# Iotalab – Progetto TMB demo

Documentazione progetto react lato client

Indice:

* Descrizione generale
* Architettura dei componenti
* Gestione dei contenitori
* WebSocket e aggiornamento dati
* Documentazione del blocco useEffect
* Gestione allarmi (Context API)
* Routing e pagine
* Componenti grafici di controllo
* Componente principale: contenitore

**1. Descrizione generale**

L'applicazione **"Asite demo TMB"** è un'interfaccia frontend sviluppata in **React.js** con gestione dinamica dei dati tramite **WebSocket**.

Visualizza una serie di contenitori (immagini) interattivi sovrapposti, ciascuno con controlli sensoriali (termometri, interruttori, indicatori di rotazione), che riflettono in tempo reale i dati ricevuti dal server.

**2. Architettura dei componenti**

src/

├- App.js # Entry point e router principale

├- controlli/ # Componenti UI come termometri, interruttori, sensori di rotazione etc.

│ ├ termometro.js

│ ├ termometro2.js

│ ├ temp1.js

│ └─ rotazione.js

├- pagine/

│ └─ dettaglio.js # Pagina di dettaglio per contenitori

├- contesto/

│ └── contestoAllarme.js # Context API per gestire gli allarmi

├- immagini/ # Assets visuali dei contenitori

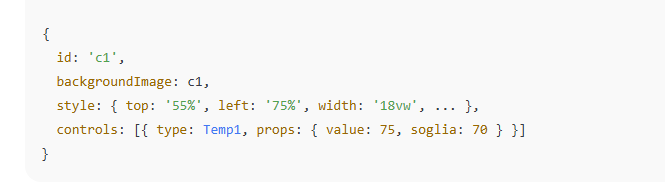
└── App.css # Stili globali

**3. Gestione dei contenitori**

Nel file App.js, i contenitori (containersData) sono definiti come array di oggetti con:

* id: identificativo univoco
* backgroundImage: immagine di sfondo
* style: stile CSS (posizionamento assoluto)
* controls: lista di componenti React dinamici (es. <Termo1 />, <Temp1 />)

Esempio:



Ogni contenitore è visualizzato tramite il componente <Contenitore />.

**4. Websocket e aggiornamento dei dati**

Questa versione dimostrativa dell’applicativo gestisce l’acquisizione dei dati (provenienti da una fonte dati esterna costituita da un’applicazione per smartphone realizzata in Flutter) tramite una connessione Websocket.

La useEffect() in App.js stabilisce una connessione Websocket a: ws://192.168.1.210:5000.

Quando riceve un messaggio nel formato:

*temp1; temp2; direction; state* {valore temperatura1, valore temperatura2, senso di rotazione [oraria/antioraria], stato dello switch [on/off] }

aggiorna dinamicamente i dati dei componenti tramite setContainers().

Nota: solo i contenitori presenti nella lista containersToUpdate, in questo esempio (c1,c4,c5,c8,c10) vengono aggiornati.

Esempio di aggiornamento:



Questo codice si trova all'interno di una **map,** usata per aggiornare dinamicamente i componenti presenti su uno "schermo" o "impianto", in seguito alla ricezione di un messaggio via WebSocket.

La sua funzione è quella di aggiornare la **temperatura visualizzata** in un componente di tipo Termo1 (un termometro) con il nuovo valore ricevuto dal WebSocket (temp1Value). In dettaglio:

 : controlla se il tipo del controllo corrente (ctrl) è Termo1. Questo per sapere quale tipo controllo si sta aggiornando.

 : crea un nuovo oggetto controllo, copiando tutte le sue proprietà originali (…ctrl…), ma con delle modifiche.

: aggiorna i props del controllo,

* *..ctrl.props*: copia tutti i props originali
* *value: temp1Value*: sovrascrive (o aggiunge) il prop *value* con il valore ricevuto dal Websocket.

In pratica: Se ad esempio ctrl fosse:

{

type: Termo1,

props: { width: 200, height: 200, value: 60, soglia: 70 }

} (prima dell’arrivo di temp1Value=75) , dopo l’esecuzione del codice illustrato si otterrà:

{

type: Termo1,

props: { width: 200, height: 200, value: 75, soglia: 70 }

}

Questo meccanismo reattivo permette di aggiornare lo stato dei controlli (termometri, indicatori, sensori…) senza modificare direttamente il DOM o i componenti già montati. React si occupa poi della renderizzazione del nuovo stato.

5. Documentazione del blocco useEffect

Questa funzione si occupa di

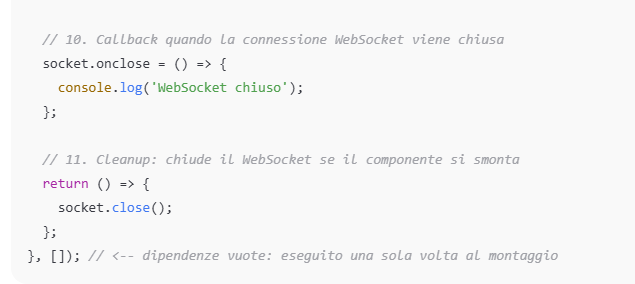
* stabilire una connessione con il Websocket
* ricevere dati in tempo reale (temperatura, direzione, stato…)
* aggiornare i controlli grafici (Termo1, termo2…) presenti sull’interfaccia.

**5. Documentazione del blocco useEffect: codice commentato**









6. Gestione allarmi (context API)

* *Il* ***Context*** *è una funzionalità offerta da React che permette di* ***condividere dati globali*** *(come uno stato, una funzione, o un oggetto) tra più componenti* ***senza doverli passare manualmente tramite props*** *a ogni livello della gerarchia.*

In questo progetto è stato definito un contesto denominato “**AlarmConte**xt” e lo si trova nel file separato *contestoAllarme.js*.



Si utilizza per condividere lo stato degli allarmi (alarms) e la funzione per aggiornarli (setAlarm) a qualsiasi componente che lo richieda.

* *Un* ***wrapper*** *è il componente padre che avvolge (wrap) altri componenti per fornirgli* ***il contesto****.*

Nel nostro caso : AlarmProvider è un wrapper che fornisce a tutti i componenti figli (inclusi Contenitore, DetailsPage…) l’accesso condiviso a AlarmContext.

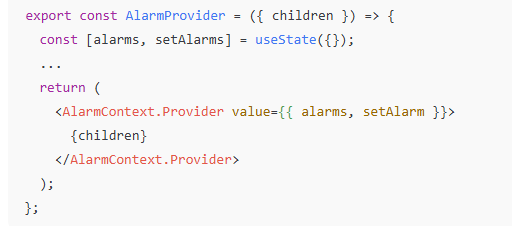
Nel codice:

**Funzionamento nella gestione degli allarmi**

1. Creazione del contesto :



1. Creazione del provider (il wrapper):



* Questo oggetto **value** sarà accessibile da qualsiasi componente figlio tramite **l’ hook** useAlarm().
* Gli allarmi sono salvati nello stato **{[id]: boolean}**

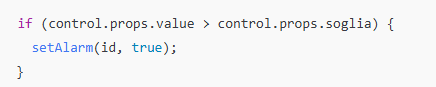
1. Utilizzo nei componenti figli :



Ora i componenti come detailsPage posso:

* **Leggere lo stato** di allarme corrente (alarms)
* **Aggiornare lo stato** (setAlarm (id, true/false))

Nella pagina di dettaglio (dettaglio.js), che si attiva con un click dal quadro principale, ogni controllo può aggiornare lo stato di allarme:



**7. Routing e pagine**

Si utilizza react-router-dom per il routing:

/ -> Homepage rappresenta il sinottico con griglia contenitori

/details:id -> Pagina di dettaglio contenitore.

In App.js



**8. Componenti grafici di controllo**

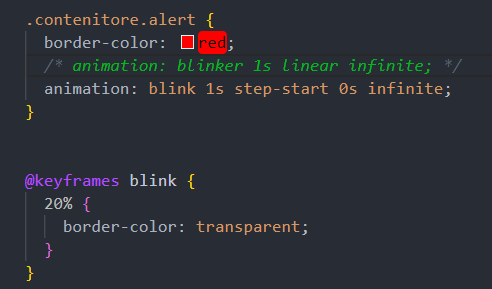
A scopo illustrativo si riporta il comportamento di **Termo2.js – Termometro avanzato**

* Props: *temparature, soglia*
* Calcola visualizzazione dinamica su <svg>
* Stato locale allarmato se *temperature>soglia*



* In condizione di allarme , tramite css, viene disegnato un bordo rosso sul componente e viene mostrato un messaggio.

Contenitore.css



Altri componenti simili, presenti nel progetto: (**cartella controlli)**

* *Termo1.js, temp1.js* : esempi di varianti di visualizzazione termica
* *RotationIndicator.js* : sensore di rotazione, con indicazione del senso di rotazione
* *Interruttore.js* : visualizza stato ON/OFF

In questa versione del sw , tutti i controlli ricevono valori da Websocket e possono attivare allarmi.

9. Componente principale: contenitore

Utilizzo:

Il componente Contenitore rappresenta **un blocco grafico interattivo del sinottico**, associato a uno specifico **id** e contenente una o più componenti di controllo (es. Termo1, switch1, temp2 ...).  
Gestisce **lo stato di allarme locale**, mostra visivamente un’**icona di allarme** e invia gli allarmi al **contesto globale**.

| **Prop** | **Tipo** | **Descrizione** |
| --- | --- | --- |
| id | string | Identificatore univoco del contenitore (usato anche per routing e allarmi) |
| backgroundImage | string | URL dell’immagine di sfondo del contenitore |
| controls | array | Array di componenti React (es. Termo1) che rappresentano controlli da monitorare |
| style | object | Stili inline personalizzati (facoltativo) |
| visible | boolean | Se false, il contenitore viene nascosto con la classe CSS invisible |

Stato interno

| **Stato** | **Tipo** | **Descrizione** |
| --- | --- | --- |
| isAlert | boolean | Indica se almeno uno dei controlli è in stato di allarme |
| controlRefs | useRef([]) | Contiene i riferimenti ai controlli per verificarne lo stato (isAllarmato) |

*Funzionalità principali*

1. **checkControls()**

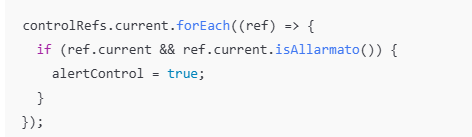
Funzione useCallback che controlla se uno dei controlli figli (del contenitore) ha attivato un allarme.



Se rileva un cambiamento, aggiorna lo stato locale (setIsAlert) e quello globale (setAlarm) tramite useAlarm().

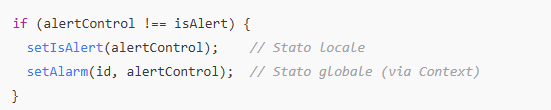
In dettaglio:

* Questa funzione serve a **scansionare tutti i controlli non visibili nel quadro principale** (es. Termo1, switch1.. ) per capire se almeno uno di loro è in stato di allarme.
* Viene usata come useCallback per evitare chiamate ridondanti.



* controlRefs è un array di ref, uno per ciascun controllo.
* ref.current punta al componente React montato.
* Ogni controllo deve avere un metodo .isAllarmato() che restituisce true o false.
* Se anche **un solo controllo** risulta allarmato → alertControl = true.

Cosa succede se cambia lo stato di allarme ?



Si aggiorna sia lo stato interno (isAlert) sia quello condiviso nel **contesto degli allarmi** (setAlarm), ma **solo se è cambiato** rispetto al valore precedente.

Esempio di flusso

1. Un controllo interno segnala un allarme.

2. checkControls() lo rileva.

3. Chiama:

* setIsAlert(true) → cambia l’aspetto del contenitore (es. classe alert).

E’ una funzione restituita da useState()



*Controlla solo* ***solo se questo contenitore specifico è in allarme****.*

*Viene usata per* ***modificare l’aspetto visivo*** *del contenitore (es. classe alert).*

*Serve per il rendering locale (es mostrare un badge !, cambiare colore…*

* setAlarm(id, true) → aggiorna il contesto condiviso. (Stato globale condiviso Context).
* E’ fornita dal contesto useAlarm()
* Aggiorna l’oggetto *alarms,* dove ogni id è un contenitore e il valore è true o false

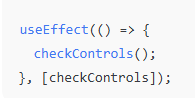


**Serve per la sincronizzazione globale**, utile per sapere **quali contenitori sono in allarme da qualsiasi parte dell’app**.

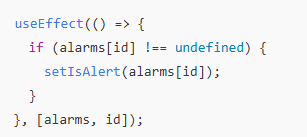
1. **useEffect : Controllo e sincronizzazione iniziale**

Nel codice ci sono **due hook useEffect** distinti:

*Primo useEffect; controllo iniziale*

* 
* Viene eseguito dopo il montaggio del componente
* Chiama checkControls() per rilevare eventuali allarmi iniziali nei controlli
* Può essere utile quando i controlli hanno già dei valori critici al primo render, quindi è necessario attivare un allarme.

*Secondo useEffect: sincronizzazione con il contesto*



* Ha la funzione di osservare il valore di alarms[id] dal contesto (useAlarm()).
* Se cambia il valore globale dell’allarme per questo contenitore aggiorna lo stato locale isAlert.

In sintesi:

* La funzione checkControls() : legge i controlli interni e decide se è presente un allarme, poi aggiorna lo stato locale e globale
* La funzione useEffect() : Al primo render controlla gli allarmi ->poi si sincronizza con il contesto (useAlarm).