributi npme, durata e descrizione, insieme a un metodo toJSON per la serializzazione dell'oggetto. Con questa nuova interfaccia inoltre si estende l'interfaccia PerformanceResourceTiming, che rappresenta i tempi di risorsa associati a una richiesta di risorsa. In questa estensione, si aggiunge un nuovo attributo serverTiming per ottenere le metriche di performance comunicate dal server. Attraverso l'interfaccia definita nel documento abbiamo scritto la parte del test per calcolare il tempo utilizzato per effettuare le operazioni e il tempo complessivo della risposta del server.

4.2.4 Ambiente di esecuzione dei test

Per eseguire i test, abbiamo deciso di creare una dashboard simile su tutte le soluzioni analizzate, utilizzando un dataset pertinente e uguale per ciascuna di esse. Nelle immagini 4.2, 4.3 e 4.4 vengono riportate le immagini delle dashboard

sviluppate con Grafana, Metabase, e il framework proposto. Queste dashboard sono state progettate per garantire la coerenza tra le soluzioni e consentire un confronto accurato delle prestazioni. Per quanto riguarda il dataset da analizzare abbiamo optato per un dataset legato a dati energetici in quanto in questo ambito bisogna analizzare tanti dati alla volta che rispecchia pienamente l'obiettivo di questa tesi. In particolare il dataset scelto è Daily Generation by Energy Source [28] che rappresenta un monitoraggio giornaliero nella produzione netta per ente di bilanciamento e fonte energetica negli Stati Uniti ed è fornito dall'EIA (Energy Information Administration) attraverso un'API pubblica a cui poter fare accesso. Un esempio dei dati analizzati è presente nell'immagine 4.5.

Per i test è stata selezionata una porzione del dataset contenente 200.000 righe, considerata un buon punto di partenza. Tuttavia, in ambienti di lavoro reali, è probabile che si debba gestire una quantità di dati significativamente maggiore nel tempo.

Sulla base di questi dati abbiamo scelto di rappresentare i seguenti grafici nelle Web Dashboard per i vari tool:

- **Pie Chart**: grafico a torta che rappresenta la distribuzione percentuale di ciascun tipo di carburante (fueltype) rispetto al totale dei consumi.

- **Bar Chart**: grafico a barre che rappresenta il minimo, la media ed il massimo di un determinato tipo di carburante (fueltype) rispetto al periodo.

- **Time Series**: grafico a linea che rappresenta la variazione della media di un valore di un certo tipo di carburante (fueltype) nel tempo (period).

In ogni dashboard è stato poi aggiunto un filtro per selezionare quale tipo di carburante si vuole vedere nel grafico a linea.

Abbiamo deciso anche di selezionare due tipi diversi di database per ogni tool, SQLite e ClickHouse. Il primo, SQLite, è un tipo di database SQL che salva i dati su tabelle organizzate in righe e in cui ogni riga rappresenta un'istanza della tabella, mentre il secondo ClickHouse, salva i dati in tabelle organizzate in colonne che risulterà essere un modo più veloce per svolgere calcoli su una quantità consistente di dati.

Per garantire la replicabilità dei test su diversi ambienti, abbiamo utilizzato Docker per creare immagini ad-hoc delle dashboard sviluppate con Grafana e Metabase, collegate alle rispettive fonti dati. Queste immagini sono state caricate su Docker Hub e sono disponibili per essere scaricate da chiunque. In aggiunta, è stato creato un repository GitHub con un file README in cui si spiega come replicare i test effettuati, oltre ai test scritti con PlayWright. La dashboard creata con il framework proposto, invece, viene eseguita localmente.

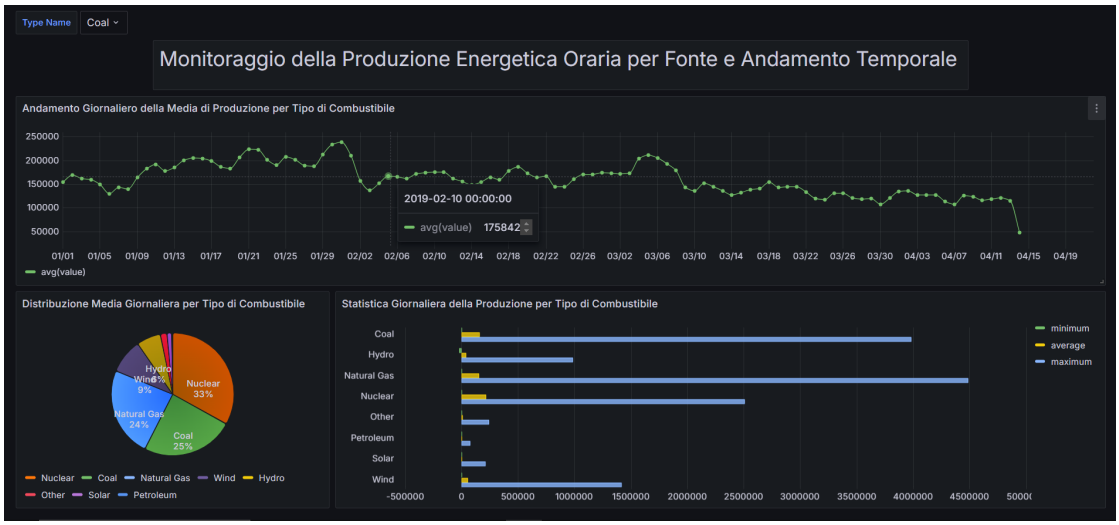


Figura 4.2: Dashboard creata con Grafana

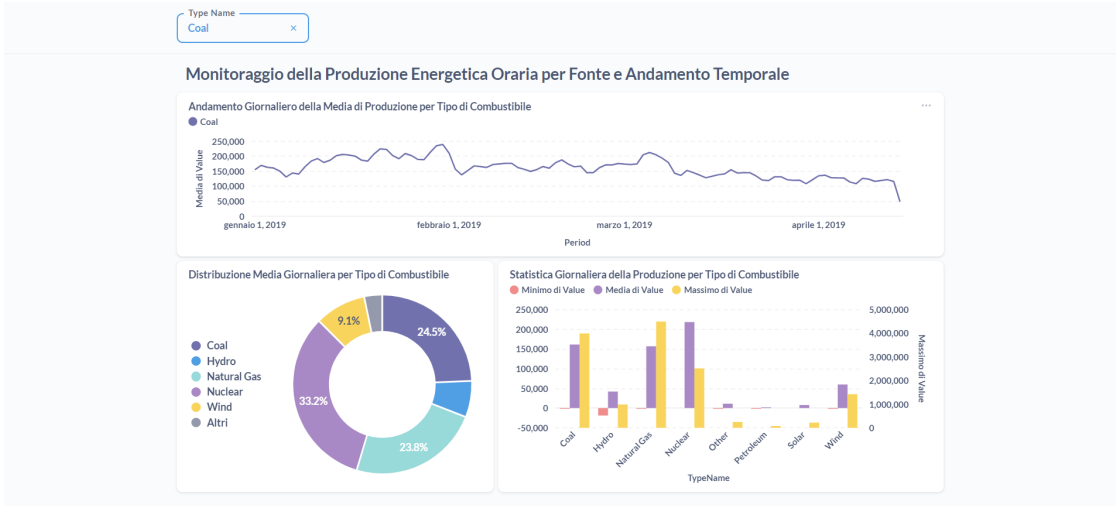


Figura 4.3: Dashboard creata con Metabase

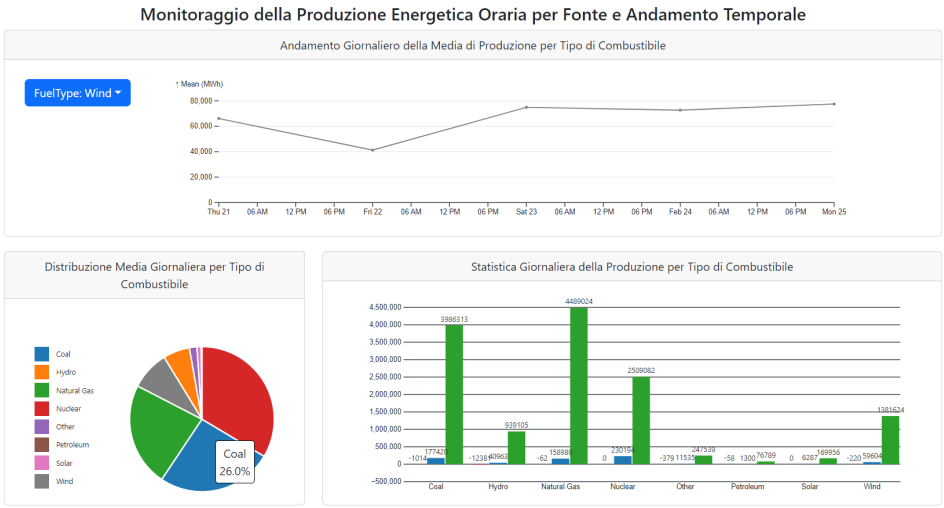
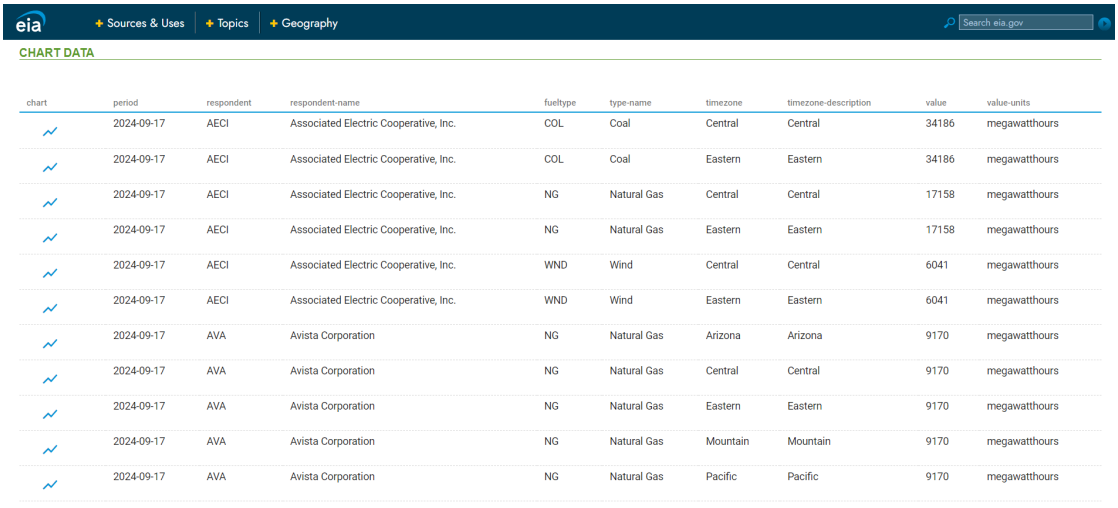


Figura 4.4: Dashboard creata con il framework prototipale



eia + Sources & Uses + Topics + Geography — Search eia.gov

CHART DATA

chart	period	respondent	respondent-name	fueltype	type-name	timezone	timezone-description	value	value-units
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Central	Central	34186	megawatthours
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	COL	Coal	Eastern	Eastern	34186	megawatthours
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	NG	Natural Gas	Central	Central	17158	megawatthours
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	NG	Natural Gas	Eastern	Eastern	17158	megawatthours
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	WND	Wind	Central	Central	6041	megawatthours
	2024-09-17	AECI	Associated Electric Cooperative, Inc.	WND	Wind	Eastern	Eastern	6041	megawatthours
	2024-09-17	AVA	Avista Corporation	NG	Natural Gas	Arizona	Arizona	9170	megawatthours
	2024-09-17	AVA	Avista Corporation	NG	Natural Gas	Central	Central	9170	megawatthours
	2024-09-17	AVA	Avista Corporation	NG	Natural Gas	Eastern	Eastern	9170	megawatthours
	2024-09-17	AVA	Avista Corporation	NG	Natural Gas	Mountain	Mountain	9170	megawatthours
	2024-09-17	AVA	Avista Corporation	NG	Natural Gas	Pacific	Pacific	9170	megawatthours

Figura 4.5: Esempio dei dati analizzati nelle Web Dashboard

4.2.5 Scrittura dei test

Una volta analizzati i documenti descritti in precedenza siamo passati al mettere in atto le nozioni acquisite per la scrittura dei vari test usando PlayWright. Come affermato in precedenza sono stati scritti due test per ognuna delle soluzioni che verranno analizzate (Grafana, Metabase e il framework prototipale), uno per la dashboard che utilizza ClickHouse come database e uno che utilizza SQLite. Per quanto riguarda le dashboard che verranno create con Grafana e Metabase test automatizzati seguono un flusso ben definito, visibile anche nella figura 4.6, che comprende i seguenti passaggi:

1. Il test inizia navigando verso l'interfaccia del tool (Grafana o Metabase), successivamente viene simulato l'accesso con credenziali predefinite tramite l'interfaccia utente. Una volta autenticato, il test esplora il menu principale per trovare e aprire la dashboard specifica collegata al database di interesse (ClickHouse o SQLite). Questo passaggio simula esattamente il comportamento di un utente che si autentica e accede a una dashboard.

2. Dopo aver aperto la dashboard, il test applica una serie di configurazioni iniziali:

   - Selezione di un intervallo temporale predefinito per visualizzare i dati (ad esempio, dal 2019 fino alla data corrente).
   - Configurazione di variabili di filtro dinamico, come il tipo di risorsa da visualizzare (ad esempio, il tipo di energia, come il carbone o l'idroelettrico), in modo da replicare scenari reali di utilizzo da parte dell'utente.

3. A questo punto, utilizzando quando dedotto dai documenti analizzati in precedenza, sono state scritte le parti del test per ogni metrica che è stata scelta da analizzare.

4. Per facilitare la lettura dei risultati ogni metrica raccolta durante il test viene confrontata con le serie di soglie prestabilite che classificano le prestazioni nelle tre categorie descritte in precedenza.

5. Infine tutti i risultati vengono stampati su un file per poi poter essere letti e analizzati.

Mentre per quanto riguarda la dashboard creata con il framework prototipale il flusso è leggermente diverso in quanto non è prevista autenticazione, in questo caso infatti il flusso seguito dai test, visibile anche nella figura 4.7, è il seguente:

1. Il test inizia navigando verso l'interfaccia della Web Dashboard sviluppata. In questa fase, non è richiesta autenticazione e la distinzione tra l'utilizzo di

SQLite e ClickHouse viene gestita attraverso modifiche nel codice del server, in base al database configurato.

2. Successivamente, sono state definite e implementate le parti del test relative alle metriche di performance. Queste metriche sono state scelte basandosi sui criteri individuati nei documenti di analisi discussi precedentemente, concentrandosi su aspetti come i tempi di risposta, l'efficienza del flusso di dati e l'interattività dell'interfaccia.

3. I risultati raccolti durante i test sono stati poi confrontati con soglie prestabilite, classificate in tre categorie, come fatto nei capitoli precedenti. Questo processo ha facilitato l'interpretazione delle prestazioni, fornendo una valutazione chiara del comportamento della dashboard in termini di efficienza e reattività.

4. Infine, tutti i dati e i risultati dei test sono stati raccolti e stampati in un file di log per essere analizzati in modo più dettagliato. Questo ha permesso una visione complessiva delle prestazioni della Web Dashboard rispetto agli obiettivi e alle aspettative iniziali.

In conclusione, questo capitolo ha descritto la metodologia adottata per condurre i test sulle diverse soluzioni di Web Dashboard, inclusa la scelta degli strumenti, la configurazione degli ambienti di test e le metriche chiave utilizzate per la valutazione delle prestazioni. Nel prossimo capitolo verranno eseguiti i test sulle dashboard sviluppate con Grafana, Metabase e il framework proposto, e sarà presentato un confronto dettagliato dei risultati ottenuti.

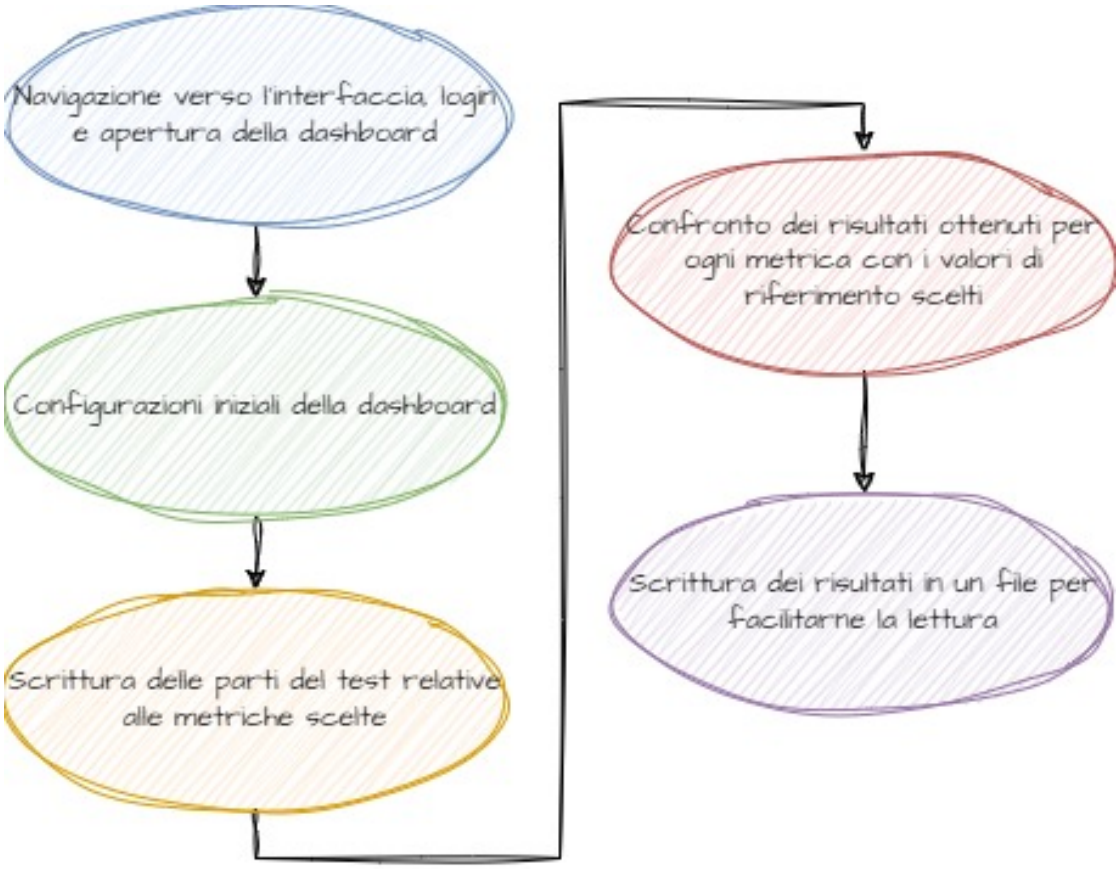


Figura 4.6: Flusso di esecuzione dei test eseguiti su Grafana e Metabase

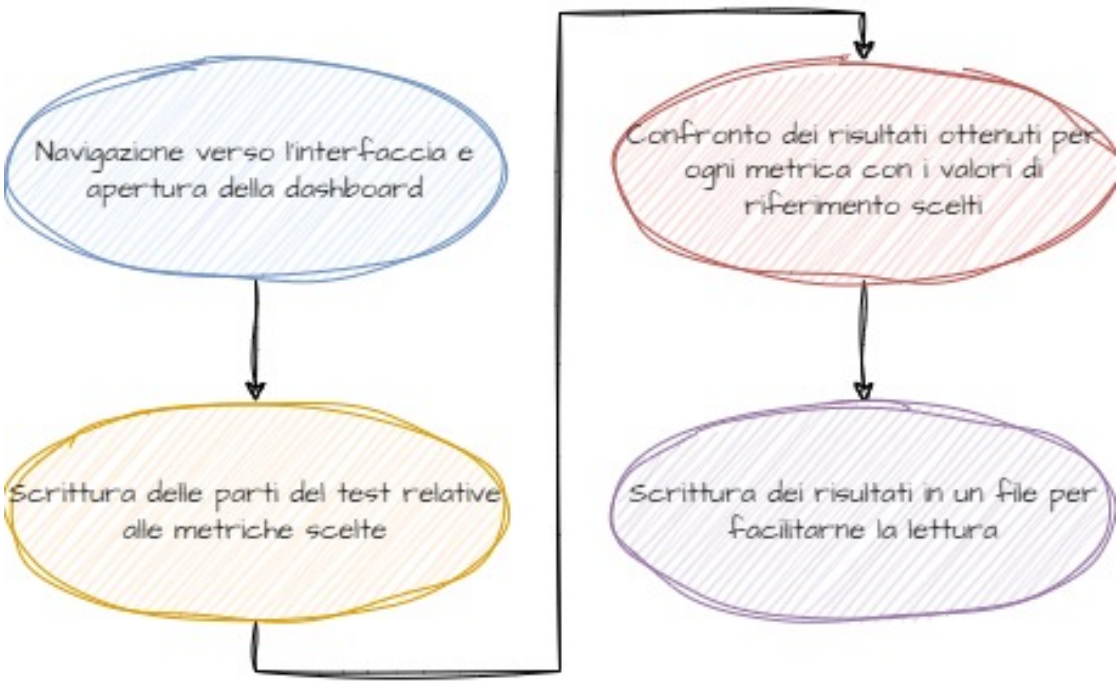


Figura 4.7: Flusso di esecuzione dei test eseguiti sul framework prototipale

Capitolo 5

Risultati

In questo capitolo verrà descritto il processo di esecuzione dei test di performance sulle tre soluzioni analizzate: Grafana, Metabase e la Web Dashboard sviluppata con il framework prototipale.

Nella prima parte del capitolo, saranno presentati i risultati dei test condotti su ciascuna soluzione in modo individuale. Per ogni piattaforma, si analizzeranno le prestazioni nei diversi scenari di utilizzo, mettendo in evidenza i punti di forza e le aree critiche, come la velocità di aggiornamento dei dati in tempo reale e la capacità di gestire elevati carichi di lavoro.

Successivamente, verrà effettuato un confronto diretto tra i risultati ottenuti dalle tre soluzioni, per identificare differenze significative in termini di prestazioni e scalabilità. Questa analisi comparativa consentirà di valutare l'efficacia della soluzione proposta rispetto a Grafana e Metabase, evidenziando eventuali vantaggi o svantaggi nell'utilizzo del framework per la gestione di dati temporali in tempo reale.

Infine, si analizzeranno i miglioramenti introdotti dal framework rispetto alle altre due soluzioni, sottolineando come l'uso di tecnologie come WebAssembly e Rust, insieme a un'architettura modulare e scalabile, abbia contribuito a ottimizzare le prestazioni e a migliorare l'interattività della Web Dashboard. Verranno discussi i benefici ottenuti in termini di reattività e gestione efficiente dei dati rispetto alle alternative esaminate.

5.1 Esecuzione dei test e Risultati Ottenuti

Dopo aver descritto le metriche e l'ambiente di test, si passa ora all'esecuzione dei test. Questa fase permette di raccogliere i dati necessari per valutare le prestazioni delle diverse soluzioni, analizzando i risultati ottenuti per ciascuna dashboard.

L'obiettivo è raccogliere i valori ottenuti per ciascuna metrica e confrontarli con i valori di riferimento, al fine di valutare le prestazioni di ogni soluzione e identificare i punti di forza e di debolezza.

I test sono stati eseguiti su una macchina Windows con le seguenti specifiche tecniche:

Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz (settima generazione)
Memoria RAM: 8GB

Inoltre, per valutare il comportamento delle dashboard in diversi contesti di utilizzo, i test sono stati effettuati su quattro browser: Chromium, Mozilla Firefox, Google Chrome e Microsoft Edge.

Per garantire l'accuratezza dei risultati, ogni test è stato eseguito 10 volte su ciascuna combinazione di soluzione e browser. Successivamente, è stata calcolata la media dei valori ottenuti, per ridurre l'impatto di eventuali variazioni legate a fattori esterni o imprevedibili. Questo approccio consente di ottenere misurazioni più precise e affidabili rispetto all'esecuzione di un singolo test.

Nei paragrafi successivi, verranno analizzati i risultati dei test per ciascuna soluzione, mettendo in evidenza le performance ottenute e confrontandole con i valori di riferimento per ogni metrica.

5.1.1 Grafana

Eseguendo i test descritti nel capitolo precedente sulla dashboard sviluppata con Grafana e utilizzando SQLite come database, i risultati per ciascuna metrica sono stati riportati nelle tabelle 5.1, 5.2, 5.3 e 5.4. Da un'analisi dei risultati, emerge che quasi tutte le metriche nei browser testati non rispettano i benchmark stabiliti nella tabella 4.1, evidenziando che la dashboard è lenta e inadeguata per l'obiettivo previsto.

Quando lo stesso test è stato eseguito sulla dashboard Grafana con il database ClickHouse, i risultati mostrati nelle tabelle 5.5, 5.6, 5.7 e 5.8 indicano che la dashboard risponde in modo più efficiente allo scopo per cui è stata progettata. Tuttavia, l'aggiornamento dei dati ogni 5 secondi potrebbe risultare inadeguato in alcuni contesti, riducendo l'efficacia complessiva. Inoltre, è possibile notare che la dashboard per la metrica "Interaction to Next Paint" necessita di miglioramenti per raggiungere dei valori ottimali.

Pertanto, possiamo concludere che Grafana associato a un database strutturato per righe, come SQLite, è poco efficiente nel gestire grandi volumi di dati in tempo reale, mentre, se abbinato a un database organizzato per colonne come ClickHouse, risulta più adatto a gestire questo tipo di carichi ma presenta comunque dei fattori che ne limitano l'uso per lo scopo per cui è stato pensato.

5.1.2 Metabase

I test condotti sulla dashboard Metabase con SQLite come database hanno prodotto i risultati riportati nelle tabelle 5.9, 5.10, 5.11 e 5.12. L'analisi dei dati indica che la maggior parte delle metriche si colloca nella fascia "Non accettabile", tranne alcuni valori che ricadono nella fascia "Possibili miglioramenti". Di conseguenza, Metabase con SQLite si dimostra inefficace ed inefficiente su tutti i browser testati per gestire tanti dati in tempo reale.

Eseguendo i test sulla stessa dashboard, ma utilizzando ClickHouse come database, i risultati visibili nelle tabelle 5.13, 5.14, 5.15 e 5.16 rivelano un leggero miglioramento delle metriche, ma i risultati complessivi restano comunque insoddisfacenti.

In conclusione, possiamo affermare che Metabase non è una soluzione adeguata per creare Web Dashboard che richiedono la gestione di grandi quantità di dati in tempo reale. I risultati delle metriche, insieme al fatto che i dati vengono aggiornati solo ogni minuto, indicano scarse prestazioni.

5.1.3 Framework Prototipale

Infine, eseguendo i test sulla Web Dashboard creata con il framework prototipale presentato nel capitolo 3 abbiamo ottenuto i seguenti risultati.

Per quanto riguarda l'interazione con il database organizzato a righe SQLite i risultati li possiamo trovare nelle tabelle 5.17, 5.18, 5.19, 5.20.

Mentre per quanto riguarda i risulati dei test effettuati sulla dashboard che usa come database quello organizzato a colonne, quindi ClickHouse, troviamo i risultati nelle tabelle 5.21, 5.22, 5.23 e 5.24.

Durante i test della dashboard sviluppata, è stato registrato un tempo di caricamento della pagina piuttosto elevato, così come valori notevoli per le metriche di First Contentful Paint (FCP) e Largest Contentful Paint (LCP). Questi risultati possono essere attribuiti a diversi fattori, uno dei quali è la configurazione del server, impostato, per semplicità di sviluppo, in modalità localhost sulla stessa macchina utilizzata per eseguire i test. In particolare, come in precendeza, i test sono stati effettuati su un computer Windows dotato di un processore Intel di settima generazione, specificamente un *Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz*, con 8GB di memoria RAM. Sebbene queste specifiche siano adeguate per attività di sviluppo e test, non risultano ottimali per un server in grado di gestire operazioni di elaborazione in tempo reale e l'interazione di più utenti simultaneamente.

In aggiunta, l'uso della stessa macchina come server e client implica che non si considerano i potenziali ritardi di rete che potrebbero emergere in un ambiente di produzione. Questa scelta, sebbene semplifichi il contesto di test, limita la capacità di valutare come la dashboard si comporterebbe in condizioni reali, dove la latenza della rete può influenzare in modo significativo l'esperienza dell'utente finale.

5.2 Confronto dei risultati dei test eseguiti nelle varie soluzioni

L'analisi dei risultati ottenuti dai test condotti su Grafana e Metabase, offre un'opportunità preziosa per confrontare le prestazioni e l'efficacia delle soluzioni adottate rispetto alla Web Dashboard sviluppata utilizzando il framework prototipale descritto nel capitolo 3. Questo confronto si rivela cruciale per comprendere le dinamiche operative di ciascuna soluzione e per identificare le aree di forza e di debolezza.

Una delle osservazioni più evidenti riguarda la superiorità dei database colonnari, come ClickHouse, rispetto ai tradizionali database relazionali basati su righe, come nel caso di SQLite. Questa differenza strutturale è fondamentale: i database colonnari sono progettati per ottimizzare le query analitiche e le operazioni di aggregazione, risultando così più efficienti nell'elaborazione e nel recupero di grandi volumi di dati. Questa architettura si traduce in prestazioni superiori, specialmente quando si lavora con dataset di dimensioni significative, come nel nostro caso.

Un elemento cruciale da tenere in considerazione è la metrica dell'Interaction to Next Paint, che si rivela particolarmente significativa nel contesto della nostra applicazione. Questa metrica sottolinea il vantaggio competitivo di eseguire operazioni direttamente sul client piuttosto che sul server. Con l'architettura implementata, ogni interazione dell'utente non richiede l'invio di una richiesta al server per ottenere i dati elaborati; i calcoli vengono eseguiti in tempo reale nel browser, permettendo così un'esperienza utente molto più fluida e reattiva. Questo approccio contribuisce a ridurre il tempo di attesa percepito dall'utente, migliorando la soddisfazione complessiva e l'usabilità della dashboard.

Infatti, l'Interaction to Next Paint era l'unica metrica che necessitava miglioramenti in tutte le combinazioni di database-browser analizzate su Grafana e Metabase. Anche utilizzando Grafana con ClickHouse, si notava infatti un aumento del tempo di interazione man mano che cresceva il volume di dati, poiché aumentava anche la quantità di informazioni da trasmettere tra client e server ad ogni interazione dell'utente.

Analizzando i risultati ottenuti per la metrica Interaction to Next Paint della dashboard sviluppata con il framework prototipale, è possibile osservare un miglioramento significativo. Questo porta a una risposta più tempestiva alle interazioni degli utenti, con un conseguente aumento della usabilità della dashboard e della fluidità dell'esperienza utente.

Tuttavia, è essenziale considerare le capacità hardware della macchina su cui è in esecuzione la dashboard. Una macchina più potente, dotata di un processore più veloce e una maggiore quantità di RAM, sarà in grado di gestire elaborazioni più complesse e di fornire prestazioni superiori. Questo punto sottolinea l'importanza di ottimizzare l'hardware nei contesti in cui si prevede un uso intensivo delle

applicazioni web, in particolare quelle che richiedono elaborazioni in tempo reale. In effetti, i miglioramenti sono già evidenti anche su una macchina standard. Utilizzando un server dedicato e ottimizzato, è probabile che si ottengano risultati accettabili anche per altre metriche, rendendo la soluzione proposta nettamente superiore rispetto ai tool analizzati.

In conclusione, il confronto tra i risultati dei test condotti sui tool esistenti e quelli sulla Web Dashboard sviluppata con il framework mette in luce sia i punti di forza che le aree di miglioramento della soluzione proposta. Sebbene i database colonnari come ClickHouse dimostrino una superiorità in termini di prestazioni, la configurazione del server e le specifiche hardware hanno influito negativamente su alcune metriche di performance. Tuttavia, l'architettura client-side della dashboard ha mostrato vantaggi significativi in termini di interattività e reattività, suggerendo un percorso promettente per ottimizzazioni future e l'affermazione di questa soluzione nel panorama delle applicazioni di analisi dei dati.

5.3 Valutazione dei Miglioramenti Introdotti dal Framework

Alla luce di quanto analizzato è possibile valutare i significativi miglioramenti introdotti dal framework prototipale proposto in questo lavoro di tesi per la creazione della Web Dashboard. Questi miglioramenti si manifestano non solo in termini di prestazioni, ma anche in relazione alla funzionalità e all'esperienza utente complessiva.

Una delle innovazioni principali è l'architettura client-side, che consente di eseguire operazioni computazionali direttamente nel browser. Questa scelta ha portato a una riduzione dei tempi di risposta alle interazioni degli utenti, migliorando così l'usabilità della dashboard. In particolare, la metrica Interaction to Next Paint ha mostrato miglioramenti evidenti, evidenziando come il framework consenta di effettuare calcoli e visualizzazioni in tempo reale, senza la necessità di inviare continuamente richieste al server. Ciò non solo velocizza l'interazione, ma aumenta anche la soddisfazione dell'utente, rendendo l'esperienza complessivamente più fluida e reattiva.

Inoltre, l'integrazione di WebAssembly e Rust ha offerto un altro vantaggio fondamentale. La capacità di gestire operazioni computazionali intensive direttamente nel browser, senza compromettere le performance, ha reso la dashboard più versatile e adatta a gestire grandi volumi di dati. Utilizzando Apache Arrow per la serializzazione dei dati, si è potuto ottimizzare ulteriormente la gestione delle informazioni, migliorando l'efficienza e la velocità di elaborazione.

Un ulteriore aspetto da considerare è l'implementazione della comunicazione in tempo reale tramite WebSocket. Questo approccio ha consentito un flusso

continuo di aggiornamenti dati, rendendo la dashboard particolarmente adatta per applicazioni che richiedono l'analisi di informazioni in tempo reale. L'efficienza di questa comunicazione è stata evidenziata nei test, dove si è osservato come i dati vengano trasmessi in modo fluido, riducendo al minimo il rischio di congestione o ritardi.

D'altro canto, nonostante i miglioramenti tangibili, sono emerse anche alcune aree in cui è possibile un'ulteriore ottimizzazione. Ad esempio, le performance della dashboard sono state influenzate dalle specifiche hardware della macchina utilizzata durante i test. Una configurazione più potente potrebbe ridurre ulteriormente i tempi di caricamento e migliorare altre metriche di performance, come il First Contentful Paint e il Largest Contentful Paint. Pertanto, è fondamentale considerare l'infrastruttura hardware come un fattore chiave per l'ottimizzazione futura della dashboard.

In conclusione, il framework prototipale ha introdotto miglioramenti significativi che hanno impattato positivamente sull'efficacia e l'efficienza della Web Dashboard. L'architettura client-side, l'uso di tecnologie avanzate come WebAssembly e Rust, e la gestione della comunicazione in tempo reale tramite WebSocket rappresentano innovazioni cruciali per garantire un'esperienza utente ottimale. Tuttavia, è importante continuare a esplorare opportunità di miglioramento, soprattutto in termini di prestazioni, per garantire che la dashboard possa soddisfare le esigenze sempre crescenti degli utenti e degli scenari di utilizzo in tempo reale.

Metrica	**Valore misurato**	**Fascia di prestazioni**
Page Load Time	4.49 s	Non accettabile
First Contentful Paint	2.67 s	Possibili miglioramenti
Largest Contentful Paint	2.79 s	Possibili miglioramenti
Interaction to Next Paint	0.87 s	Non accettabile
Server Response Time	1.81 s	Non accettabile

Tabella 5.1: Risultati Grafana-SQLite-Chromium

Metrica	**Valore misurato**	**Fascia di prestazioni**
Page Load Time	4.13 s	Non accettabile
First Contentful Paint	0.1 s	Ideale
Largest Contentful Paint	3.45 s	Possibili Miglioramenti
Interaction to Next Paint	0.93 s	Non accettabile
Server Response Time	1.77 s	Possibili miglioramenti

Tabella 5.2: Risultati Grafana-SQLite-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.57 s	Non accettabile
First Contentful Paint	2.64 s	Possibili miglioramenti
Largest Contentful Paint	2.73 s	Possibili miglioramenti
Interaction to Next Paint	0.8 s	Non accettabile
Server Response Time	1.9 s	Non accettabile

Tabella 5.3: Risultati Grafana-SQLite-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.61 s	Non accettabile
First Contentful Paint	2.74 s	Possibili miglioramenti
Largest Contentful Paint	2.85 s	Possibili miglioramenti
Interaction to Next Paint	0.81 s	Non accettabile
Server Response Time	1.91 s	Non accettabile

Tabella 5.4: Risultati Grafana-SQLite-Edge

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	1.65 s	Ideale
First Contentful Paint	1.03 s	Ideale
Largest Contentful Paint	1.16 s	Ideale
Interaction to Next Paint	0.29 s	Possibili miglioramenti
Server Response Time	0.51 s	Ideale

Tabella 5.5: Risultati Grafana-ClickHouse-Chromium

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	1.63 s	Ideale
First Contentful Paint	0.2 s	Ideale
Largest Contentful Paint	2.44 s	Ideale
Interaction to Next Paint	0.33 s	Possibili miglioramenti
Server Response Time	0.27 s	Ideale

Tabella 5.6: Risultati Grafana-ClickHouse-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	1.55 s	Ideale
First Contentful Paint	0.95 s	Ideale
Largest Contentful Paint	1.07 s	Ideale
Interaction to Next Paint	0.24 s	Possibili miglioramenti
Server Response Time	0.45 s	Ideale

Tabella 5.7: Risultati Grafana-ClickHouse-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	1.56 s	Ideale
First Contentful Paint	1 s	Ideale
Largest Contentful Paint	1.11 s	Ideale
Interaction to Next Paint	0.24 s	Possibili miglioramenti
Server Response Time	0.46 s	Ideale

Tabella 5.8: Risultati Grafana-ClickHouse-Edge

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	5.92 s	Non accettabile
First Contentful Paint	4.2 s	Non accettabile
Largest Contentful Paint	4.2 s	Non accettabile
Interaction to Next Paint	2.6 s	Non accettabile
Server Response Time	3.19 s	Non accettabile

Tabella 5.9: Risultati Metabase-SQLite-Chromium

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.6 s	Non accettabile
First Contentful Paint	3.39 s	Non accettabile
Largest Contentful Paint	3.39 s	Possibili Miglioramenti
Interaction to Next Paint	1.96 s	Non accettabile
Server Response Time	2.21 s	Non accettabile

Tabella 5.10: Risultati Metabase-SQLite-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.02 s	Non accettabile
First Contentful Paint	3.18 s	Non accettabile
Largest Contentful Paint	3.18 s	Possibili Miglioramenti
Interaction to Next Paint	1.97 s	Non accettabile
Server Response Time	2.16 s	Non accettabile

Tabella 5.11: Risultati Metabase-SQLite-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.18 s	Non accettabile
First Contentful Paint	3.05 s	Non accettabile
Largest Contentful Paint	3.05 s	Possibili Miglioramenti
Interaction to Next Paint	1.86 s	Non accettabile
Server Response Time	2.09 s	Non accettabile

Tabella 5.12: Risultati Metabase-SQLite-Edge

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.31 s	Non accettabile
First Contentful Paint	3.37 s	Non accettabile
Largest Contentful Paint	3.37 s	Possibili Miglioramenti
Interaction to Next Paint	1.72 s	Non accettabile
Server Response Time	1.8 s	Possibili Miglioramenti

Tabella 5.13: Risultati Metabase-ClickHouse-Chromium

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.43 s	Non accettabile
First Contentful Paint	3.16 s	Non accettabile
Largest Contentful Paint	3.18 s	Possibili Miglioramenti
Interaction to Next Paint	1.71 s	Non accettabile
Server Response Time	1.83 s	Non accettabile

Tabella 5.14: Risultati Metabase-ClickHouse-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	3.94 s	Possibili Miglioramenti
First Contentful Paint	3.29 s	Non accettabile
Largest Contentful Paint	3.29 s	Possibili Miglioramenti
Interaction to Next Paint	1.89 s	Non accettabile
Server Response Time	1.7 s	Possibili Miglioramenti

Tabella 5.15: Risultati Metabase-ClickHouse-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.06 s	Non accettabile
First Contentful Paint	3.26 s	Non accettabile
Largest Contentful Paint	3.26 s	Possibili Miglioramenti
Interaction to Next Paint	1.74 s	Non accettabile
Server Response Time	1.51 s	Possibili Miglioramenti

Tabella 5.16: Risultati Metabase-ClickHouse-Edge

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	5.33 s	Non accettabile
First Contentful Paint	2.92 s	Possibili Miglioramenti
Largest Contentful Paint	2.99 s	Possibili Miglioramenti
Interaction to Next Paint	0.17 s	Ideale
Server Response Time	-	-

Tabella 5.17: Risultati Dashboard-SQLite-Chromium

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	5.43 s	Non accettabile
First Contentful Paint	3 s	Possibili Miglioramenti
Largest Contentful Paint	3.1 s	Possibili Miglioramenti
Interaction to Next Paint	0.18 s	Ideale
Server Response Time	-	-

Tabella 5.18: Risultati Dashboard-SQLite-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.98 s	Non accettabile
First Contentful Paint	2.88 s	Possibili Miglioramenti
Largest Contentful Paint	2.95 s	Possibili Miglioramenti
Interaction to Next Paint	0.16 s	Ideale
Server Response Time	-	-

Tabella 5.19: Risultati Dashboard-SQLite-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	5.38 s	Non accettabile
First Contentful Paint	3.17 s	Non accettabile
Largest Contentful Paint	3.24 s	Possibili Miglioramenti
Interaction to Next Paint	0.19 s	Ideale
Server Response Time	-	-

Tabella 5.20: Risultati Dashboard-SQLite-Edge

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.82 s	Non accettabile
First Contentful Paint	2.7 s	Possibili Miglioramenti
Largest Contentful Paint	2.72 s	Possibili Miglioramenti
Interaction to Next Paint	0.02 s	Ideale
Server Response Time	-	-

Tabella 5.21: Risultati Dashboard-ClickHouse-Chromium

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.7 s	Non accettabile
First Contentful Paint	2.64 s	Possibili Miglioramenti
Largest Contentful Paint	2.65 s	Possibili Miglioramenti
Interaction to Next Paint	0.02 s	Ideale
Server Response Time	-	-

Tabella 5.22: Risultati Dashboard-ClickHouse-Firefox

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.21 s	Non accettabile
First Contentful Paint	2.79 s	Possibili Miglioramenti
Largest Contentful Paint	2.81 s	Possibili Miglioramenti
Interaction to Next Paint	0.01 s	Non accettabile
Server Response Time	-	-

Tabella 5.23: Risultati Dashboard-ClickHouse-GoogleChrome

Metrica	Valore misurato	Fascia di prestazioni
Page Load Time	4.9 s	Non accettabile
First Contentful Paint	2.91 s	Possibili Miglioramenti
Largest Contentful Paint	3.02 s	Possibili Miglioramenti
Interaction to Next Paint	0.02 s	Ideale
Server Response Time	-	-

Tabella 5.24: Risultati Dashboard-ClickHouse-Edge

Capitolo 6

Conclusioni

In questo ultimo capitolo del lavoro di tesi verranno tratte le conclusioni finali, offrendo una sintesi complessiva del lavoro svolto. Il capitolo inizierà con un'analisi approfondita delle attività e delle scelte metodologiche adottate durante il progetto. Verranno riassunti gli aspetti principali del lavoro effettuato, inclusi i risultati ottenuti e le scoperte significative emerse dalle diverse fasi di sviluppo e testing.

Successivamente, si discuteranno le potenzialità di miglioramento e le opportunità per lavori futuri. Verranno esplorati i possibili sviluppi e le evoluzioni della soluzione proposta, considerando anche le nuove tecnologie emergenti e le sfide future nel campo della gestione e analisi dei dati temporali. Saranno forniti suggerimenti per affinare ulteriormente la dashboard, ottimizzare le performance e ampliare le sue funzionalità.

Infine, si trarranno le conclusioni generali, sintetizzando i principali risultati e contributi del lavoro di tesi. Verrà fornita una riflessione critica sull'impatto e sull'efficacia della soluzione proposta rispetto agli obiettivi iniziali e alle aspettative del progetto. Questo capitolo offrirà una panoramica completa delle scoperte e dei progressi raggiunti, contribuendo a una comprensione chiara e completa del valore del lavoro realizzato.

6.1 Risultati ottenuti

Nel corso della tesi, sono stati conseguiti risultati significativi attraverso un approccio metodico e ben strutturato. All'inizio, è stata condotta un'analisi dettagliata dello stato dell'arte delle Web Dashboard e dei performance testing, con un focus particolare sugli strumenti disponibili e le metriche utilizzate per valutare le loro performance. Questa fase ha fornito un quadro chiaro delle soluzioni esistenti, evidenziando le lacune e le opportunità di miglioramento.

Successivamente, è stata elaborata una metodologia per la scrittura e l'esecuzione dei test sui due tool di riferimento, Grafana e Metabase. I test sono stati progettati per misurare le prestazioni in termini di velocità di elaborazione e qualità dell'interazione con l'utente. I risultati ottenuti da questa fase hanno messo in evidenza non solo i punti di forza di ciascun strumento, ma anche le aree in cui potrebbero essere implementati miglioramenti significativi.

Nel capitolo 4, è stato proposto un nuovo framework basato su WebAssembly, progettato per affrontare le sfide emerse durante l'analisi delle soluzioni esistenti. Il framework si distingue per la sua capacità di gestire l'elaborazione dei dati direttamente nel browser, riducendo la dipendenza dai server remoti e migliorando l'efficienza del trattamento dei dati. La creazione di API adattabili a diversi contesti applicativi ha rappresentato un passo cruciale per aumentare la flessibilità del sistema.

Il capitolo 5 ha visto lo sviluppo di una dashboard interattiva utilizzando il framework proposto. Questa dashboard è stata progettata per gestire in modo efficiente cambiamenti di scala e frequenti ricalcoli derivanti dall'interazione dell'utente, dimostrando così l'applicabilità pratica del framework. I test condotti sulla dashboard hanno confermato le aspettative iniziali, mostrando performance superiori rispetto a Grafana e Metabase, sia in termini di velocità di caricamento che di interattività.

Infine, è stata effettuata un'analisi comparativa dei risultati dei test. I dati raccolti hanno rivelato che la dashboard sviluppata con il nuovo framework non solo ha migliorato la velocità di elaborazione, ma ha anche offerto un'interazione utente più fluida e soddisfacente. Questo confronto ha evidenziato l'efficacia del framework nel superare le limitazioni degli strumenti esistenti, posizionandolo come un'alternativa promettente nel panorama delle Web Dashboard.

In sintesi, i risultati ottenuti nel corso della tesi non solo confermano la validità delle scelte progettuali adottate, ma offrono anche una base solida per ulteriori sviluppi e ottimizzazioni future nel campo della visualizzazione e analisi dei dati temporali.

6.2 Lavori futuri

Guardando al futuro, esistono molte opportunità per espandere e migliorare ulteriormente il framework. Alcune delle aree di sviluppo più promettenti includono:

- **Nuovi Tipi di Visualizzazione**: Sebbene il framework supporti già diversi tipi di grafici, l'aggiunta di nuove visualizzazioni interattive, come mappe geografiche o grafici tridimensionali, potrebbe migliorare ulteriormente la comprensione dei dati, soprattutto in contesti più complessi o con dati spaziali.

- **Ottimizzazione delle Prestazioni**: Sebbene l'integrazione di WebAssembly abbia già migliorato notevolmente le performance, ulteriori ottimizzazioni sono possibili. Ad esempio, si potrebbe implementare una gestione più efficiente della memoria o utilizzare tecniche di caching avanzato per ridurre i tempi di caricamento dei dati e aumentare la reattività del sistema.
- **Integrazione con Sistemi di Monitoraggio Energetico**: Un'altra direzione interessante potrebbe essere l'estensione del framework per supportare l'integrazione con altri sistemi di monitoraggio energetico. Questo potrebbe includere l'aggregazione di dati da più fonti, l'elaborazione di modelli predittivi per l'efficienza energetica o l'uso di sensori IoT per il monitoraggio in tempo reale delle infrastrutture energetiche.
- **Supporto per Piattaforme Mobili**: Un ulteriore miglioramento potrebbe consistere nel rendere il framework completamente responsive, con particolare attenzione alle prestazioni su dispositivi mobili, dove l'esperienza utente e l'ottimizzazione delle risorse sono ancora più critiche.
- **Strumenti di Analisi Avanzata**: Oltre a migliorare la visualizzazione dei dati, sarebbe possibile introdurre strumenti di analisi avanzata direttamente nel framework, come algoritmi di machine learning o analisi predittive, che potrebbero fornire informazioni più approfondite e azioni suggerite basate sui dati.

Queste proposte di sviluppo non solo migliorerebbero l'efficacia del framework, ma lo renderebbero anche un componente essenziale per progetti futuri, in grado di adattarsi a contesti complessi e in rapida evoluzione.

6.3 Conclusioni

In conclusione, il presente lavoro ha contribuito a delineare un nuovo approccio nel campo delle Web Dashboard per la visualizzazione e analisi dei dati temporali. Grazie a un'analisi approfondita e a una serie di test condotti sui tool più diffusi, come Grafana e Metabase, sono stati messi in luce i punti di forza e le debolezze di ciascun sistema. Questi risultati hanno fornito un quadro chiaro delle opportunità di miglioramento, suggerendo che vi sia spazio per innovazioni significative nel modo in cui questi strumenti vengono utilizzati e sviluppati.

La dashboard sviluppata con il nuovo framework ha dimostrato performance superiori, con un caricamento più rapido e una gestione efficiente dei dati in tempo reale. Questi risultati non solo confermano la validità delle scelte progettuali adottate, ma posizionano il framework come una valida alternativa agli strumenti esistenti. L'analisi comparativa ha evidenziato non solo i benefici in termini

di velocità e usabilità, ma anche la flessibilità e la capacità del framework di adattarsi a diverse esigenze e contesti applicativi. La possibilità di personalizzazione e ampliamento delle funzionalità rappresenta un valore aggiunto, rendendo la soluzione non solo efficace, ma anche altamente versatile.

Guardando al futuro, si delineano molte opportunità per ulteriori sviluppi e miglioramenti. Si possono immaginare iniziative che spaziano dalla creazione di nuove visualizzazioni interattive e coinvolgenti all'integrazione di strumenti di analisi avanzata, come algoritmi di machine learning, che potrebbero fornire insights predittivi basati sui dati. Inoltre, l'inclusione di funzionalità per la gestione di flussi di dati provenienti da fonti diverse, come i sensori IoT, potrebbe ampliare notevolmente l'ambito d'applicazione del framework. Queste iniziative non solo miglioreranno l'efficacia della dashboard, ma contribuiranno anche a stabilire il framework come uno strumento innovativo e leader nel panorama della gestione dei dati.

In definitiva, questo progetto rappresenta un importante passo avanti nella ricerca e nello sviluppo di soluzioni per la visualizzazione dei dati, aprendo la strada a ulteriori innovazioni nel settore. Le sfide future saranno affrontate con entusiasmo e determinazione, con l'obiettivo di migliorare continuamente la qualità e l'efficacia degli strumenti di analisi disponibili. La crescente importanza della data analytics e della visualizzazione efficace dei dati rende questo lavoro non solo rilevante, ma anche essenziale per le organizzazioni che cercano di prendere decisioni informate in un contesto in continua evoluzione. Con l'impegno costante per l'innovazione e la qualità, il framework ha il potenziale di diventare una risorsa fondamentale per gli analisti e i professionisti del settore.

Bibliografia

[1] Grafana Labs. *Grafana*. 2024. URL: https://grafana.com (cit. a p. 9).

[2] Metabase. *Metabase*. 2024. URL: https://metabase.com (cit. a p. 10).

[3] *WebAssembly*. 2024. URL: https://webassembly.org/ (cit. a p. 13).

[4] Kavindu Premachandra. «WebAssembly in modern web technology: Analysis of benefits vs challenges». In: *International Journal of Science and Research (IJSR)* (2023). URL: https://www.researchgate.net/publication/378901628_WebAssembly_in_modern_web_technology_Analysis_of_benefits_vs_challenges (cit. a p. 16).

[5] V8 Blog. *WebAssembly GC: A fundamental change to V8's JavaScript and WebAssembly engine*. https://v8.dev/blog/wasm-gc-porting. 2023 (cit. a p. 16).

[6] Simon Pietro Romano Gaetano Perrone. *WebAssembly and Security: a review*. Lug. 2024. URL: https://www.researchgate.net/publication/382332042_WebAssembly_and_Security_a_review (cit. a p. 16).

[7] Magnus Medin. «Performance comparison between C and Rust compiled to WebAssembly». Bachelor's thesis. UMEÅ, Sweden: UMEÅ UNIVERSITY, Department of Computing Science Report, giu. 2021. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-185689 (cit. a p. 16).

[8] Rishi Kiran Aiyatham Prabakar. «WebAssembly Performance Analysis: A Comparative Study of C++ and Rust Implementations». Bachelor's thesis. Karlskrona, Sweden: Faculty of Computing, Blekinge Institute of Technology, mar. 2024. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:bth-26632 (cit. a p. 16).

[9] The Rust Project Developers. *The Rust Programming Language*. 2024. URL: https://www.rust-lang.org/ (cit. a p. 16).

[10] Polars. *Polars*. 2024. URL: https://pola.rs/ (cit. a p. 17).

[11] The React Team. *React*. 2024. URL: https://react.dev/ (cit. a p. 18).

[12] Mike Bostock. *D3.js*. 2024. URL: https://d3js.org/ (cit. a p. 18).

[13] Shravan Pargaonkar. «A Comprehensive Review of Performance Testing Methodologies and Best Practices: Software Quality Engineering». In: *International Journal of Science and Research (IJSR)* 12.8 (2023). URL: https://www.researchgate.net/publication/375450774_A_Comprehensive_Review_of_Performance_Testing_Methodologies_and_Best_Practices_Software_Quality_Engineering (cit. a p. 32).

[14] Microsoft Corporation. *Playwright: End-to-End Testing for Modern Web Apps*. 2024. URL: https://playwright.dev/ (cit. a p. 35).

[15] Kush Hiren Brahmbhatt. «Comparative Analysis of Selecting a Test Automation Framework for an E-commerce Website». Master's thesis. Tallinn, Estonia: Tallinn University of Technology, School of Information Technologies, 2024. URL: https://www.etis.ee/Portal/Mentorships/Display/facacd87-4b16-4850-bbee-47420e879728 (cit. a p. 36).

[16] Web.dev. *User-Centric Performance Metrics*. 2024. URL: https://web.dev/articles/user-centric-performance-metrics (cit. a p. 37).

[17] Web.dev. *First Contentful Paint (FCP)*. 2024. URL: https://web.dev/articles/fcp (cit. a p. 37).

[18] Web.dev. *Largest Contentful Paint (LCP)*. 2024. URL: https://web.dev/articles/lcp (cit. a p. 37).

[19] Web.dev. *Interaction to Next Paint (INP)*. 2024. URL: https://web.dev/articles/inp (cit. a p. 38).

[20] Web.dev. *Time to First Byte (TTFB)*. 2024. URL: https://web.dev/articles/ttfb (cit. a p. 38).

[21] World Wide Web Consortium (W3C). *Performance Timeline*. A cura di Nicolás Peña Moreno (Google). W3C Candidate Recommendation Draft 16 February 2024. Former editors: Ilya Grigorik (Google), Jatinder Mann (Microsoft Corp.), Zhiheng Wang (Google). 2024. URL: https://www.w3.org/TR/2024/CRD-performance-timeline-20240216/ (cit. a p. 42).

[22] World Wide Web Consortium (W3C). *Navigation Timing Level 2*. A cura di Yoav Weiss (Google), Noam Rosenthal (Invited Expert). W3C Working Draft 29 July 2024. Former editors: Ilya Grigorik (Google), Tobin Titus (Microsoft Corp.) - Until 01 January 2015, Jatinder Mann (Microsoft Corp.) - Until 01 February 2014, Arvind Jain (Google Inc.) - Until 01 December 2014. 2024. URL: https://www.w3.org/TR/2024/WD-navigation-timing-2-20240729/ (cit. a p. 42).

[23] World Wide Web Consortium (W3C). *Paint Timing*. A cura di Ian Clelland (Google), Noam Rosenthal (Invited Expert). W3C Working Draft 12 January 2024. Former editors: Shubhie Panicker (Google), Nicolás Peña Moreno (Google). 2024. URL: https://www.w3.org/TR/2024/WD-paint-timing-20240112/ (cit. a p. 43).

[24] World Wide Web Consortium (W3C). *Largest Contentful Paint*. A cura di Yoav Weiss (Google), Nicolás Peña Moreno (Google). W3C Working Draft 15 January 2024. 2024. URL: https://www.w3.org/TR/2024/WD-largest-contentful-paint-20240115/ (cit. a p. 43).

[25] World Wide Web Consortium (W3C). *User Timing Level 3*. A cura di Nicolás Peña Moreno (Google). W3C Candidate Recommendation Draft 14 November 2023. Formers editors: Ilya Grigorik (Google), Jatinder Mann (Microsoft Corp.) - Until February 2014, Zhiheng Wang (Google Inc.) - Until July 2013, Anderson Quach (Microsoft Corp.) - Until March 2011. 2023. URL: https://www.w3.org/TR/2023/CRD-user-timing-20231114/ (cit. a p. 44).

[26] World Wide Web Consortium (W3C). *High Resolution Time*. A cura di Yoav Weiss (Google LLC). W3C Working Draft 19 July 2023. Formers editors: Ilya Grigorik (Google LLC) - Until 01 March 2021, James Simonsen (Google LLC) - Until 01 January 2015, Jatinder Mann (Microsoft Corp.) - Until 01 February 2014. 2023. URL: https://www.w3.org/TR/2023/WD-hr-time-3-20230719/ (cit. a p. 44).

[27] World Wide Web Consortium (W3C). *Server Timing*. A cura di Charles Vazac (Akamai), Ilya Grigorik (Google). W3C Working Draft 11 April 2023. 2023. URL: https://www.w3.org/TR/2023/WD-server-timing-20230411/ (cit. a p. 44).

[28] U.S. Energy Information Administration. *Daily Fuel Type Data for Electricity*. 2024. URL: https://www.eia.gov/opendata/browser/electricity/rto/daily-fuel-type-data?frequency=daily&data=value;&sortColumn=period;&sortDirection=desc (cit. a p. 45).