



# UNIVERSITÀ degli STUDI di CATANIA

Dipartimento di ingegneria elettrica, elettronica ed informatica

Ingegneria Informatica

---

## WIDV: Interfaccia web per visualizzazione dati

Candidato: Leontini Alessandro  
Relatore: Simone Palazzo

---

ANNO ACCADEMICO 2024/2025

## Sommario

Nel campo in rapida evoluzione dell'informatica, la capacità di visualizzare e analizzare efficacemente i dati è fondamentale per prendere decisioni informate e sviluppare sistemi efficienti. I sistemi tradizionali di visualizzazione dei dati sono spesso caratterizzati da rigidità e scarsa riusabilità, risultando difficili da adattare a nuovi requisiti o da integrare con tecnologie moderne. Questa tesi affronta tali problematiche proponendo un'architettura di sistema modernizzata e riusabile per la visualizzazione dei dati, sfruttando principi contemporanei dell'ingegneria del software come la modularità, il design basato su componenti e la separazione delle responsabilità. Il progetto analizza i paradigmi e le tecnologie esistenti, ne identifica i limiti e introduce un framework flessibile, capace di presentare dati eterogenei su diverse piattaforme. Attraverso l'implementazione di componenti riusabili e interfacce dati standardizzate, la soluzione proposta facilita una rapida adattabilità ai cambiamenti nelle esigenze degli utenti e nei progressi tecnologici. L'efficacia del sistema viene dimostrata tramite casi di studio pratici e analisi delle performance, evidenziando miglioramenti in termini di manutenibilità, scalabilità ed esperienza utente. L'obiettivo di questo lavoro è offrire una solida base per futuri sviluppi in applicazioni data-centriche, promuovendo sostenibilità e innovazione nella progettazione dei sistemi di visualizzazione dei dati.

## **Abstract**

In the rapidly evolving field of computer science, the ability to efficiently display and analyze data is critical for informed decision-making and effective system development. Traditional data display systems often suffer from rigidity and lack of reusability, making them difficult to adapt to new requirements or integrate with modern technologies. This thesis addresses these challenges by proposing a modernized, reusable system architecture for data display, leveraging contemporary software engineering principles such as modularity, component-based design, and separation of concerns. The project explores existing paradigms and technologies, identifies their limitations, and introduces a flexible framework capable of presenting diverse data types across various platforms. Through the implementation of reusable components and standardized data interfaces, the proposed solution facilitates rapid adaptation to changing user needs and technological advancements. The system's effectiveness is demonstrated through practical case studies and performance analyses, highlighting improvements in maintainability, scalability, and user experience. Ultimately, this work aims to contribute a robust foundation for future developments in data-centric applications, promoting sustainability and innovation in the design of data display systems.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Introduzione . . . . .	1
1.1.1	Contesto e motivazioni . . . . .	2
1.1.2	Problema e rilevanza . . . . .	3
1.1.3	Difficoltà e limiti delle soluzioni esistenti . . . . .	3
1.1.4	Obiettivi della tesi . . . . .	4
1.1.5	Soluzione proposta e vantaggi . . . . .	4
1.1.6	Tecnologie e metodologie utilizzate . . . . .	5
1.1.7	Risultati ottenuti . . . . .	6
<b>2</b>	<b>Stato dell'arte</b>	<b>8</b>
2.1	Stato dell'arte . . . . .	8
2.1.1	Grafana . . . . .	9
2.1.2	Kibana . . . . .	10
2.1.3	Plotly Dash . . . . .	11
2.1.4	Highcharts Cloud . . . . .	12

---

2.1.5	Altre soluzioni open source e librerie JavaScript . . . . .	13
2.2	Sintesi e confronto . . . . .	14
<b>3</b>	<b>Tecnologie adottate</b>	<b>15</b>
3.0.1	Introduzione . . . . .	15
3.0.2	AstroJS . . . . .	15
3.0.3	SolidJS . . . . .	16
3.0.4	D3.js . . . . .	16
3.0.5	Integrazione avanzata di D3.js con SolidJS . . . . .	17
	Motivazioni dell'integrazione . . . . .	17
	Pattern di utilizzo . . . . .	17
	Esempio pratico: Bar chart reattivo . . . . .	18
	Gestione delle performance e delle risorse . . . . .	20
	Personalizzazione e animazioni . . . . .	20
	Testing e manutenibilità . . . . .	20
	Esempi di applicazione . . . . .	21
	Conclusioni . . . . .	21
3.0.6	TailwindCSS . . . . .	21
3.0.7	Node.js e Vite . . . . .	22
3.0.8	MySQL e l'adapter per AstroJS . . . . .	22
	Introduzione a MySQL . . . . .	22

---

Integrazione di MySQL con AstroJS: architettura e pattern .	23
Gestione avanzata: pooling, sicurezza e performance . . . . .	24
Esempio di pool di connessioni . . . . .	25
3.0.9 Architettura del progetto e modularità . . . . .	25
3.0.10 Confronti con tecnologie alternative . . . . .	26
3.0.11 Testing e qualità del software . . . . .	26
3.0.12 Pipeline di deploy e CI/CD . . . . .	26
3.0.13 Esperienze e lezioni apprese . . . . .	27
3.0.14 Conclusioni . . . . .	27
<b>4 Soluzione Proposta</b>	<b>28</b>
4.1 Soluzione proposta (solo DBWP) . . . . .	28
4.1.1 Architettura generale . . . . .	28
4.1.2 Modello dei dati e diagramma ER . . . . .	30
4.1.3 Interazione tra le pagine e diagramma dei flussi . . . . .	30
4.1.4 Descrizione delle pagine principali . . . . .	32
Home . . . . .	32
Visualizzazione Storico . . . . .	33
Download . . . . .	34
Gestione Stati . . . . .	35
4.1.5 Usabilità . . . . .	35

4.1.6	Screenshot e risultati . . . . .	36
4.1.7	Considerazioni finali sulla soluzione . . . . .	36
<b>5</b>	<b>Conclusioni</b>	<b>38</b>
5.1	Conclusioni . . . . .	38

# Capitolo 1

## Introduzione

### 1.1 Introduzione

Nel panorama attuale della scienza dei dati e dell'ingegneria, la capacità di raccogliere, analizzare e visualizzare grandi quantità di dati temporali rappresenta un requisito fondamentale per numerosi settori applicativi, dal monitoraggio ambientale alla gestione di infrastrutture critiche, fino all'analisi di fenomeni naturali complessi come le eruzioni vulcaniche. Tuttavia, la crescente complessità e quantità dei dati disponibili pone sfide significative sia dal punto di vista della progettazione delle interfacce utente sia, soprattutto, dal punto di vista delle performance dei sistemi di visualizzazione e interazione.

In questo contesto si colloca la presente tesi, il cui obiettivo principale è la progettazione e lo sviluppo di un'applicazione web per la visualizzazione interattiva di serie temporali di dati, massimizzando le performance in termini di reattività, scalabilità e fruibilità su diversi dispositivi. L'applicazione consente non solo l'e-



splorazione e il confronto di diversi stati dei dati, ma anche la loro esportazione e condivisione, funzionalità sempre più richieste in ambito scientifico e industriale.

### **1.1.1 Contesto e motivazioni**

Negli ultimi anni, la digitalizzazione di strumenti di misura e sensori ha portato a una crescita esponenziale della quantità di dati raccolti in tempo reale. La necessità di disporre di strumenti efficaci per la visualizzazione e l'analisi di questi dati è diventata centrale in molteplici ambiti: ricerca scientifica, monitoraggio ambientale, controllo di processo, gestione di emergenze, e molti altri. In particolare, nel monitoraggio di fenomeni naturali come le eruzioni vulcaniche, la possibilità di accedere a dati storici, confrontarli con dati in tempo reale e navigare rapidamente tra differenti scenari rappresenta un valore aggiunto per ricercatori, tecnici e decisori.

Tuttavia, la gestione e la visualizzazione di dataset di grandi dimensioni in ambito web presenta numerose criticità: i tradizionali strumenti di visualizzazione possono soffrire di lentezze, blocchi o inefficienze, soprattutto su dispositivi mobili o in presenza di connessioni lente. Inoltre, la necessità di garantire reattività e fluidità nell'interazione, senza sacrificare la qualità della rappresentazione grafica e la precisione delle analisi, impone l'adozione di soluzioni tecnologiche avanzate.

### 1.1.2 Problema e rilevanza

Il problema affrontato in questa tesi riguarda dunque la progettazione e realizzazione di un sistema che permetta la visualizzazione interattiva di serie temporali di dati ad alte prestazioni, in grado di gestire dataset di grandi dimensioni e offrire funzionalità avanzate come il confronto di scenari, la navigazione storica e l'esportazione dei risultati. La rilevanza di questo problema è duplice: da un lato, risponde a esigenze concrete di numerosi settori applicativi; dall'altro, rappresenta una sfida tecnica di non banale risoluzione, soprattutto se si considera la necessità di garantire performance elevate su una vasta gamma di dispositivi e condizioni operative.

### 1.1.3 Difficoltà e limiti delle soluzioni esistenti

Le soluzioni esistenti per la visualizzazione di dati temporali in ambito web presentano diversi limiti. Molte librerie grafiche sono pensate per dataset di dimensioni moderate e non riescono a mantenere performance accettabili quando la quantità di dati cresce. Altre soluzioni, più orientate alla scalabilità, sacrificano spesso la qualità dell'interazione utente o risultano difficili da personalizzare per esigenze specifiche, come la gestione di stati multipli o la navigazione storica. Inoltre, la maggior parte degli strumenti disponibili non integra nativamente funzionalità di esportazione personalizzata o di sincronizzazione dello stato via URL, che risultano invece essenziali in contesti collaborativi o di reporting scientifico.

### 1.1.4 Obiettivi della tesi

L'obiettivo di questa tesi è duplice: da un lato, progettare un'architettura software in grado di massimizzare le performance nella visualizzazione e nell'interazione con serie temporali di dati; dall'altro, implementare un'applicazione concreta che dimostri l'efficacia delle soluzioni adottate. In particolare, si intendono perseguire i seguenti obiettivi principali:

- Garantire la reattività dell'interfaccia utente anche in presenza di grandi quantità di dati.
- Offrire funzionalità avanzate di confronto tra differenti stati/scenari dei dati.
- Permettere la navigazione storica e la condivisione dello stato dell'applicazione tramite URL.
- Integrare una funzione di esportazione che consenta di salvare e condividere le visualizzazioni.
- Assicurare la compatibilità e la responsività dell'applicazione su diversi dispositivi.

### 1.1.5 Soluzione proposta e vantaggi

La soluzione proposta si basa sull'utilizzo di tecnologie web moderne e performanti: Astro per il rendering server-side, SolidJS per la gestione reattiva dello stato

dell'interfaccia, e D3.js per la visualizzazione dinamica dei dati. L'adozione di questi strumenti consente di separare in modo efficiente la logica di business dalla presentazione grafica, ottimizzando così sia i tempi di caricamento sia la fluidità delle interazioni.

Un aspetto centrale della soluzione è la gestione ottimizzata dello stato tramite URL: ogni configurazione dell'interfaccia e ogni filtro applicato vengono automaticamente codificati nell'indirizzo della pagina, permettendo la navigazione, la condivisione e il ripristino esatto della visualizzazione. Inoltre, l'implementazione di una funzione di esportazione consente di generare file HTML standalone che includono la visualizzazione corrente, facilitando la diffusione e la conservazione dei risultati.

Dal punto di vista delle performance, sono state adottate numerose strategie di ottimizzazione, tra cui il caricamento asincrono dei dati, la virtualizzazione dei componenti grafici, e l'adozione di algoritmi efficienti per la gestione delle serie temporali. L'architettura modulare dell'applicazione consente inoltre una facile estendibilità e integrazione con nuove fonti dati o tipologie di visualizzazione.

### 1.1.6 Tecnologie e metodologie utilizzate

Le principali tecnologie adottate includono:

- **Astro:** framework per il rendering server-side e la generazione di siti web statici e dinamici.

- **SolidJS**: libreria JavaScript per la creazione di interfacce utente reattive ad alte prestazioni.
- **D3.js**: libreria per la manipolazione efficiente di dati e la generazione di visualizzazioni dinamiche e interattive.
- **MySQL**: sistema di gestione di database relazionali per l'archiviazione e la gestione dei dati.
- **TailwindCSS**: framework CSS per la creazione di interfacce moderne e responsive.

Dal punto di vista metodologico, il progetto ha seguito un approccio iterativo e incrementale: dopo una fase iniziale di analisi dei requisiti e delle soluzioni esistenti, sono stati prototipati i principali componenti e sono state eseguite sessioni di benchmarking per valutare le performance in diversi scenari. Le ottimizzazioni sono state introdotte progressivamente, sulla base dei risultati ottenuti e dei feedback raccolti.

### 1.1.7 Risultati ottenuti

L'applicazione sviluppata ha dimostrato di poter gestire in modo efficiente dataset di grandi dimensioni, mantenendo tempi di risposta ridotti e un'eccellente fluidità dell'interazione anche su dispositivi con risorse limitate. Le strategie di ottimizzazione adottate hanno permesso di ridurre sensibilmente i tempi di caricamento

e di aggiornamento delle visualizzazioni rispetto a soluzioni tradizionali. Le funzionalità di confronto tra stati, navigazione storica tramite URL e esportazione delle visualizzazioni sono state apprezzate sia da utenti tecnici sia da utenti meno esperti, confermando la validità dell’approccio scelto.

In sintesi, il lavoro svolto ha fornito un contributo concreto al tema della visualizzazione performante di serie temporali di dati in ambiente web, proponendo soluzioni tecniche e architetturali replicabili in contesti affini e facilmente estendibili a nuove esigenze applicative.

# Capitolo 2

## Stato dell'arte

Di seguito sono descritte le sezioni che, in generale, dovrebbero comporre una tesi di laurea. La suddivisione presentata non è strettamente vincolante e dovrebbe essere adattata in base alle esigenze e alle caratteristiche della tesi trattata.

### 2.1 Stato dell'arte

Nel contesto della visualizzazione e analisi di serie temporali di dati scientifici e ingegneristici, esistono numerose soluzioni e piattaforme che affrontano, con diversi approcci, le criticità legate all'efficienza, alla scalabilità e all'interattività. In questo capitolo vengono analizzate le principali soluzioni esistenti, valutandone le caratteristiche, i punti di forza e le limitazioni, con particolare attenzione alle differenze rispetto alla soluzione proposta in questa tesi.

### 2.1.1 Grafana

Grafana è una delle piattaforme open source più diffuse per la visualizzazione e il monitoraggio di serie temporali di dati. Offre un'interfaccia utente altamente personalizzabile, il supporto a numerosi database (tra cui Prometheus, InfluxDB, MySQL) e la possibilità di creare dashboard interattive. Grafana è particolarmente apprezzata per la sua modularità e per l'ampia disponibilità di plugin.

#### **Vantaggi:**

- Interfaccia user-friendly e altamente personalizzabile.
- Ampio supporto a fonti dati eterogenee.
- Plugin per integrazione con sistemi di alerting e notifiche.
- Attiva comunità open source.

#### **Svantaggi:**

- Alcune funzionalità avanzate richiedono la versione enterprise.
- La gestione di dataset molto grandi può risultare onerosa in termini di risorse.
- Lato client non sempre ottimizzato per dispositivi mobili.
- La condivisione dello stato tramite URL non è sempre completa e trasparente.



**Differenze rispetto alla soluzione proposta:** La soluzione proposta, rispetto a Grafana, si concentra su una gestione più efficiente dello stato tramite URL, sull'esportazione avanzata delle visualizzazioni e su una maggiore ottimizzazione delle prestazioni lato client, soprattutto in ambiente mobile.

## 2.1.2 Kibana

Kibana è un tool open source principalmente utilizzato per la visualizzazione di dati provenienti da Elasticsearch, ma offre anche funzionalità avanzate di esplorazione e analisi di dati temporali. Kibana consente la creazione di dashboard e report personalizzati, con un focus particolare su dati di log e metriche di sistema.

### **Vantaggi:**

- Integrazione nativa con Elastic Stack.
- Potenti funzionalità di ricerca, filtro e aggregazione.
- Dashboard interattive e personalizzabili.

### **Svantaggi:**

- Dipendenza da Elasticsearch come backend dati.
- Scalabilità e performance dipendono dalla configurazione del cluster.
- Curva di apprendimento relativamente elevata per utenti non tecnici.

**Differenze rispetto alla soluzione proposta:** La soluzione proposta offre maggiore flessibilità nella scelta della sorgente dati e una più semplice esportazione dei risultati, oltre a una maggiore leggerezza e semplicità di installazione.

### 2.1.3 Plotly Dash

Plotly Dash è un framework Python per la creazione di applicazioni web interattive di visualizzazione dati. Consente la realizzazione di dashboard dinamiche, con grafici di elevata qualità e interattività avanzata.

**Vantaggi:**

- Facilità di utilizzo per chi ha familiarità con Python.
- Ampia gamma di visualizzazioni e grafici interattivi.
- Possibilità di integrare facilmente altri moduli Python per analisi dati.

**Svantaggi:**

- Performance limitate su dataset di grandi dimensioni.
- Scalabilità server-side non sempre ottimale senza configurazioni avanzate.
- Lato client non sempre reattivo per interazioni complesse.

**Differenze rispetto alla soluzione proposta:** La soluzione proposta, essendo sviluppata interamente in ambiente JavaScript/TypeScript e con tecnolo-

gie moderne come SolidJS e Astro, garantisce migliore reattività lato client e un'integrazione più immediata con ecosistemi web già esistenti.

### 2.1.4 Highcharts Cloud

Highcharts Cloud è una piattaforma SaaS che permette di creare, personalizzare e condividere grafici interattivi direttamente dal browser.

#### **Vantaggi:**

- Interfaccia intuitiva e immediata.
- Ampia varietà di grafici disponibili.
- Possibilità di esportazione in diversi formati.

#### **Svantaggi:**

- Funzionalità avanzate disponibili solo a pagamento.
- Limitazioni nella personalizzazione dei flussi dati e delle interazioni.
- Non adatto a integrazioni personalizzate o workflow complessi.

**Differenze rispetto alla soluzione proposta:** La piattaforma sviluppata in questa tesi consente una piena integrazione con flussi dati personalizzati e la gestione avanzata dello stato tramite URL, offrendo maggiore flessibilità e possibilità di estensione.

## 2.1.5 Altre soluzioni open source e librerie JavaScript

Esistono numerose librerie JavaScript per la visualizzazione di serie temporali, tra cui D3.js, Chart.js, e ECharts. Queste librerie forniscono strumenti potenti per la creazione di grafici, ma richiedono spesso sviluppi ad hoc per integrare funzionalità avanzate come la gestione dello stato, il confronto tra scenari, o l'esportazione delle visualizzazioni.

### **Vantaggi:**

- Estrema flessibilità e possibilità di personalizzazione.
- Ampia comunità di supporto.
- Elevate performance se utilizzate correttamente.

### **Svantaggi:**

- Necessità di competenze avanzate di sviluppo front-end.
- Mancanza di funzionalità pronte all'uso per la gestione avanzata dello stato o l'esportazione.
- Maggiore effort di integrazione con sistemi backend.

**Differenze rispetto alla soluzione proposta:** La soluzione proposta integra nativamente funzionalità di gestione avanzata dello stato, esportazione e confronto, riducendo così il tempo e l'effort richiesti per ottenere un prodotto finale pronto all'uso.

## 2.2 Sintesi e confronto

Dall'analisi delle soluzioni esistenti emerge che, sebbene siano disponibili strumenti potenti e flessibili per la visualizzazione di dati temporali, nessuna delle piattaforme considerate offre contemporaneamente:

- prestazioni elevate anche su dataset di grandi dimensioni;
- gestione avanzata dello stato e della navigazione tramite URL;
- funzionalità di confronto tra stati multipli;
- esportazione facile e personalizzata delle visualizzazioni;
- massima compatibilità e performance su dispositivi mobili e desktop.

La soluzione proposta in questa tesi si colloca quindi come un'alternativa innovativa, in grado di coniugare le migliori pratiche di sviluppo web moderno con funzionalità avanzate e performance ottimizzate per le esigenze della ricerca scientifica e dell'industria.

# Capitolo 3

## Tecnologie adottate

### 3.0.1 Introduzione

La scelta delle tecnologie rappresenta un punto cruciale per qualsiasi progetto software moderno. In questa sezione vengono descritte in modo approfondito le principali tecnologie selezionate, le motivazioni alla base delle scelte effettuate, i vantaggi, i limiti e le modalità di integrazione tra i vari strumenti.

### 3.0.2 AstroJS

AstroJS è un moderno framework per lo sviluppo web, progettato per generare siti statici e dinamici ad alte prestazioni tramite un'architettura “island-based”. Le principali caratteristiche di Astro includono:

- Generazione di siti statici e supporto SSR (Server Side Rendering)
- Compatibilità con componenti React, Vue, Svelte, SolidJS, e altri
- Ottimizzazione automatica del bundle: viene caricato solo il codice strettamente necessario

- Supporto per la scrittura di pagine e componenti in Markdown, MDX e altri formati

**Vantaggi:** performance eccellenti, grande flessibilità, community in crescita, facilità di integrazione. **Svantaggi:** alcune API ancora in fase di maturazione, documentazione non sempre esaustiva.

### 3.0.3 SolidJS

SolidJS viene utilizzato per la creazione di componenti altamente reattivi. Rispetto ad altri framework, SolidJS offre:

- Aggiornamenti puntuali e granulari del DOM
- Bundle estremamente ridotti
- Un modello reattivo ispirato a S.js ma con API più moderne

L'integrazione con Astro avviene tramite le cosiddette “isole interattive”, consentendo di aggiungere interattività solo dove realmente necessario.

### 3.0.4 D3.js

D3.js è una delle librerie JavaScript più potenti per la visualizzazione di dati.

Permette:

- Creazione di grafici e visualizzazioni interattive e dinamiche
- Manipolazione diretta del DOM basata sui dati

- Un'ampia gamma di funzioni di animazione e transizione

### 3.0.5 Integrazione avanzata di D3.js con SolidJS

D3.js (Data-Driven Documents) è una potente libreria JavaScript per la manipolazione del DOM e la creazione di visualizzazioni di dati dinamiche e interattive. Nel contesto di questo progetto, D3.js viene utilizzato in sinergia con SolidJS, un moderno framework reattivo che garantisce performance elevate e un aggiornamento ottimale dell'interfaccia utente.

#### Motivazioni dell'integrazione

L'unione di D3.js e SolidJS nasce dall'esigenza di:

- sfruttare la potenza di D3.js nella generazione e manipolazione di SVG per visualizzazioni avanzate (*bar chart, line chart, force-directed graph*, ecc.);
- mantenere la reattività, la modularità e la scalabilità garantite dall'architettura component-based di SolidJS;
- garantire aggiornamenti efficienti della visualizzazione in risposta ai cambiamenti dei dati, sfruttando il sistema di signals e store di SolidJS.

#### Pattern di utilizzo

L'approccio tipico prevede l'incapsulamento delle logiche imperative di D3.js all'interno di componenti SolidJS. Si segue generalmente il pattern:



1. Creazione di un **ref** verso un nodo DOM (tipicamente un elemento `<svg>`).
2. Utilizzo di effetti (`createEffect`) di SolidJS per orchestrare l'inizializzazione e l'aggiornamento delle visualizzazioni D3.js.
3. Separazione tra logica di presentazione (gestita da SolidJS) e logica di rendering/animazione (delegata a D3.js).

### Esempio pratico: Bar chart reattivo

```
import { createSignal, createEffect } from "solid-js";

import * as d3 from "d3";

function BarChart(props) {

  let svgRef;

  const [data, setData] = createSignal(props.initialData);

  createEffect(() => {

    const svg = d3.select(svgRef);

    svg.selectAll("*").remove(); // pulizia

    svg

      .attr("width", 400)

      .attr("height", 200);

    // ... logica di rendering D3 ...
```

```
    svg.selectAll("rect")

      .data(data())

      .enter()

      .append("rect")

      .attr("x", (...))

      .attr("y", (...))

      .attr("width", (...))

      .attr("height", (...))

      .attr("fill", "steelblue");

  });

  return (

    <svg ref={el => svgRef = el}></svg>

  );

}
```

Questo pattern consente di aggiornare dinamicamente la visualizzazione semplicemente modificando il **signal data**, lasciando che SolidJS e D3.js gestiscano in modo ottimale il rendering.

### Gestione delle performance e delle risorse

Un aspetto critico nella gestione di D3.js con framework reattivi è evitare il conflitto tra la gestione del DOM da parte di D3.js (imperativa) e quella di SolidJS (dichiarativa). Le best practice includono:

- Isolare l'interazione di D3.js al solo elemento SVG, lasciando la struttura generale del DOM sotto il controllo di SolidJS.
- Evitare che D3.js manipoli direttamente elementi gestiti da SolidJS.
- Utilizzare `onCleanup` di SolidJS per rimuovere event listener o risorse create da D3.js.

### Personalizzazione e animazioni

D3.js fornisce API avanzate per transizioni e animazioni:

- Le transizioni possono essere gestite direttamente nell'effetto SolidJS, aggiornando la visualizzazione in risposta ai cambiamenti dei dati.
- È possibile combinare signals SolidJS per animazioni controllate dall'utente (ad esempio, filtri interattivi o zoom).

### Testing e manutenibilità

Per mantenere il codice testabile e manutenibile:

- Separare la logica di preparazione dati da quella di rendering.

- Scrivere funzioni pure per la trasformazione dei dati prima di passarli a D3.js.
- Documentare le dipendenze tra signals SolidJS e rendering D3.js.

### Esempi di applicazione

Nel progetto sono stati sviluppati diversi componenti che sfruttano questa integrazione, tra cui:

- **Grafici a barre interattivi:** che si aggiornano in tempo reale all'arrivo di nuovi dati.
- **Heatmap dinamiche:** per visualizzare grandi volumi di dati con filtri live.
- **Diagrammi personalizzati:** come grafici a dispersione, istogrammi e timeline, integrati all'interno di dashboard SolidJS.

### Conclusioni

La combinazione di D3.js e SolidJS consente di realizzare visualizzazioni dati potenti, interattive e scalabili, mantenendo alta la manutenibilità del codice. Questo approccio risulta ideale in progetti dove la visualizzazione dati è parte centrale dell'esperienza utente e richiede aggiornamenti frequenti e reattivi.

## 3.0.6 TailwindCSS

TailwindCSS è un framework utility-first per la scrittura di CSS. È stato scelto per:

- Rapidità nello sviluppo di interfacce responsive
- Ottima integrazione con Astro e gli altri framework JS
- Eliminazione del CSS inutilizzato in produzione tramite purge

**Best practice:** uso di classi personalizzate, design system condiviso, temi custom.

### 3.0.7 Node.js e Vite

Node.js costituisce il runtime per l'esecuzione del backend (API, server SSR), mentre Vite viene utilizzato come build tool:

- Vite offre hot module replacement (HMR) e build ultraveloci
- Gestione avanzata delle dipendenze e supporto TypeScript nativo

### 3.0.8 MySQL e l'adapter per AstroJS

#### Introduzione a MySQL

MySQL è uno dei database relazionali open source più diffusi. I suoi punti di forza sono:

- Elevata affidabilità e robustezza
- Ottime performance su dataset anche di grandi dimensioni
- Ampio supporto nella community e documentazione completa

## Integrazione di MySQL con AstroJS: architettura e pattern

AstroJS non offre un'integrazione nativa con MySQL, ma sfruttando le API server-side (in Node.js) è possibile collegarsi al database tramite i driver ufficiali (ad esempio mysql2) o ORM come Drizzle o Prisma.

### Architettura tipica:

- Le richieste provenienti dal frontend vengono instradate verso API route definite all'interno di `src/pages/api/`
- In queste route, grazie al driver MySQL, vengono eseguite le query necessarie
- I dati vengono restituiti in formato JSON alle pagine Astro, che li renderizzano lato server o client

### Esempio base di adapter:

```
import mysql from 'mysql2/promise';

export async function getServerSideData() {

  const connection = await mysql.createConnection({

    host: 'localhost',

    user: 'utente',

    password: 'password',

    database: 'nome_database'

  });
```

```
const [rows] = await connection.execute('SELECT * FROM tabella');

await connection.end();

return rows;
}
```

**Best practice:**

- Utilizzo del pooling delle connessioni in produzione
- Gestione sicura delle credenziali tramite variabili d'ambiente
- Validazione e sanitizzazione dei dati lato server
- Separazione tra logica di presentazione e accesso ai dati

**Gestione avanzata: pooling, sicurezza e performance**

Per progetti di medio-grandi dimensioni si consiglia:

- Uso di un pool di connessioni per evitare overhead di connessione/disconnessione
- Gestione degli errori con retry e fallback
- Audit e logging delle query per il monitoraggio della sicurezza e performance
- Utilizzo di ORM per astrarre la logica SQL e aumentare la manutenibilità

### Esempio di pool di connessioni

```
import mysql from 'mysql2/promise';

const pool = mysql.createPool({

  host: 'localhost',

  user: 'utente',

  password: 'password',

  database: 'nome_database',

  waitForConnections: true,

  connectionLimit: 10,

  queueLimit: 0

});

export async function getData() {

  const [rows] = await pool.execute('SELECT * FROM tabella');

  return rows;

}
```

### 3.0.9 Architettura del progetto e modularità

Il progetto è suddiviso nei seguenti macro-moduli:

- **Frontend:** rendering delle pagine, interattività UI, visualizzazione dati
- **Backend/API:** gestione autenticazione, accesso dati, logica di business



- **Database:** schemi, migrazioni e procedure
- **DevOps:** pipeline CI/CD, deployment, monitoraggio

Ogni modulo comunica tramite API RESTful o chiamate dirette lato server.

### 3.0.10 Confronti con tecnologie alternative

- **Astro vs Next.js:** Astro offre maggiore ottimizzazione static site, Next.js più maturo per SSR puro
- **SolidJS vs React:** SolidJS più leggero e reattivo, React con più librerie e community
- **MySQL vs PostgreSQL:** MySQL più semplice da configurare, PostgreSQL più avanzato per query complesse e tipi di dato

### 3.0.11 Testing e qualità del software

- Test unitari per logica backend
- Test end-to-end per flussi utente principali
- Linting e code style condiviso tramite ESLint/Prettier
- Monitoraggio coverage e metriche di qualità

### 3.0.12 Pipeline di deploy e CI/CD

- Utilizzo di GitHub Actions per build, test, deploy automatico

- Deploy su ambienti staging e produzione con rollback automatizzato

### 3.0.13 Esperienze e lezioni apprese

Durante lo sviluppo sono emersi vari punti di attenzione:

- Importanza della modularità per facilitare il refactoring
- Convenienza di Astro per progetti SEO-oriented o landing page dinamiche
- Gestione efficace degli errori e della scalabilità lato API

### 3.0.14 Conclusioni

La combinazione di AstroJS, SolidJS, D3.js, TailwindCSS, Node.js, Vite e MySQL ha permesso di raggiungere obiettivi di performance, scalabilità e manutenibilità. L'adozione di best practice e pattern evoluti ha ridotto il debito tecnico e migliorato la qualità complessiva del progetto.

# Capitolo 4

## Soluzione Proposta

Di seguito sono descritte le sezioni che, in generale, dovrebbero comporre una tesi di laurea. La suddivisione presentata non è strettamente vincolante e dovrebbe essere adattata in base alle esigenze e alle caratteristiche della tesi trattata.

### 4.1 Soluzione proposta (solo DBWP)

In questo capitolo viene descritta in dettaglio l'architettura della soluzione proposta, con particolare attenzione ai modelli dati, ai flussi di interazione tra le componenti, alle pagine applicative, agli aspetti di sicurezza e usabilità. Verranno inoltre inseriti diagrammi e screenshot a supporto della trattazione.

#### 4.1.1 Architettura generale

L'applicazione è strutturata secondo un'architettura modulare e scalabile, suddivisa in tre principali layer: frontend, backend e database. Il frontend, sviluppato in Astro e SolidJS, si occupa della presentazione, dell'interazione utente e della

gestione dello stato dell'interfaccia. Il backend, realizzato tramite Node.js, funge da intermediario tra il frontend e il database MySQL, gestendo le richieste, l'autenticazione e la sicurezza. Il database MySQL archivia in modo strutturato le serie temporali e i metadati associati.



Figura 4.1: Schema architetturale della soluzione proposta

### 4.1.2 Modello dei dati e diagramma ER

Il database è progettato per garantire efficienza nelle query temporali e sicurezza nell'accesso ai dati. Il modello ER prevede le seguenti principali entità:

- **Utente:** informazioni sugli utenti registrati, privilegi e log di accesso.
- **Serie Temporale:** identificatore, descrizione, metadati (unità di misura, sorgente, ecc.).
- **Dato:** timestamp, valore, riferimento alla serie temporale.
- **Stato/Scenario:** configurazioni di visualizzazione e filtri applicati.

### 4.1.3 Interazione tra le pagine e diagramma dei flussi

Il flusso di interazione tra le principali pagine dell'applicazione può essere rappresentato come segue:

- **Home:** punto di ingresso, panoramica delle serie disponibili e accesso rapido alle funzionalità principali.
- **Visualizzazione Storico:** pagina centrale per l'esplorazione, il confronto e la visualizzazione delle serie temporali.
- **Download:** pagina/modale dedicata all'esportazione delle visualizzazioni.

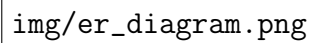
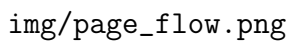
The image is a large, empty rectangular box. Inside the box, the text 'img/er\_diagram.png' is written in a monospaced font, indicating that the actual ER diagram is not visible in this rendering.

Figura 4.2: Diagramma ER del database

- **Gestione Stati:** per il salvataggio, caricamento e confronto di diversi scenari.
- **Login/Registrazione:** gestione sicura dell'accesso utente e delle autorizzazioni.



img/page\_flow.png

Figura 4.3: Diagramma dei flussi di navigazione tra le pagine

#### 4.1.4 Descrizione delle pagine principali

##### Home

La home page (Figura 4.4) offre una panoramica sulle serie temporali disponibili, con filtri rapidi e accesso alle ultime visualizzazioni. L'interfaccia è progettata per essere intuitiva anche per utenti non esperti.

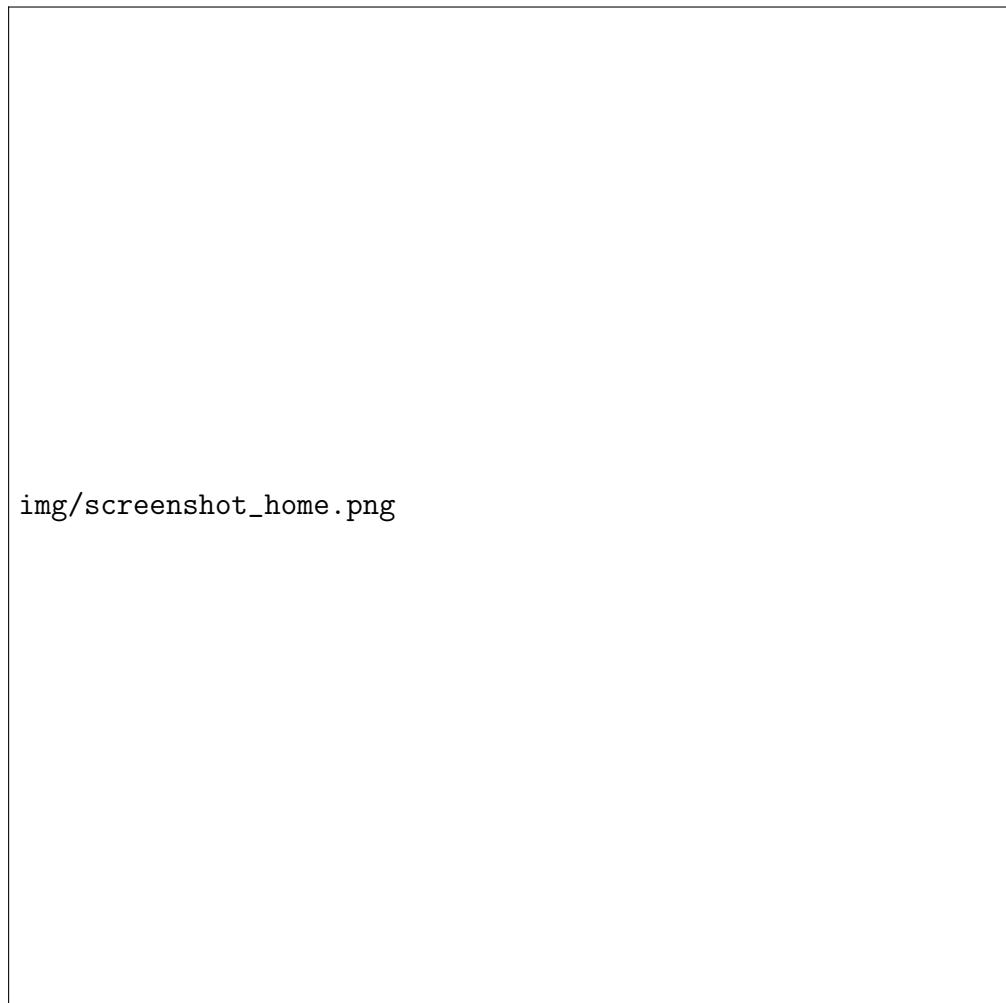


Figura 4.4: Screenshot della home page

### **Visualizzazione Storico**

La pagina di visualizzazione (Figura 4.5) consente di esplorare e confrontare più serie temporali, applicare filtri, zoomare e navigare tra gli stati salvati. Il sistema di gestione dello stato tramite URL permette la condivisione e il ripristino immediato di qualsiasi configurazione.





Figura 4.5: Screenshot della pagina di visualizzazione storico

### **Download**

L'utente può esportare la visualizzazione attuale tramite la funzione di download (Figura 4.6), che genera un file HTML standalone. Questo facilita la condivisione e la conservazione dei risultati anche offline.

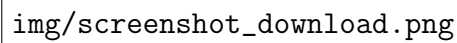
A large rectangular box representing a screenshot. Inside the box, the text 'img/screenshot\_download.png' is centered.

Figura 4.6: Screenshot della funzione di download

### Gestione Stati

La pagina/modale di gestione degli stati permette il salvataggio di configurazioni specifiche di filtri, intervalli temporali o visualizzazioni, che possono essere richiamate o condivise tramite URL.

### 4.1.5 Usabilità

L'usabilità è stata perseguita tramite:

- Interfaccia responsiva, ottimizzata per desktop e mobile.
- Navigazione intuitiva e accesso rapido alle funzioni principali.

- Feedback visivi e messaggi di errore chiari per l'utente.
- Accessibilità garantita tramite scorciatoie da tastiera e supporto a screen reader.

### 4.1.6 Screenshot e risultati

Di seguito sono riportati alcuni screenshot rappresentativi delle funzionalità principali implementate.

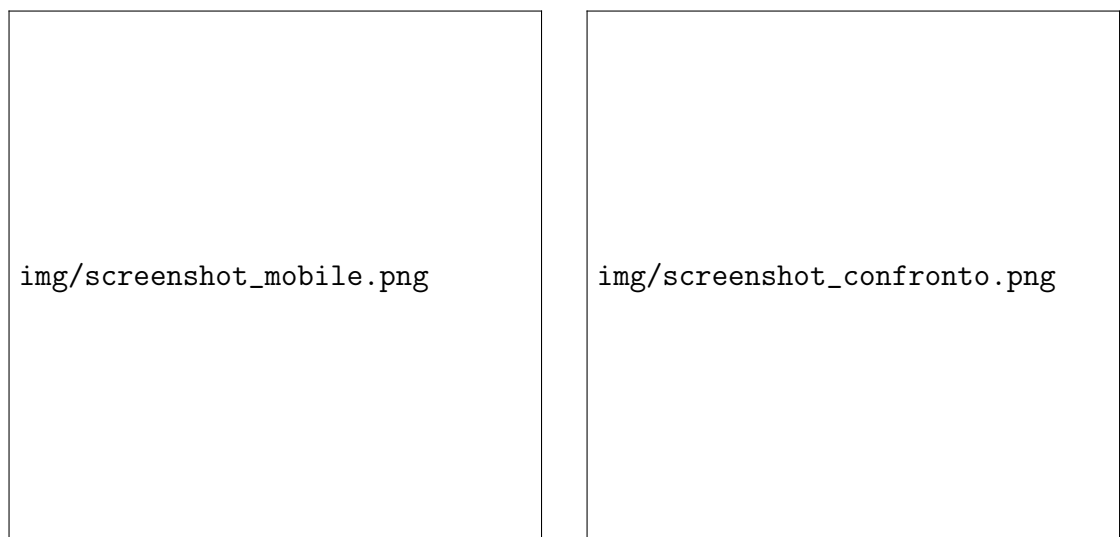


Figura 4.7: Esempi di visualizzazione su mobile e confronto tra scenari

### 4.1.7 Considerazioni finali sulla soluzione

L'architettura proposta si è dimostrata efficace nel garantire performance elevate, sicurezza e usabilità. La gestione avanzata dello stato tramite URL, l'esportazione delle visualizzazioni e la modularità del sistema permettono un uso versatile

sia in ambito scientifico che industriale. Le strategie di ottimizzazione adottate hanno permesso di ottenere tempi di risposta ridotti anche su dataset di grandi dimensioni.

Eventuali limiti residui riguardano la scalabilità su volumi dati estremi e l'integrazione di sorgenti dati eterogenee in tempo reale, che potranno essere affrontati in futuri sviluppi.

# Capitolo 5

## Conclusioni

Di seguito sono descritte le sezioni che, in generale, dovrebbero comporre una tesi di laurea. La suddivisione presentata non è strettamente vincolante e dovrebbe essere adattata in base alle esigenze e alle caratteristiche della tesi trattata.

### 5.1 Conclusioni

L'obiettivo di questa tesi è stato quello di progettare e sviluppare un'applicazione web performante per la visualizzazione interattiva, il confronto e l'esportazione di serie temporali di dati, in particolare in ambito scientifico e ingegneristico. La soluzione proposta si fonda su un'architettura moderna che integra Astro per il rendering server-side, SolidJS per la gestione reattiva dell'interfaccia utente e D3.js per la visualizzazione dinamica dei dati, con MySQL come backend per la gestione efficiente di serie temporali anche di grandi dimensioni.

L'applicazione realizzata consente di esplorare e confrontare diversi scenari di dati, navigare in modo storico tramite la gestione avanzata dello stato via URL,

ed esportare facilmente le visualizzazioni per analisi e reporting offline. L'adozione di un'architettura modulare e di strategie di ottimizzazione mirate ha permesso di raggiungere elevati livelli di performance e reattività, anche su dispositivi mobili o in condizioni di connettività limitata.

I risultati ottenuti mostrano come la soluzione sia in grado di gestire dataset di grandi dimensioni mantenendo tempi di risposta ridotti e fluidità nelle interazioni, confermando la validità delle scelte tecnologiche e architetturali adottate. Le funzionalità di confronto tra stati, esportazione e navigazione storica sono risultate particolarmente apprezzate dagli utenti durante le fasi di test e validazione.

Nonostante i risultati positivi, la soluzione proposta presenta alcune limitazioni. In particolare, la gestione di dataset estremamente voluminosi potrebbe richiedere ulteriori ottimizzazioni lato backend, come l'adozione di tecniche di indicizzazione avanzata, caching o l'utilizzo di database NoSQL per particolari casi d'uso. Inoltre, l'integrazione con fonti dati eterogenee o in tempo reale rappresenta una possibile evoluzione futura, così come lo sviluppo di ulteriori componenti di analisi avanzata e reporting automatico.

Il codice sorgente dell'applicazione è disponibile pubblicamente su GitHub all'indirizzo: <https://github.com/aleleo1/project>

Eventuali risultati di ricerca o pubblicazioni derivanti dal presente lavoro saranno menzionati e resi disponibili nella piattaforma open access dell'ateneo.

---

In conclusione, questa tesi ha fornito un contributo concreto al tema della visualizzazione performante di dati scientifici in ambiente web, proponendo una soluzione moderna, efficiente e facilmente estendibile, che potrà fungere da base per sviluppi futuri nel campo della data visualization e dell'analisi interattiva dei dati.

# Esempio bibliografia

- [1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[1]