

Analisi Tecnica e Documentazione di Progetto - SolarTech Monitor

Sviluppatore: Giovanni Piras

1. Introduzione

Il progetto SolarTech Monitor nasce dalla necessità di fornire a un'azienda di energia solare uno strumento per monitorare lo stato di salute dei propri impianti fotovoltaici. Spesso, un calo di produzione può essere confuso con una giornata nuvolosa; questo software risolve il problema incrociando i dati reali di produzione con quelli meteorologici.

2. Requisiti del Sistema

2.1 Requisiti Funzionali

(ovvero i compiti principali che il sistema deve svolgere):

- **Visualizzazione Dashboard:** Una schermata principale che elenca tutti i parchi solari con i dati aggregati (Produzione totale, Media giornaliera, Efficienza attuale).
- **Integrazione Meteo:** Il sistema deve collegarsi automaticamente a un servizio esterno (Open-Meteo) per recuperare l'irradiazione solare specifica per le coordinate di ogni impianto, in modo da avere un incrocio dei dati che permette di rilevare un eventuale guasto di quest'ultimo.
- **Rilevamento Guasti Intelligente:** Applicazione di una logica di controllo: se il sole è forte ($> 5 \text{ MJ/m}^2$) ma la produzione è minima ($< 10\%$ dell'efficienza), il sistema deve generare un allarme visivo di guasto.
- **Gestione Dati (CRUD):** Possibilità di inserire manualmente nuove letture tramite un form, con selezione facilitata dell'impianto, e immediato aggiornamento della DashBoard.
- **Analisi Storica:** Ogni impianto ha una pagina dedicata con un grafico dell'andamento temporale e una tabella cronologica dello storico di tutte le letture.

2.2 Requisiti Non Funzionali :

- **Usabilità:** L'interfaccia deve essere semplice e pulita, con indicatori visivi colorati (Verde/Giallo/Rosso) per facilitare la lettura rapida.
- **Prestazioni:** Uso di un pool di connessioni al database per rispondere velocemente anche con più richieste simultanee.

- **Resilienza:** Gestione dei valori nulli o delle divisioni per zero nelle query SQL per evitare crash del sistema.
- **Sicurezza:** Separazione delle credenziali sensibili tramite variabili d'ambiente (.env).
- **Design Responsivo:** L'applicazione è stata progettata per adattarsi automaticamente a schermi di diverse dimensioni. Grazie a una struttura a griglia flessibile, la dashboard è perfettamente consultabile sia da computer che da dispositivi mobili (tablet/smartphone).

3. Scelte Tecnologiche e Vantaggi

Per questo progetto ho scelto tecnologie moderne che offrono grandi vantaggi anche a chi è alle prime armi:

Next.js 15 (App Router)

Next.js è stato scelto come framework principale perché permette di sviluppare un'applicazione full-stack in un unico progetto, combinando interfaccia utente, logica server-side e API. Grazie all'App Router e al rendering lato server, il sistema offre elevate prestazioni, maggiore sicurezza e un accesso diretto al database senza la necessità di un backend separato.

MySQL (Database Relazionale)

Ho scelto un database SQL perché i dati degli impianti sono strutturati e legati tra loro (un impianto ha molte letture). Usare MySQL mi ha permesso di delegare i calcoli complessi (come le somme e le medie) direttamente al motore del database tramite il linguaggio SQL. Questo rende l'applicazione molto più efficiente, perché invece di scaricare migliaia di righe e calcolarle con Javascript, chiediamo a MySQL di inviarci direttamente il risultato finale già pronto.

React (Libreria per l'interfaccia)

Ho utilizzato React per costruire l'intera interfaccia utente in modo componente. I vantaggi principali che ho riscontrato sono:

- **Componenti riutilizzabili: Ho creato un componente**

SolarPanelCard che viene ripetuto per ogni impianto, evitando di scrivere lo stesso codice più volte.

- **Stato Dinamico (Hooks):**

Grazie a funzioni come

useState e useEffect

le card si aggiornano da sole non appena i dati meteo arrivano dall'API.

Questo permette di avere una pagina "viva" che non deve essere ricaricata ogni volta.

- **Interfaccia Reattiva:** Se un impianto va in guasto, l'allarme rosso compare istantaneamente perché React si accorge del cambiamento dei dati e aggiorna solo quel pezzetto di schermo.

TypeScript (Il linguaggio) :

Per scrivere il codice non ho usato il classico JavaScript, ma TypeScript. Il vantaggio principale è che TypeScript aggiunge un controllo sui tipi di dati, segnalandomi eventuali errori mentre sto scrivendo il codice (ad esempio se si prova a sommare un numero con una parola). Questo ha reso lo sviluppo più sicuro e ha ridotto drasticamente i bug dell'applicazione.

Tailwind CSS

Per lo stile ho usato Tailwind. Il vantaggio è che non serve scrivere lunghi file CSS separati, ma si possono applicare le classi direttamente nel codice HTML/React. Questo rende lo sviluppo dell'interfaccia molto più veloce e coerente.

Recharts & Lucide React

- **Recharts:**

Mi ha permesso di creare grafici professionali in modo semplice.

- **Lucide:**

Fornisce icone leggibili che aiutano l'utente a capire subito i dati (es. il sole per il meteo, il fulmine per la produzione).

4. Architettura dei Dati

Il cuore del sistema è il database **SolarTech**, strutturato con due tabelle collegate tra loro:

- **Comunicazione tramite API REST:**

Lo scambio di dati tra il frontend e il database avviene tramite **API REST** protette. I dati vengono inviati in formato **JSON**, che è lo standard nell'industria del software. Questo rende il sistema "aperto": in futuro, si potrebbe collegare un'applicazione mobile o un altro servizio esterno a queste stesse API senza dover modificare il database.

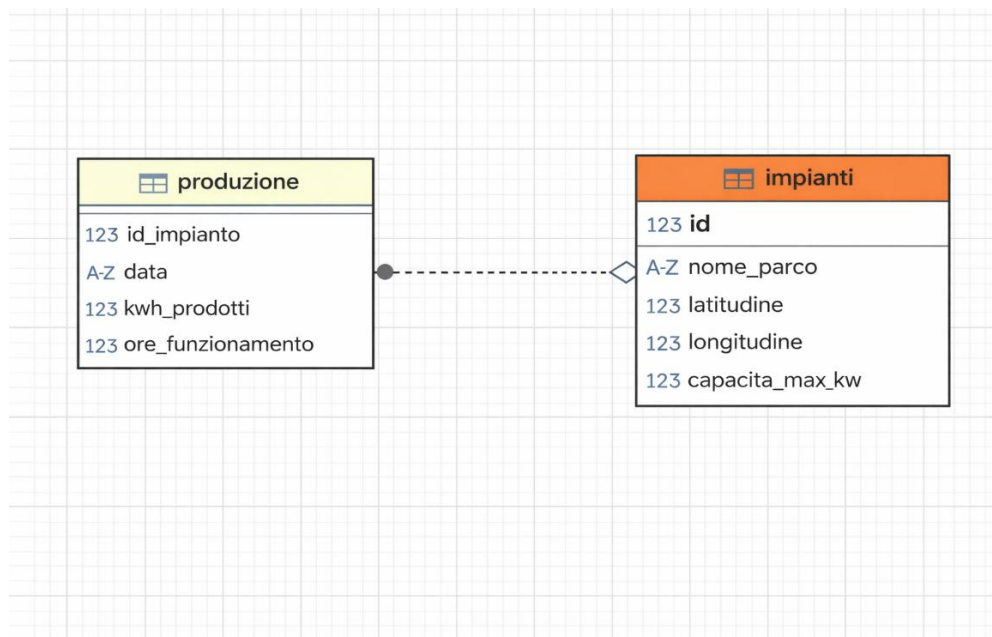
- **Tabella impianti:**

Contiene l'anagrafica (ID, Nome, Coordinate, Capacità Max).

- **Tabella produzione:**

Contiene le letture giornaliere. Ogni lettura è collegata a un impianto tramite una Foreign Key (id_impianto).

Modello ER:



Quest'immagine del modello ER del database è stata creata grazie al Software Draw.io.

La logica di calcolo del Performance Ratio è stata la parte più complessa: viene calcolata rapportando i kWh prodotti alle ore di lavoro e alla capacità massima del pannello, permettendoci di capire quanto l'impianto stia rendendo rispetto al suo potenziale teorico.

5. Logica di Diagnostica

Il punto di forza del progetto è l'algoritmo di rilevamento degli errori.

Esempio reale:

- Se l'API meteo restituisce 8 MJ/m² (bella giornata) ma il database calcola un rendimento del 2%, il sistema capisce che non è colpa del meteo e attiva l'allarme.
- Se invece il rendimento è basso ma l'API meteo dice 1 MJ/m² (molto nuvoloso), il sistema non segnala guasti perché la bassa produzione è normale.

Questo progetto rappresenta una soluzione completa e scalabile per la gestione energetica, unendo analisi dati, sviluppo web e diagnostica automatizzata.

Sommario delle Tecnologie :

| Tecnologia | Ruolo nel Progetto |
|----------------|--|
| Next.js 15 | Framework principale (Telaio) |
| React | Gestione dell'interfaccia e dei componenti |
| TypeScript | Linguaggio di programmazione per evitare errori |
| MySQL | Database per salvare impianti e letture |
| Tailwind CSS | Design grafico e stile moderno |
| Recharts | Generazione dei grafici di produzione |
| Lucide React | Icone tecniche (sole, fulmine, avvisi) |
| Open-Meteo API | Fonte esterna per i dati di irraggiamento solare |