

# ER Climate Monitoring Applicazioni e Servizi Web

Matteo Iorio - 0001112276 - matteo.iorio2@studio.unibo.it  
Fabio Vincenzi - 0001137554 - fabio.vincenzi2@studio.unibo.it  
Stefano Furi - 0001125057 - stefano.furi@studio.unibo.it

January 2025

# Indice

<b>1 Requisiti</b>	<b>4</b>
1.1 Introduzione . . . . .	4
1.1.1 Descrizione del Problema . . . . .	4
1.1.2 Obiettivi del Progetto . . . . .	4
1.1.3 Contesto e Stakeholders . . . . .	4
1.1.4 Tecnologie . . . . .	5
1.2 Requisiti non funzionali . . . . .	5
1.2.1 Requisito numero 1 . . . . .	5
1.2.2 Requisito numero 2 . . . . .	5
1.2.3 Requisito numero 3 . . . . .	6
1.2.4 Requisito numero 4 . . . . .	6
1.2.5 Requisito numero 5 . . . . .	7
1.3 Analisi degli Utenti: Personas e Scenari . . . . .	7
1.4 Personas . . . . .	7
1.5 Scenari d'Uso . . . . .	8
<b>2 Design</b>	<b>10</b>
2.1 Panoramica dell'architettura . . . . .	10
2.1.1 Authentication Service . . . . .	11
2.1.2 Notification Service . . . . .	11
2.1.3 Detection Service . . . . .	12
2.1.4 Sensor Registry Service . . . . .	13
2.1.5 API Gateway . . . . .	13
2.2 Tecnologie Utilizzate . . . . .	13
2.2.1 Tecnologie di Frontend . . . . .	13
2.2.2 Tecnologie di Backend . . . . .	14
<b>3 Progettazione dell'interfaccia utente</b>	<b>16</b>
3.1 Progettazione dell'Interfaccia utente . . . . .	16
3.1.1 Pagina Principale . . . . .	16
3.1.2 Login . . . . .	17
3.1.3 Registrazione . . . . .	18
3.1.4 Pagina Sensori . . . . .	19
3.1.5 Pagina Impostazioni Sensore . . . . .	20
3.1.6 Pagina delle notifiche . . . . .	21
3.2 Accessibilità e usabilità . . . . .	23
3.2.1 Accessibilità . . . . .	23
3.2.2 Usabilità . . . . .	24
<b>4 Implementazione</b>	<b>27</b>
4.1 Implementazione del Frontend . . . . .	27
4.1.1 Struttura del progetto . . . . .	27
4.1.2 Pagina Home . . . . .	27
4.1.3 Pagina Login . . . . .	29

4.1.4	Pagina Registrazione . . . . .	31
4.1.5	Notifiche . . . . .	32
4.1.6	Pagina Sensori . . . . .	35
4.1.7	Componente singolo della lista . . . . .	37
4.1.8	Pagina Impostazioni singolo sensore . . . . .	38
<b>5</b>	<b>Test</b>	<b>40</b>
5.1	Controllo qualità del codice . . . . .	40
5.2	Verifica UX . . . . .	42
<b>6</b>	<b>Deployment</b>	<b>44</b>
6.1	Backend . . . . .	44
6.2	Costruire e Lanciare i sensori . . . . .	44
6.3	Frontend . . . . .	45
<b>7</b>	<b>Conclusioni</b>	<b>45</b>
7.1	Sviluppi Futuri . . . . .	45

# 1 Requisiti

## 1.1 Introduzione

L'obiettivo di questa fase all'interno del progetto, permette di comprendere il contenuto ed i vari requisiti che fanno parte del nostro progetto. Questa fase è di vitale importanza, in quanto ci permette di capire quali sono le esigenze che il nostro sito andrà a soddisfare per fornire un'esperienza ottimale ai vari fruitori del nostro sito web.

### 1.1.1 Descrizione del Problema

Il problema che si è voluto affrontare riguarda una problematica che al giorno oggi è diventato un tema chiave per il futuro del nostro pianeta, il clima. Più in particolare il nostro progetto si concentrerà nella zona dell'Emilia Romagna. In questo modo i vari utenti saranno in grado di monitorare in tempo reale, la situazione climatica nel nostro territorio, potendo inoltre filtrare in base ai vari fenomeni atmosferici. Attualmente i nostri utenti, necessitano di una piattaforma in grado di fornirgli dati in tempo reale, garantendo anche una visualizzazione chiara ed efficace. Inoltre, alcuni dei nostri utenti, potrebbero avere bisogno di ricevere notifiche su alcuni topic riguardanti specifici eventi atmosferici. Assieme ai vari utilizzatori normali del nostro sito web, vi potrebbero essere anche degli admin, il cui compito invece è quello di interagire con i vari sensori installati, al fine di modificare alcuni dei loro parametri o in casi estremi spegnere il sensore stesso.

### 1.1.2 Obiettivi del Progetto

Il nostro progetto mira a:

- Migliorare la modalità di accesso ai dati riguardanti i vari fenomeni atmosferici;
- Implementare un sistema di notifiche riguardanti diversi fenomeni atmosferici;
- Fornire un accesso real time alla visualizzazione dei dati;
- Rendere il sito responsive per garantire l'accesso ai diversi dispositivi;

### 1.1.3 Contesto e Stakeholders

Il nostro sito web verrà principalmente utilizzato dai cittadini della regione Emilia Romagna. Oltre ai cittadini, questo sito web verrà utilizzato anche da coloro che dovranno transitare o permanere per un periodo generico di tempo sui territori dell'Emilia Romagna.

#### 1.1.4 Tecnologie

Per sviluppare tale sito web verranno utilizzate diverse tecnologie, tra cui:

- **Frontend:** Vue.js, Tailwind
- **Backend:** Node.js, Express, Socket.io
- **Database:** MongoDB, Mongoose

Il sito web che si andrà a sviluppare dovrà essere accessibile e conforme agli standard del WCAG.

## 1.2 Requisiti non funzionali

Il sito web che andremo a sviluppare dovrà soddisfare un'insieme di requisiti non funzionali. Qui di seguito verranno elencati tutti i requisiti non funzionali che il nostro progetto dovrà avere.

### 1.2.1 Requisito numero 1

Il primo requisito non funzionale che il nostro sito web dovrà rispettare, sarà la **velocità** con la quale andrà a fornire i dati ai vari utilizzatori del sito web. I tempi di risposta dovranno essere rapidi evitando tempi di attesa troppo lunghi.

**Quality Attribute Scenario :** Il nostro sito web si vedrà arrivare una richiesta per visualizzare i vari sensori presenti sul territorio Emiliano Romagnolo, riguardanti una specifica categoria di eventi atmosferici. La sorgente di tale evento è un utente generico che chiameremo utente A. Il nostro sistema, una volta accettata la richiesta, ritornerà all'utente l'elenco dei sensori installati, riguardanti la specifica categoria di fenomeno atmosferico che vuole visualizzare. I tempi di risposta da parte del nostro sistema dovranno essere inferiori ai 2 secondi. In questo modo si riuscirà a garantire al nostro utente A, una fruizione del sistema priva di attese lunghe. In questo modo l'utente potrà visualizzare e interagire con i vari sensori che verranno poi visualizzati sulla specifica cartina geografica.

### 1.2.2 Requisito numero 2

Il secondo requisito non funzionale che il nostro sistema dovrà avere, è **l'accessibilità**. Infatti si vuole garantire che il nostro sistema possa essere utilizzato da qualsiasi persona, indipendentemente dalle disabilità che quest'ultima possiede.

**Quality Attribute Scenario :** Dal momento in cui il nostro sito web, gestisce dati che possono tornare utili all'intera comunità, nessuno escluso è necessario garantire un accesso ai dati accessibile. Quando il nostro sistema si vedrà chiedere la collezione dei sensori riguardanti una determinata categoria di fenomeni atmosferici, da parte del nostro utente B, che presenta una cecità

totale. Il nostro sistema quando ritornerà i sensori, dovrà formattare i dati in maniera che questi siano perfettamente accessibili al nostro utente, così facendo anche il nostro utente B potrà interagire con i vari sensori e monitorare i dati ricevuti. In questo modo tutte le persone che presentano una qualsiasi tipo di disabilità potranno accedere ai nostri dati senza riscontrare alcun blocco.

### 1.2.3 Requisito numero 3

Il terzo requisito non funzionale che il nostro sistema deve avere riguarda la capacità di effettuare **modifiche** al sistema in maniera agile e veloce, senza andare a costruire un sistema troppo rigido. Così facendo i vari sviluppi futuri potranno essere eseguiti in maniera facile.

**Quality Attribute Scenario** : Siccome il servizio che andremo a sviluppare, come qualsiasi altro servizio esistente, dovrà prima o poi subire delle modifiche sarà necessario strutturare il sistema affinché quest'ultimo possa evolvere nel tempo. Si ipotizzi dunque che si voglia aggiungere una nuova API, che potrà poi essere utilizzata dai vari utenti del nostro sistema. Dunque l'ingegnere X avrà il compito di introdurre questa nuova funzionalità. In questo modo il nostro sito web potrà evolvere nel tempo. I tempi con la quale questa nuova funzionalità dovrà essere aggiunta non dovrà superare le 40 ore di lavoro. In questo modo potranno essere aggiunte nuove funzionalità nel corso del tempo, mantenendo i tempi di evoluzione bassi.

### 1.2.4 Requisito numero 4

Il quarto requisito che dovremmo rispettare riguarda la **sicurezza**. Più in particolare, si dovrà garantire l'accesso ad informazioni sensibili solamente a coloro che sono autorizzati a vedere ed interagire con tali dati.

**Quality Attribute Scenario** : Come descritto in precedenza vi sono degli utenti identificati con un ruolo di *admin*. Questo significa che questi utenti hanno più poteri rispetto ad un normalissimo utente. Il nostro sito web si vedrà arrivare una richiesta di login come admin, tale richiesta l'ha effettuata lo user X. Il sistema, una volta presa in carico la richiesta, controllerà se l'utente possiede o meno il ruolo di 'admin'. Nel caso in cui l'utente non sia autorizzato, il sistema lo informerà e lo reindirizzerà in una parte del nostro sito web che non presenta alcuna informazione sensibile. In caso contrario invece, all'utente verrà data la possibilità di accedere a delle zone con dati sensibili. Ma, ad ogni richiesta su un qualsiasi dato che presenti delle informazioni sensibili verrà sempre controllato che l'utente sia autorizzato ad accedere a tali dati. In questo modo riusciremo a mantenere un livello di sicurezza tale da isolare completamente qualsiasi tentativo di accesso ai dati sensibili.

### 1.2.5 Requisito numero 5

Il quinto ed ultimo requisito che il nostro sito web dovrà avere è l'**efficienza**. Più in particolare, bisognerà fare in modo che il sito web non consumi le capacità computazionali del dispositivo sul quale sta girando.

**Quality Attribute Scenario** : Dal momento in cui il nostro sito web è in grado di fornire una visualizzazione in tempo reale dei dati, bisognerà fare in modo che questo avvenga nella maniera più semplice ed efficace possibile. Il nostro sito web si vedrà arrivare una richiesta di visualizzazione in tempo reale dei dati riguardanti uno specifico sensore, tale richiesta è stata effettuata dall'utente Y. Una volta presa in carico tale richiesta, il nostro sito web dovrà attivare una logica attraverso la quale sarà possibile visualizzare in tempo reale i dati raccolti da uno specifico sensore. Tale logica non dovrà consumare più del 10 % della CPU. Così facendo, il nostro utente potrà godere di una navigazione fluida e priva di interruzioni, evitando di allocare troppa computazione sul proprio dispositivo, dal momento in cui vi potrebbero essere utenti che accedono al nostro sito web con dispositivi non all'avanguardia e non di ultima generazione.

## 1.3 Analisi degli Utenti: Personas e Scenari

La fase primaria che abbiamo condotto prima della vera realizzazione del sistema è stata quella di un'indagine basata su **Personas e Scenari d'uso**. Tramite l'utilizzo di questa strategia è possibile definire con maggiore chiarezza chi utilizzerà il nostro sito, quali sono i loro obiettivo e quali sfide potrebbero incontrare durante la navigazione nel sito web. Tutte le informazioni raccolte verranno utilizzate per guidare il design e lo sviluppo del sito stesso.

## 1.4 Personas

Le personas rappresentano gli utenti tipi del nostro sito web, basate su ricerche ed ipotesi.

**Personas 1:** La prima *Personas* che andremo a descrivere è Stefano Vincenzi, la cui età è 29 anni, come professione svolge l'attività di studente universitario. Tale utente ha una esperienza digitale molto alta ed utilizza con molta frequenza siti web e app, il suo obiettivo è quello di verificare con esattezza le condizioni meteo riguardante il suo territorio natale, ovvero l'Emilia Romagna. Le sue frustrazioni principali sono quelle di non trovare un sito web in grado di fornire informazioni dettagliate per i vari eventi atmosferici. Come dispositivo principale utilizza un computer.

**Personas 2:** La seconda persona che è stata individuata si chiama Ester Alfonso, l'età di tale signora è di 39 anni, come professione svolge l'attività di igienizzazione e pulizia di strutture ospedaliere. Il suo obiettivo è quello di monitorare la situazione meteo nella zona dell'Emilia Romagna, dal momento in

cui, tale zona è spesso afflitta da allagamenti ed esondazioni, in modo da andare e tornare da lavoro con estrema precauzione. Le sue frustrazioni principali sono quelle di non riuscire a trovare un sito web in grado di mantenerla sempre aggiornata con le ultime notifiche riguardanti gli eventi atmosferici. Il dispositivo principale che tale signora utilizza è lo smartphone.

**Personas 3:** La terza ed ultima *Personas* che andremo ad esplorare si chiama Mario Le Pinze. La cui età è di 27 anni. Come professione svolte l'attività di monitoraggio di sensori per fenomeni atmosferici all'interno del suolo Emiliano Romagnolo. Tale utente è ammesso ad interagire con i vari sensori installati su tale suolo, con la conseguenza di poter modificare i vari sensori. Le sue frustrazioni sono quelle di non riuscire ad avere un sito web in grado di fornirgli un'interfaccia semplice all'interno della quale egli riesce a modificare i vari sensori installati, poterli cercare tramite appositi filtri. Il dispositivo principale che tale utente utilizza è il computer.

## 1.5 Scenari d'Uso

Gli scenari d'uso ci permettono di illustrare le situazioni tipiche attraverso la quale i vari utenti interagiscono con il nostro sito web.

**Scenario 1: Monitoraggio in tempo reale della situazione** Le *personas* coinvolte in tale scenario d'uso sono sia Stefano Vincenzi e sia Ester Alfano. La situazione in cui queste due *personas* si trovano è molto generica e può coinvolgere molte più *personas* di quelle elencate precedentemente. Lo scenario di cui parliamo è rappresentato dalla preparazione per un itinerario per una visita/viaggio in direzione di una località all'interno della Emilia Romagna. Le nostre *personas* accedono al nostro sito web, e tramite le varie funzionalità possono monitorare ad esempio le precipitazioni in determinate zone di loro passaggio, la temperatura e così via. Per soddisfare questo bisogno, bisognerà fornire agli utenti una modalità di accesso ai dati semplice ed efficace.

**Scenario 2: Ricezione delle notifiche** La *personas* coinvolta per questo scenario è Ester Alfano. La situazione di Ester Alfano è la seguente, lei ogni mattina deve recarsi a lavoro, ma come purtroppo sta accadendo sempre più spesso negli ultimi anni, il territorio Emiliano Romagnolo, è duramente colpito da nubifragi violenti, che sfociano poi in esondazioni ed allagamenti. Ester nel tragitto casa lavoro spesso affianca dei fiumi, per evitare di rimanere incastrata nel traffico a causa di fenomeni atmosferici violenti, o ancora meglio per evitare possibili allagamenti dovuti ai fiumi che lei costeggia, necessita di ricevere notifiche in tempo reale (real time) riguardanti determinati topic su specifici fenomeni atmosferici, come ad esempio: 'se cadono più di X mm di acqua'. Così facendo lei potrà monitorare il suo percorso deviandolo nel caso in cui questo potrebbe essere interdetto. Per andare in contro alle esigenze di Ester, sarà necessario

fornire un sistema di notifiche real time, in grado di inviare anche notifiche push (come email) all'utente sottoscritto alla notifica.

**Scenario 3: Manutenzione dei sensori da parte degli Admin** Il terzo ed ultimo scenario di utilizzo riguarda solamente una cerchia ristretta di utenti nel nostro dominio, i così detti **Admin**. Tali utenti, come ad esempio la nostra *personas* Mario Le Pinze, sono in grado di interagire con i vari sensori installati, ad esempio modificano alcuni parametri riguardanti il campionamento dei dati dei vari sensori, l'orario in cui campionare, i giorni in cui effettuare le rilevazioni per poi finire nell'ultima operazione, ovvero quella di spegnimento di uno specifico sensore. In questo scenario, oltre a fornire le funzionalità sopra elencate, sarà fondamentale fornire anche un meccanismo di ricerca efficace per i vari sensori, in modo da filtrare i vari risultati di ricerca.

## 2 Design

### 2.1 Panoramica dell'architettura

Il sistema realizzato si basa sull'architettura di tipo *client-server*, il cui **Frontend** comunica con un **Backend** centralizzato per interagire con il sito web stesso. Il Backend a sua volta si può scomporre in parti più piccole, dal momento in cui è stata adottata una modalità di sviluppo via microservizi. Ognuno dei vari microservizi rappresenta un sotto dominio dell'intero sistema, dove ognuno dei quali definisce il proprio *bounded context*. Oltre ai vari microservizi sviluppati, all'interno del Backend è presente un componente di transito principale, l'**API Gateway**. Tutte le richieste che partono dal Frontend arrivano all'**API Gateway**, il quale poi avrà il compito di instradare la richiesta del client ad un servizio specifico. I vari microservizi individuati sono i seguenti:

- **Authentication Service**
- **Notification Service**
- **Detection Service**
- **Sensor Registry Service**

Per funzionare, ognuno dei seguenti microservizi individuati necessita di un database. Per soddisfare questa necessità si è deciso di utilizzare un database NoSQL come MongoDB. Infine per permettere la comunicazione tra i vari servizi, si è deciso di utilizzare le REST API, utilizzando JSON come tipologia di dato scambiato tra le varie richieste. Si è deciso di sviluppare tale sistema tramite i microservizi, in modo da aumentare la sua scalabilità e la sua facilità di manutenzione.

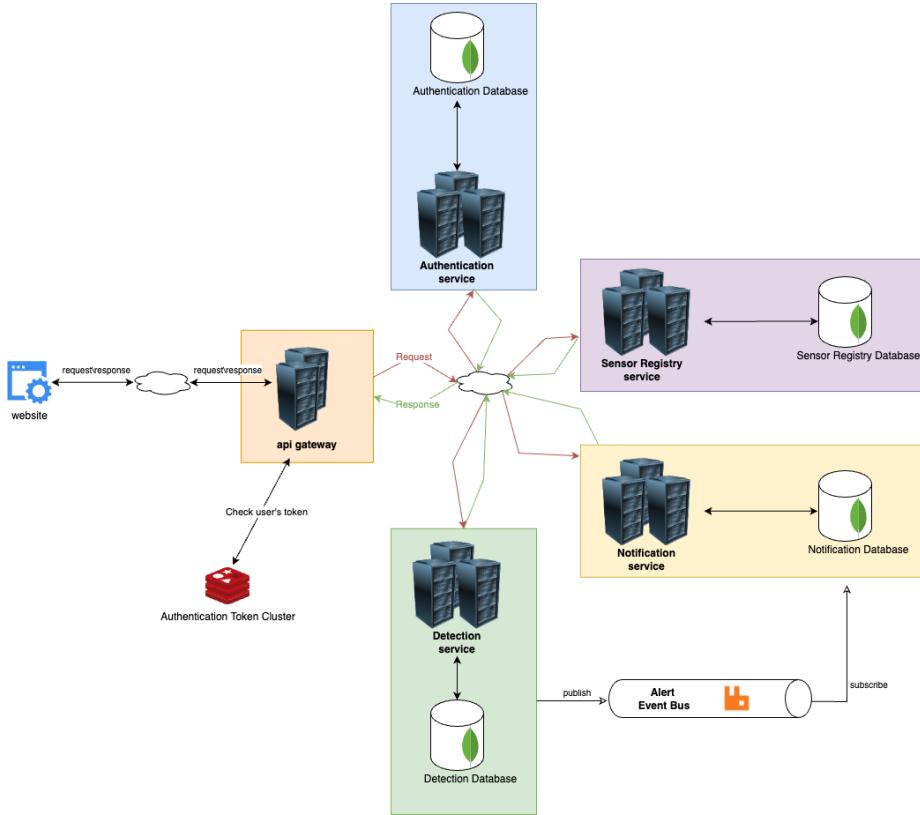


Figura 1: Architecture Overview

### 2.1.1 Authentication Service

Il primo microservizio che andremo a descrivere è l'authentication service. Il bounded context che ricopre riguarda la gestione dei vari utenti registrati, non registrati e la generazione di token per verificare se l'utente è o meno autenticato (la funzione di autenticazione, si intersecherà poi con l'API Gateway). Come descritto in precedenza, tale servizio dispone del proprio database, all'interno del quale vengono memorizzati tutti gli utenti appartenenti al dominio. Si possono distinguere due categorie differenti di utenti, gli utenti normali e gli admin. Tale differenza si ripercuote poi sulle varie funzionalità che il sistema mette a disposizione.

### 2.1.2 Notification Service

Il sistema di notifica è composto principalmente da due componenti:

- Un **MessageBroker**, in grado di gestire l'arrivo di eventi all'interno di un event bus spediti dal Detection Service (2.1.3).

- Un **SocketManager**, in grado di stabilire e gestire le connessioni persistenti con gli utenti che richiedono la sottoscrizione ad un determinato evento, e il perciò il relativo *routing* in base ad esso.

Per quanto riguarda gli *eventi*, essi sono scaturiti dai sensori stessi e propagati tramite il servizio delle misurazioni, il quale a sua volta pubblica gli eventi tramite un event bus, in modo tale da mantenere un tipo di comunicazione asincrona con una QoS di tipo *best effort* (essendo il contenuto delle notifiche non di importanza critica). Il servizio di notifiche estrapola dalla *Detection* informazioni fondamentali per la costruzione della notifica che verrà spedita al client, in particolare viene estratto il *tipo* dell'evento, il *nome del sensore* e la *query* riguardante l'evento: quest'ultima può rappresentare un valore di soglia, come per esempio il superamento della capienza di un fiume del 75%, oppure un semplice valore nominale come *severe wind gusts*. In questo modo si effettua il routing delle notifiche in base al **topic** su cui gli utenti si iscrivono. In particolare, il topic è della forma **notifications.<type>.<sensor-name?>.<query?>**, dove il nome del sensore e della query possono essere sostituiti da wildcard (# oppure \*), in modo tale da poter fornire meccanismi a grana grossa e via via sempre più fine riguardo il tipo di notifiche che un utente desidera ricevere. Riguardo questi ultimi, le connessioni risultano persistenti per avere un comportamento *real-time* senza la necessità di meccanismi di polling al server. Infine, il servizio dispone di un database documentale contente due tipologie di documenti: il primo contiene informazioni riguardanti le sottoscrizioni degli utenti, mentre il secondo mantiene gli eventi serializzati spediti attraverso l'event bus dal detection service. Concludendo, è possibile configurare dinamicamente il comportamento del **notification-service** alla ricezione di un nuovo evento, tramite l'aggiunta di una **notificationCallback**, ossia l'azione che viene eseguita all'arrivo di ogni notifica dall'event bus. In questo modo è infatti possibile definire meccanismi più articolari per la spedizione di notifiche, come l'invio di e-mail, l'emissione di un evento tramite il canale WebSocket, oppure la segnalazione di ulteriori servizi.

### 2.1.3 Detection Service

Il Detection Service ha il compito di gestire e rendere disponibili le detection provenienti dai vari sensori presenti nel sistema. Questo servizio offre un'interfaccia per la memorizzazione e il recupero delle detection e una connessione in tempo reale tramite socket per la ricezione immediata delle detection generate dai sensori.

In particolare il detection service mette a disposizione una serie di route per interagire con il sistema, ognuna delle quali permette di effettuare diverse operazioni sui dati riguardanti le detection.

Il servizio consente di effettuare il salvataggio delle varie detection prodotte dai sensori attivi, permetterà inoltre di ottenere la posizione di tutti i sensori disponibili filtrandoli per tipologia di sensore ed infine sarà possibile ottenere uno storico di detection di uno specifico sensore.

Il detection service offre una comunicazione "real-time" attraverso l'utilizzo di una connessione socket per permettere l'invio delle detection ricevute dal servizio ai client che si sottoscrivono alla ricezione di aggiornamenti per un singolo sensore, questa comunicazione risulterà utile per l'aggiornamento realtime dei grafici che mostreranno i valori delle detection di un singolo sensore.

#### 2.1.4 Sensor Registry Service

Come descritto in precedenza ogni microservizio fa riferimento ad uno specifico bounded context, in particolare il Sensor Registry Service, permette di interagire con i vari sensori che sono installati sul territorio. Questo microservizio in particolare, permette funge da registro principale, all'interno del quale sono mantenuti tutti i riferimenti dei vari sensori, tra cui il loro indirizzo IP e la loro Porta. In questo modo nel caso in cui un admin, avesse la necessità di interrompere un sensore, egli si avvalerebbe di tale servizio. Anche in questo caso, tutti i vari sensori sono memorizzati all'interno di un database che fa riferimento solo ed esclusivamente a questo servizio.

#### 2.1.5 API Gateway

Infine l'ultimo servizio realizzato è l'API Gateway, il cui compito è quello di raccogliere in input tutte le richieste del client e reindirizzarle al giusto servizio. Successivamente, una volta ricevuta la risposta dal servizio interrogato, egli ritnerà la risposta al client. Un'altra delle funzionalità principali di questo sistema, è il servizio di autenticazione. Per fare ciò, questo servizio si avvale dell'Authentication Service, il quale una volta ritornato un token, l'API Gateway lo memorizzerà all'interno di un repository, in modo da poter effettuare delle ricerche ottimizzate per validare i permessi e le responsabilità dell'utente che ha effettuato la richiesta.

## 2.2 Tecnologie Utilizzate

Per lo sviluppo di questo progetto sono state utilizzate diverse tecnologie, qui di seguito verranno separate in tecnologie di Frontend e di Backend.

### 2.2.1 Tecnologie di Frontend

Le prime tecnologie che andremo ad elencare riguardano tutto lo sviluppo dell'interfaccia utente. Come prima cosa siamo partiti dallo scegliere il linguaggio di programmazione, in questo caso ci siamo appoggiati a **TypeScript**, in modo da ottenere i vantaggi di **Javascript** e la sicurezza di definire i tipi delle variabili grazie a **TypeScript**. Una volta scelto il linguaggio di programmazione, ci siamo spostati nella selezione del framework di sviluppo, tra le varie possibilità abbiamo scelto di usare **Vue.js**, il quale tramite **Composition API** e la facilità nel supporto di **TypeScript** ci ha permesso di sviluppare interfacce in maniera semplice e veloce. Assieme a tale framework, sono state utilizzate diverse librerie per permettere l'interazione tra client e server come la libreria **Axios** e la

libreria `Socket.io` per permettere la comunicazione tramite `socket`. Successivamente, come framework CSS, abbiamo scelto `Tailwind`, dal momento in cui permette di essere adottato in `Vue.js` in maniera intuitiva. Una volta definite tutte queste dipendenze, abbiamo inoltre introdotto un’insieme di funzionalità che ci permettessero di mantenere un’elevata qualità del codice. Ad esempio, come descritto nei requisiti non funzionali, una delle qualità principali che le nostre interfacce devono rispettare riguardano l’accessibilità, per verificare che le pagine del nostro sito web siano sempre accessibili, abbiamo utilizzato la libreria `eslit-plugin-vuejs-accessibility` la quale combinata con il `linter` ci ha permesso di correggere e rendere le pagine accessibili. Oltre a ciò, per permettere che le definizioni dei componenti siano privi di errori, abbiamo utilizzato la libreria `eslit`.

### 2.2.2 Tecnologie di Backend

Per quanto riguarda tutto il lato backend, si è scelto di utilizzare `NodeJS` con l’impiego della libreria `Express`. Tutto il codice è stato scritto utilizzando `TypeScript`, in modo da mantenere la type safety e la facilità di estensibilità del codice. In alcuni dei microservizi implementati viene utilizzata la libreria `Socket.io` per creare e mantenere comunicazioni attraverso il protocollo `WebSocket`. Come riportato in precedenza, ognuno dei microservizi dispone del proprio database `NoSQL Mongodb`, utilizzato assieme alla libreria `Mongoose` per la definizione e l’interazione con i vari documenti. Ed infine ognuno dei vari microservizi è stato testato tramite l’impiego di librerie apposite per la creazione dei test, tra cui: `Mocha`, `Supertest` e `Jest`. Ognuno dei test utilizza un database apposito in modo da non interferire con i database di produzione. Di seguito poi verranno esplicitate alcune tecnologie che fanno riferimento solamente ad alcuni dei servizi implementati.

**Authentication Service** Il servizio di autenticazione, come descritto in precedenza si occupa anche della creazione e validazione di Token. Per effettuare questo task, viene utilizzata la libreria `jwt` e `jwt-decode`.

**API Gateway** All’interno dell’API Gateway, viene utilizzato `Redis` come repository all’interno della quale memorizzare i token ricevuti dall’Authentication Service. Inoltre per instradare le richieste del client verso gli altri servizi, viene utilizzata la libreria `Axios`.

**Notification Service** Il servizio di notifiche, per far sì che venga mantenuto un tipo di comunicazione asincrona verso il produttore di eventi (detection service), utilizza un broker `RabbitMQ` per la ricezione di messaggi da quest’ultimo. La loro comunicazione avviene attraverso un unico *exchange*, mentre il routing dei vari eventi avviene attraverso l’apposita *routing key*, composta similmente al *topic*, ossia, `<type>. <sensor-name>. <query>`, con la differenza che tutti i parametri sono obbligatori. In questo modo è possibile creare un numero di code dinamico in base al tipo di sensori, al loro nome o alle query che espongono,

poiché le code vengono generate in base alla *routing key* specificata nell'evento. Grazie a questo meccanismo, l'intero servizio di notifiche risulta agnostico rispetto alle diversità dei sensori, con la limitazione che la *routing key* sia sempre della forma richiesta.

### 3 Progettazione dell'interfaccia utente

#### 3.1 Progettazione dell'Interfaccia utente

Una delle componenti principali che l'utente finale andrà ad utilizzare sono le interfacce. Dunque, questa considerazione ci ha portato a realizzare dei mockup delle varie interfacce.

##### 3.1.1 Pagina Principale

Come si può notare dal mockup, la pagina principale è composta da tre sezioni principali:

- **Navbar:** contenente il titolo dell'applicazione e i link di navigazione, da notare che se un utente è un admin potrà accedere anche alla sezione sensors come si può notare dai due mockup, inoltre presenta le icone per accedere alle sezioni notifiche e alle informazioni del profilo
- **Mappa:** Visualizza la posizione di tutti i sensori disponibili per la tipologia di sensore selezionata
- **Selettore dei sensori:** Permette di visualizzare sulla mappa solo le posizioni dei sensori appartenenti alla categoria selezionata

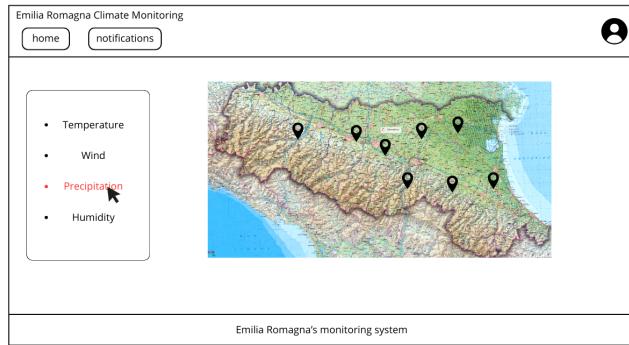


Figura 2: Mockup Pagina principale

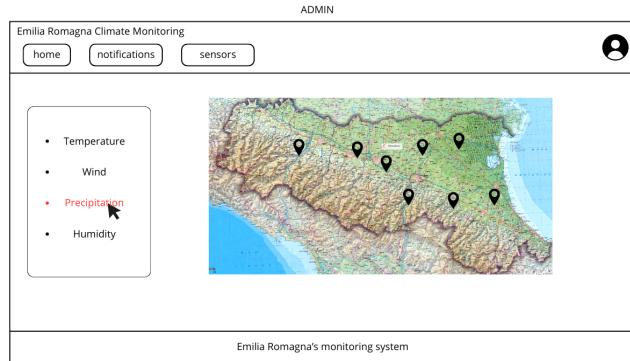


Figura 3: Mockup Pagina principale (admin)

### 3.1.2 Login

Come descritto all'interno dell'analisi del dominio, ogni utente ha la possibilità di loggare all'interno del sistema. Per permettere tale azione, è stato necessario realizzare la pagina di Login.

**Come raggiungere la pagina** La pagina che riguarda il login può essere raggiunta tramite click sul logo dell'utente che si trova in alto a destra della barra di navigazione., oppure tramite pulsante login che si trova all'interno del form di registrazione di un nuovo utente 6.

Figura 4: Mockup Pagina del Login

Come si può notare dal mockup 4, è presente una `check box` 'login as admin', la quale una volta cliccata permetterà di visualizzare un altro componente del form dove all'interno del quale sarà necessario inserire il *secret key*. Così facendo, la pagina che verrà visualizzata permetterà di effettuare il login ad uno user di tipo admin come riportato in figura 5

The mockup shows a 'Login' form with the following fields:

- Email: An input field.
- Password: An input field.
- Login as admin: A checked checkbox labeled 'Login as admin'.
- Secret Key: An input field.
- Login: A teal-colored button labeled 'Login'.
- Register: A grey button labeled 'Register'.

Figura 5: Mockup Pagina del Login Admin

In questo modo siamo in grado di differenziare e fornire una modalità di accesso differente in base al tipo di user che sta provando ad accedere.

### 3.1.3 Registrazione

Siccome all'interno del nostro sistema è possibile loggarsi, dovrà anche essere necessario poter registrare un nuovo utente. Per fare ciò, è stato necessario realizzare la pagina per registrare un nuovo utente.

**Come raggiungere la pagina** La pagina di registrazione di un nuovo utente può essere raggiunta tramite click dalla barra di navigazione o tramite click sul bottone 'register' nell'interfaccia di login 4.

A wireframe mockup of a registration form. At the top center is the word "Register". Below it are two input fields: one for "Email" and one for "Password", each with a small placeholder text inside. At the bottom of the form are two buttons: a teal-colored button labeled "Register" and a grey button labeled "Register" directly below it.

Figura 6: Mockup Pagina di Registrazione

Come si può notare dal mockup di tale pagina (6, è possibile registrare utenti 'normali' e non admin, dal momento in cui quest'ultimi potranno essere aggiunti da *root* users del sistema tramite query separate ed esterne.

### 3.1.4 Pagina Sensori

Uno dei vantaggi che un utente di tipo admin possiede, è quello di poter visualizzare in tempo reale la lista di tutti i sensori che attualmente sono impiegati per la raccolta di dati in tempo reale. All'interno di questa pagina, l'admin potrà inoltre interagire con i vari sensori, in modo da poterne cambiare le impostazioni.

**Come raggiungere la pagina** Per raggiungere tale pagina, bisognerà come prima cosa effettuare il login come admin, successivamente, una volta verificata l'identità dell'utente ed aver visualizzato i suoi permessi, sarà possibile accedere a tale pagina utilizzando la barra di navigazione. All'interno di questa pagina, come si può vedere dalla figura 7, verranno visualizzati tutti i sensori presenti sul territorio, con alcune delle loro informazioni principali:

- *Name*: nome del sensore;
- *IP*: Indirizzo Ip del sensore;
- *Port*: Porta sulla quale ascolta il sensore;

Per permettere inoltre una ricerca ottimizzata dei sensori, abbiamo pensato di introdurre anche una sezione dedita alla ricerca, così facendo i vari sensori verranno filtrati in base a ciò che si sta cercando. Dal momento in cui i vari sensori presentano tre diverse proprietà, come elencato in precedenza, abbiamo pensato

di inserire tre radio button in modo da filtrare la ricerca in base al tipo di attributo. Oltre a ciò sotto ognuno dei sensori saranno anche presenti due pulsanti, il primo, 'shut down', avrà la funzione di spegnere il sensore di riferimento, mentre il secondo, 'settings', aprirà un nuovo componente che permetterà all'admin di interagire con il sensore.

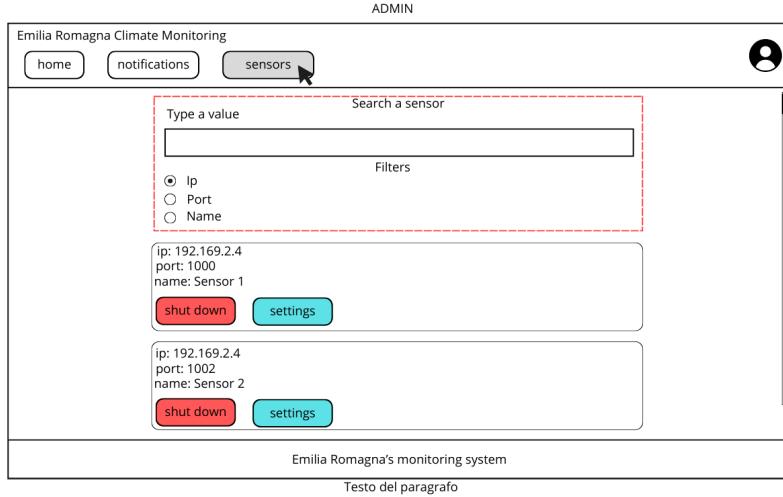


Figura 7: Mockup Pagina dei sensori

### 3.1.5 Pagina Impostazioni Sensore

Una volta che l'utente admin visualizzerà i vari sensori, potrà avere la necessità di dover modificare le impostazioni di alcuni di essi, per permettere ci abbiamo pensato alla pagina per effettuare delle modifiche sui sensori esistenti, come indicato in figura 8.

**Come raggiungere la pagina** Per raggiungere tale pagina, prima di tutto bisognerà effettuare il login come admin e posizionarsi all'interno della pagina che permette di visualizzare tutti i sensori, figura 7. Una volta che l'admin sarà su tale pagina, cliccherà sul bottone 'settings', il quale aprirà la finestra delle impostazioni. All'interno di questo componente sarà possibile eseguire del modifiche che riguarderanno il sensore, come ad esempio la modifica del nome del sensore, la modifica dei giorni in cui effettuare le rilevazioni ed infine l'ora in cui effettuare la rilevazione.

ADMIN

**Sensor Settings**

Change sensor name

Name:

**change name**

Cronjob days [format: dd-dd]

Days:

**change days**

Cronjob time

Hours:  Minutes:

**change time**

Figura 8: Mockup Pagina Impostazioni del Sensore

### 3.1.6 Pagina delle notifiche

Il sistema permette all’utente di sottoscriversi ad un sistema di notifiche in tempo reale. Prima di tutto, viene definito il concetto di *topic* su cui è possibile sottoscriversi: un *topic* è una tripla della forma `<type>. <sensor-name?>. <query?>`, dove rispettivamente si hanno:

- **type**: il tipo dell’evento generato dai sensori (e.g. *river-level*, *rain*, *wind*, *temperature*, ...). Questa informazione è obbligatoria per sottoporre una sottoscrizione;
- **sensor-name**: il nome del sensore da cui ricevere notifiche, del tipo specificato nel campo *type*;
- **query**: il nome dell’evento che un determinato tipo di sensori espongono, come per esempio un valore di soglia in percentuale, un valore nominale indicante un evento anormale (e.g. *light/moderate/severe gusts* per anemometri). Generalmente, una *query* fa riferimento ad un valore di soglia, ed un nome identificativo della stessa.

Questi ultimi due campi sono opzionali, e permettono di effettuare sottoscrizioni più a grana fine. È possibile quindi specificare solo un tipo, un tipo ed un sensore, un tipo ed una query oppure un tipo, un sensore ed una query (forma completa).

All’interno dell’interfaccia utente deve quindi poter essere possibile sottoscriversi a una tipologia di eventi di notifiche, permettendo solo sottoscrizioni valide, vale a dire composte da almeno un *type*, ed optionalmente riferite ad un sensore e/o una query. Allo stesso tempo, deve essere possibile visualizzare un

elenco di tutte le sottoscrizioni attive e permettere all'utente di disiscriversi da esse.

In figura 9 è mostrato un *mockup* di come si è pensato di realizzare il pannello di visualizzazione e gestione delle notifiche. Essendo un componente visitabile da ogni pagina, sarebbe opportuno rappresentarlo tramite un modale accessibile mediante un icona all'interno dell'header. D'altra parte, per motivi di semplicità, è stato pensato di includere tutte le funzionalità all'interno del medesimo modale, in modo tale da non necessitare di componenti separati, uno per la visualizzazione delle notifiche (presumibilmente il modale) e l'altro per la gestione delle sottoscrizioni.

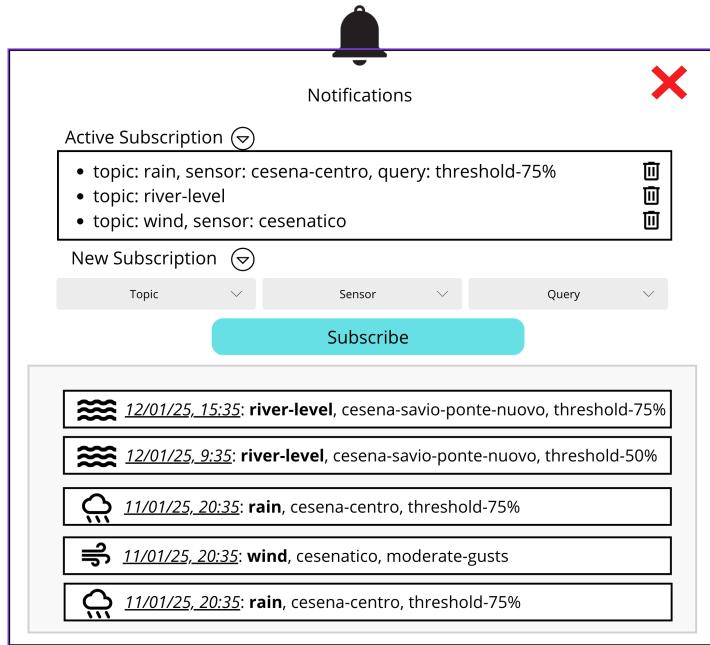


Figura 9: Modale del pannello di visualizzazione e gestione notifiche

Per rendere l'interfaccia più intuitiva e meno confusionaria possibile, è stato pensato di nascondere l'elenco delle sottoscrizioni attive e il *form* per la creazione di una nuova sottoscrizione. È quindi possibile accedervi clickando al di sopra della scritta o del logo per il dropdown. In figura 9 è mostrata la versione completamente espansa, al cui interno possiamo distinguere tre sotto componenti:

- Il primo componente è il pannello di gestione delle sottoscrizioni attive, dove è possibile visualizzarle e, tramite l'icona dedicata, disiscriversi da una di queste;
- Il secondo componente è il *form* per la creazione di una nuova sottoscrizione. Tramite l'impiego di un dropdown menu, è possibile far selezionare

all’utente i soli campi che risultano validi (i.e. i tipi/sensori/query esistono all’interno del sistema).

- Il terzo ed ultimo componente è l’elenco delle notifiche ricevute. Questo elenco viene aggiornato in tempo reale, e ogni notifica è caratterizzata da un logo esemplificativo per il tipo di evento, data e ora, tipo, sensore e nome della query dell’evento.

In quest’ultimo elenco è possibile clickare su una notifica per espandere il suo contenuto all’interno di un ulteriore modale come mostrato in figura 10.

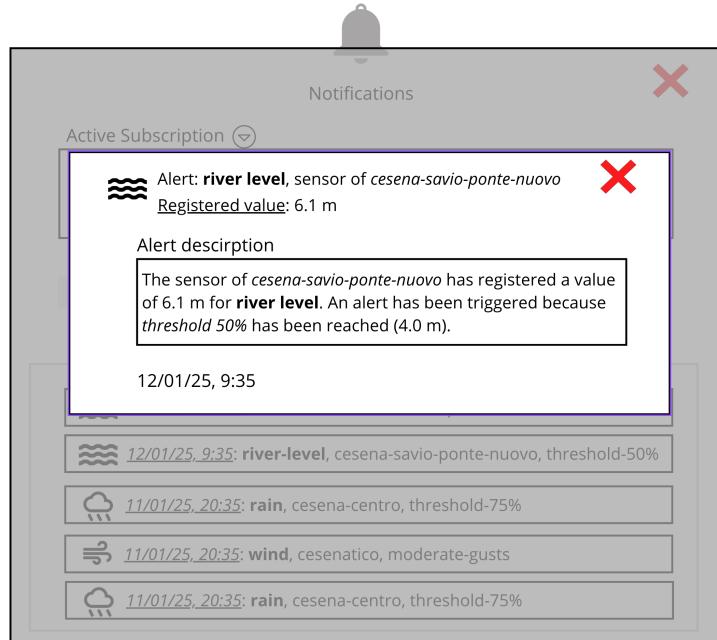


Figura 10: Modale dell’espansione di una notifica dall’elenco notifiche

In questo pannello sono contenute informazioni più dettagliate riguardanti la notifica, come per esempio il valore di soglia che ha scaturito l’evento e il valore attualmente registrato dal sensore.

## 3.2 Accessibilità e usabilità

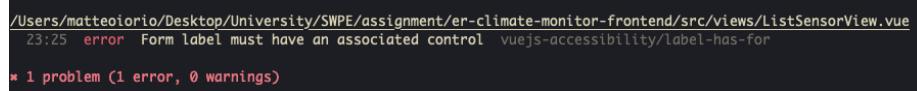
### 3.2.1 Accessibilità

Nel contesto del web, l’accessibilità è uno degli elementi fondamentali per garantire un’esperienza utente inclusiva ed intuitiva. Un sito web accessibile, permette a tutti gli utenti, indipendentemente dalle loro capacità fisiche o cognitive, di fruire dei contenuti e dei servizi offerti. Secondo le linee guida del **WCAG 2.1**

(Web Content Accessibility Guidelines), un sito web per essere accessibile deve rispettare quattro principi fondamentali:

- **Percepibile:** I contenuti devono essere disponibili in forme alternative per utenti con disabilità visive o uditive, come ad esempio ogni immagine deve avere un testo alternativo;
- **Utilizzabile:** L’interfaccia e la navigazione devono essere facilmente fruibili, garantendo l’uso della tastiera come alternativa al mouse;
- **Comprensibile:** I contenuti devono essere chiari e leggibili;
- **Robusto:** Il sito deve essere compatibile con diverse tecnologie assistive.

Dal momento in cui garantire l’accessibilità del nostro sito web è una delle nostre missioni principali, abbiamo preso seriamente questo problema. Come prima cosa, dal momento in cui le nostre interfacce utenti vengono realizzate a run-time, può risultare veramente molto complicato testare la pagina nella sua interezza. Siccome uno dei vantaggi di `Vue.js` è la possibilità di utilizzare librerie aggiuntive, abbiamo scelto di utilizzare la libreria `eslint-plugin-vuejs-accessibility`. Questa libreria viene utilizzata come pacchetto aggiuntivo all’interno dell’`eslint`, ovvero il software il cui compito è quello di monitorare i `<template>`, `<script>` e i file `*.vue` per possibili errori di sintassi, incorretto uso di direttive e violazioni nella scrittura di codice `Vue`. Grazie al nuovo pacchetto aggiunto, siamo in grado, in ogni istante della fase di sviluppo, di monitorare se all’interno del codice, in particolare nei tag `template`, vi siano possibili violazioni di accessibilità, come mostrato nella figura 25.



```
/Users/matteiorio/Desktop/University/SWPE/assignment/er-climate-monitor-frontend/src/views/ListSensorView.vue
23:25  error  Form label must have an associated control  vuejs-accessibility/label-has-for
* 1 problem (1 error, 0 warnings)
```

Figura 11: Errore nella violazione di accessibilità

In questo modo siamo in grado di mantenere un livello di accessibilità tale da permettere l’utilizzo del nostro sito web a qualsiasi tipo di utente, indipendentemente dalla loro condizione.

### 3.2.2 Usabilità

L’usabilità di un sito web, fa riferimento alla facilità con la quale i vari utenti possono navigare ed interagire con l’interfaccia utente. Inoltre, dal momento in cui è possibile che i vari utenti utilizzino dispositivi diversi per accedere al sito web, è stato necessario adottare una politica di sviluppo delle pagine di tipo *Desktop First*, dal momento in cui prevediamo che gli utenti utilizzino dei dispositivi con interfacce grandi abbastanza da visualizzare la cartina geografica in maniera chiara. Per permettere comunque che il sito web si adatti anche a

dispositivi con capacità di visualizzazione minori, come ad esempio smartphone, abbiamo dovuto adottare delle politiche di sviluppo che permettessero di adattare il nostro sito web a qualsiasi tipo di schermo. Per la realizzazione del nostro sito web, abbiamo scelto di utilizzare Tailwind CSS, un framework CSS utility-first che ci ha permesso di sviluppare un'interfaccia moderna e altamente personalizzabile. Grazie alle classi responsive di Tailwind (come `lg:`, `md:`, `sm:`), abbiamo strutturato il layout pensando prima all'esperienza su desktop, aggiungendo poi regole specifiche per rendere il sito completamente fruibile anche su smartphone e tablet. L'utilizzo di Tailwind CSS ci ha permesso di mantenere il codice pulito ed efficiente, migliorando le prestazioni del sito e garantendo una perfetta esperienza utente sia su desktop che su mobile. Inoltre, durante la creazione del nostro sito web abbiamo voluto seguire alcune regole fornite da **Jacob Nielsen**, tra cui:

- *Design ed estetica minimalista*
- *Riconoscimento anziché ricordo*
- *Consistenza e standard*
- *Corrispondenza tra sistema e mondo reale*

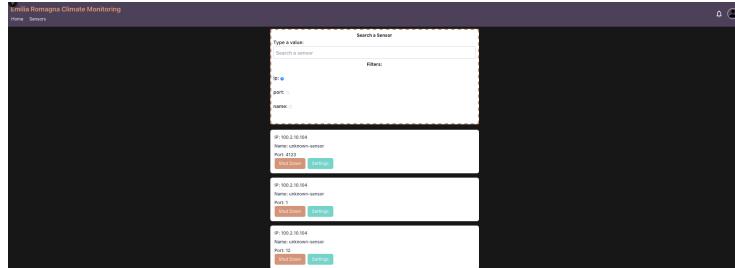


Figura 12: Pagina dei sensori visualizzata su Dekstop

The screenshot shows a web-based monitoring interface titled "Emilia Romagna Climate Monitoring". At the top, there are navigation links for "Home" and "Sensors", along with a user icon. Below the header, a search bar is labeled "Search a Sensor" with the placeholder "Search a sensor". A "Filters:" section contains three dropdown menus: "ip:" set to "○", "port:" set to "○", and "name:" set to "○". The main content area displays three sensor entries, each in a separate card:

- IP: 100.2.10.104  
Name: unknown-sensor  
Port: 4123  
Buttons: Shut Down (orange), Settings (green)
- IP: 100.2.10.104  
Name: unknown-sensor  
Port: 1  
Buttons: Shut Down (orange), Settings (green)
- IP: 100.2.10.104  
Name: unknown-sensor  
Port: 12  
Buttons: Shut Down (orange), Settings (green)

Figura 13: Pagina dei sensori visualizzata su un'interfaccia ridotta

## 4 Implementazione

All'interno di questo capitolo verranno descritte le pratiche di implementazione del sistema. Concentrandoci in particolare sui componenti del frontend e sulle interazioni tra interfacce utente e backend.

### 4.1 Implementazione del Frontend

Come descritto in precedenza, le interfacce utente sono state sviluppate utilizzando `Vue.js` in collaborazione con `Tailwind CSS`. Tutte le comunicazioni verso il backend avvengono tramite chiamate `REST`, utilizzando la libreria `Axios`.

#### 4.1.1 Struttura del progetto

L'inizializzazione del progetto è stato realizzato tramite l'utilizzo del comando `npm create vue@latest`, il quale ci ha permesso di configurare a priori l'intero progetto. Tramite questo comando, tra le varie impostazioni che si possono scegliere, abbiamo deciso di utilizzare `TypeScript` ed il supporto di `ESLINT`. Questo ci ha permesso di ottenere una base di appoggio per realizzare l'intero nostro progetto.

#### 4.1.2 Pagina Home

La pagina Home è il punto di accesso principale dell'interfaccia utente, essa offre un'interfaccia che consente all'utente di interagire con i sensori e visualizzare i dati in tempo reale.

#### Componenti principali

1. **NavBar:** attraverso la nav bar è possibile accedere alle differenti sezioni dell'applicativo, visualizzare le informazioni dell'utente loggato ed accedere all'area notifiche.
2. **MapComponent:** La mappa è gestita tramite `Leaflet.js` e mostra la posizione di tutti i sensori disponibili per il tipo selezionato. Ogni sensore è rappresentato da un marker, cliccando su di esso viene visualizzato un popup contenente un grafico delle rilevazioni recenti del sensor. Il grafico viene poi aggiornato in tempo reale tramite una connessione `WebSocket`
3. **SensorTypeSelect:** Questo componente permette di selezionare il tipo di sensore da una lista, al modificarsi della selezione il componente della mappa andrà ad aggiornarsi mostrando i sensori disponibili per il tipo selezionato dalla lista.
4. **Popup:** Questo componente viene mostrato al click su un marker all'interno della mappa il quale mostra le ultime detection su un grafico utilizzando la libreria `chart.js`

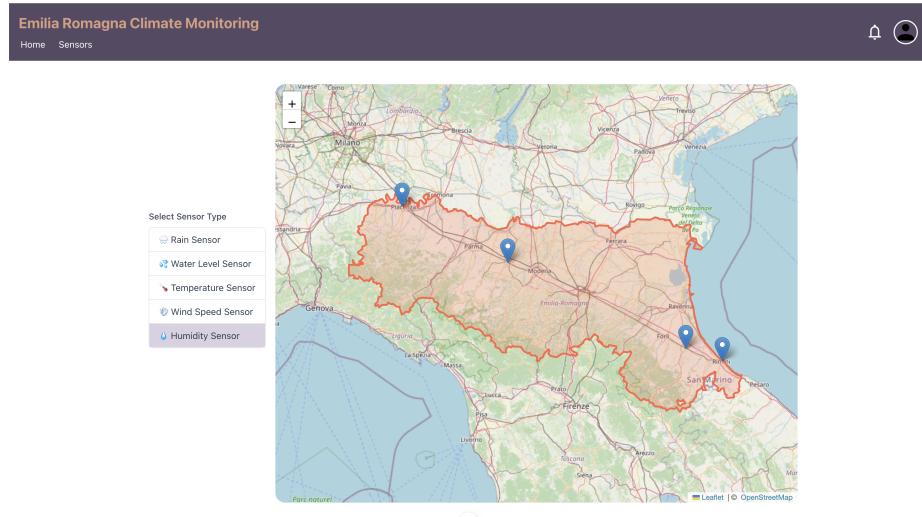


Figura 14: Pagina Home contenente NavBar Mappa e Lista dei sensori



Figura 15: Popup di uno specifico sensore contenente grafico delle detections

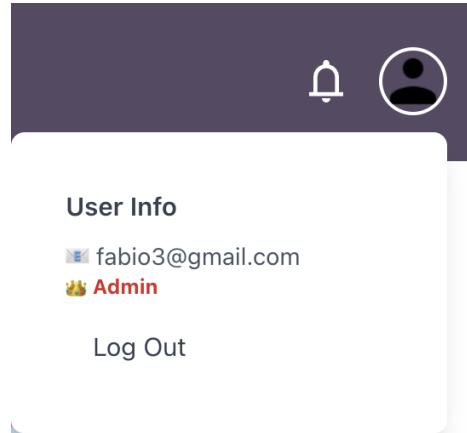


Figura 16: Modale per mostrare le informazioni riguardanti l’utente loggato

**Gestione della visibilità** La navbar mostra o nasconde alcuni elementi in base allo **stato dell’utente**. Per determinare se l’utente è un **amministratore**, viene effettuata una richiesta API con il **token di autenticazione** memorizzato nel local storage. Se l’utente loggato è un admin sarà possibile visualizzare e quindi accedere alla pagina di gestione dei sensori e si vedrà una scritta admin nel modal del profilo.

#### Gestione degli eventi

- **Interazione con la mappa:** Quando un utente clicca su un marker nella mappa viene effettuata una chiamata all’api-gateway per ottenere le ultime rilevazioni del sensore selezionato per poi mostrarle nel grafico, successivamente viene stabilita una connessione attraverso **WebSocket** grazie alla quale ogni volta che il detection-service effettua il salvataggio di una detection per il sensore selezionato quest’ultima viene inviata e mostrata nel grafico del popup
- **Selezione del sensore:** il componente di selezione emette un evento che aggiorna il calore selezionato tramite **v-model**.

#### 4.1.3 Pagina Login

La pagina di login è stata implementata utilizzando la sintassi *Composition API*. Questa pagina consente agli utenti di autenticarsi fornendo la propria *email* e *password* e, se selezionato, permette l’accesso amministrativo tramite un’*API Key*.

**Struttura del Template** Il codice HTML contiene un modulo `<form>` che raccoglie i dati dell’utente. I campi di sono i seguenti:

- **email**: per l'email dell'utente, associato alla variabile v-model="email"
- **password**: per la password dell'utente, associato a v-model="password"
- **isAdmin**: checkbox per selezionare il login come amministratore, collegato a v-model="isAdmin"
- **apiKey**: campo opzionale per la chiave API richiesto solo se il login è amministrativo (v-if="isAdmin")

**Gestione errori** Dal momento in cui stiamo utilizzando un form di compilazione, è possibile che l'utente inserisca delle informazioni non corrette. Per gestire tale evenienza, abbiamo realizzato un componente denominato <ErrorMessage> per mostrare messaggi di errori in maniera dinamica.

**Variabili** All'interno della sezione <script>, sono state definite un'insieme di variabili reattive che permettessero di fare da tramite con i campi del form. Sono state quindi definite le seguenti variabili reattive:

- email, password, apiKey: memorizzano i dati di input.
- isAdmin: determina se il login è amministrativo.
- errorMessage: mostra eventuali errori.
- router: permette di navigare tra le pagine.

**Funzioni di gestione** La pagina di login, per poter funzionare deve riuscire ad interagire con le variabili del componente e deve essere in grado di reagire ai diversi eventi che l'utente di input può far scaturire. Qui di seguito sono elencate le due funzioni principali:

1. **goToRegister()**: permette all'utente di essere reindirizzato verso la pagina di registrazione.
2. **handleLogin()**: controlla se le variabili che fanno riferimento al form sono formattate correttamente, nel caso in cui queste lo fossero viene effettuata una chiamata REST all'api gateway. Una volta ricevuta la risposta, se il codice di ritorno è positivo, l'utente viene reindirizzato verso la home, nel caso contrario un errore verrà mostrato a schermo tramite il componente ErrorMessage.

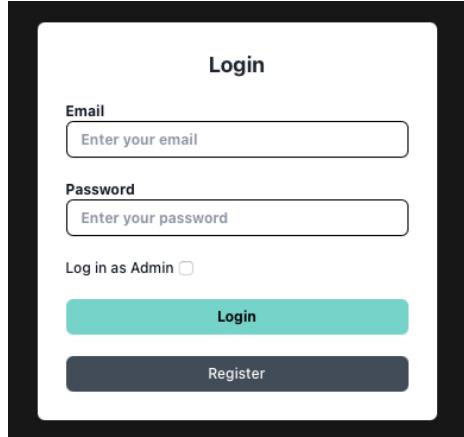


Figura 17: Implementazione della pagina di Login

#### 4.1.4 Pagina Registrazione

L’interfaccia utente di registrazione, è stata sviluppata tramite l’impiego di *Composition API*. Questa pagina permette ai vari utenti di potersi registrare all’interno del sistema utilizzando la propria *email* ed una *password*. All’interno di questa pagina, è importante sottolineare che sarà possibile registrare utenti normali, e non utenti di tipo ’admin’.

**Struttura del Template** Il codice *HTML*, all’interno di tale pagina, contiene un modulo `<form>` che raccoglie i dati dell’utente. I campi sono i seguenti:

- **email**: per l’email dell’utente, associato alla variabile `v-model="email"`;
- **password**: per la password dell’utente, associato a `v-mode="password"`

**Gestione errori** Come visto precedentemente, siccome si sta utilizzando un form per interagire con l’utente, è importante avvisare l’utente in caso vi siano degli errori. Anche in questo caso, si utilizza il componente `<ErrorMessage>` per mostrare eventuali errori all’utente.

**Variabili** La pagina di registrazione, per permettere che il form sia utilizzabile dall’utente, sono state definite un’insieme di variabili reattive per gestire il form.

- `email`, `password`: per memorizzare i dati di input;
- `errorMessage`: mostra l’eventuale messaggio di errore.

**Funzioni di gestione** L’ultima componente che è stata realizzata all’interno del file riguarda tutte le funzioni che permettono di gestire gli eventi di input dell’utente. In questo caso sono state realizzate due funzioni principali:

1. `handleRegister`: il compito di tale funzione è quello di controllare che i dati del form rispettino specifiche regole, come ad esempio che la password abbia almeno 8 caratteri ecc ecc. Nel caso in cui, email e password siano validi, verrà inviata una richiesta HTTP al backend, il quale poi informerà dell'esito dell'operazione, in caso positivo lo user verrà reindirizzato nella pagina principale;
2. `goToLogin`: questa funzione permette di reindirizzare l'utente di input verso la pagina di login.

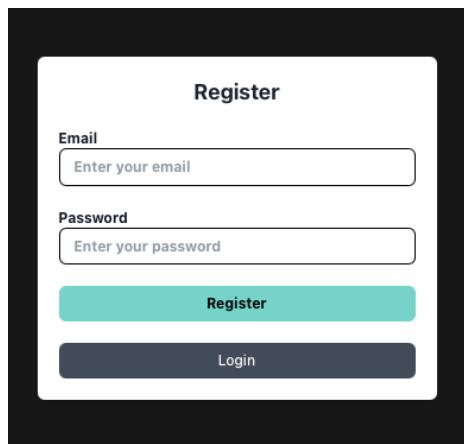


Figura 18: Implementazione della pagina di registrazione

#### 4.1.5 Notifiche

Ogni utente può sottoscriversi a molteplici tipi di notifiche. Ogni sottoscrizione viene mantenuta in maniera persistente all'interno di un database documentale nel backend. Al login dell'utente, viene caricata la lista delle sottoscrizioni dell'utente, e vengono recuperate tutte le notifiche riguardanti ognuna di esse. Essendo un processo complesso e potenzialmente oneroso, è necessario instaurare un meccanismo in modo tale che le notifiche recuperate sopravvivano ad un ricaricamento della pagina: per questo motivo è stato utilizzato un *local storage*; oltre a ciò, quest'ultimo permette la sincronizzazione automatica di ogni componente che ne fa uso, risultando molto utile rispetto ad una soluzione che faccia uso delle direttive `provide/inject`.

In secondo luogo, per la ricezione in tempo reale delle notifiche, è necessario, al login dell'utente e al ricaricamento della pagina, stabilire le connessioni persistenti con il backend. Per questo, vengono inizialmente richieste al backend le sottoscrizioni attive da parte dell'utente corrente, e per ognuna di queste viene attuata la procedura dell'instaurazione della connessione persistente composta dai seguenti step:

1. L'utente recupera e richiede di essere connesso agli eventi a cui era sottoscritto;
2. Il backend risponde con uno UID univoco e un indirizzo (`topicAddr`) con cui è possibile stabilire la connessione persistente. Se l'utente è già registrato con uno UID, viene ritornato quest'ultimo assieme all'indirizzo;
3. Il client presenta attraverso la *room* “`registration`” lo UID e l'indirizzo inviatogli precedentemente;
4. In caso di esito positivo, il client è registrato alla ricezione delle notifiche in tempo reale;
5. Il client può definire la callback per ogni evento in arrivo al topic sul quale è stato sottoscritto.

In questo modo è possibile assicurarsi che le connessioni WebSocket avvengano solo da utenti autorizzati e registrati (tramite UID). Per quanto riguarda l'impiego di un indirizzo per il topic (`topicAddr`), questo riguarda un aspetto principalmente legato alle performance di socket.io: l'utilizzo, lato sender di una *room* permette di inviare un singolo messaggio per più destinazioni, non necessitando quindi di inviare esplicitamente una notifica per ogni utente registrato (*point-to-point communication*), ma seguendo un modello *publish subscribe*.

**Interfaccia Utente** Per quanto riguarda l'interfaccia utente delle notifiche, come già precedentemente descritto in 2.1.2, essa è realizzata mediante un modale mostrabile interagendo con l'icona delle notifiche nell'header. All'interno del modale sono contenuti diversi componenti tra cui: `AlertList`, il cui compito è quello di mostrare le notifiche in tempo reale, `SubscriptionControlPanel` (collapsible) il quale permette all'utente la visualizzazione e gestione delle sottoscrizioni attive e il componente `AlertSelector` (collapsible) per potersi sottoscrivere a nuove tipologie di notifiche. Infine, il modale per la visualizzazione del dettaglio di una notifica è rappresentato dal componente `ExpandedAlert`. È possibile illustrare il posizionamento dei componenti all'interno dell'interfaccia grafica tramite la figura 19.

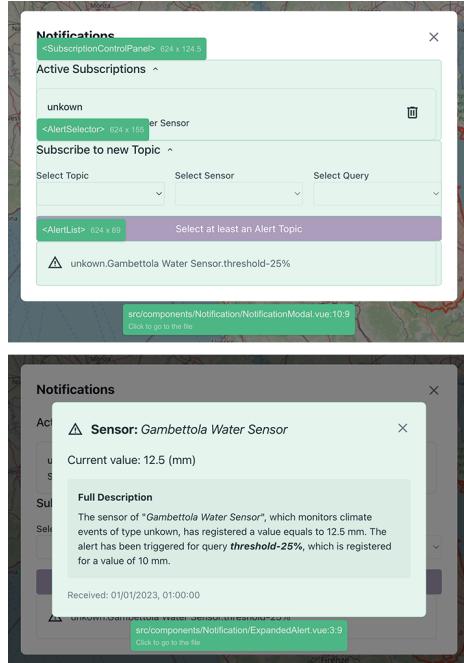


Figura 19: Interfaccia utente del modale delle notifiche, con dettaglio sui componenti di VUE.js

Concludendo, viene esposto infine un ultimo componente per le notifiche: un **NotificationPopup**. Quest'ultimo compare ogni qualvolta una nuova notifica è spedita al client, e viene mostrata nell'angolo in alto a destra della pagina (figura 20). In caso di multiple notifiche, esse verranno sovrapposte l'una all'altra scorrendo verso il basso. Ogni notifica rimarrà nella pagina per esattamente cinque secondi, al termine dei quali scomparirà. È possibile forzare la scomparsa della notifica clickando sulla “x” nell'angolo in alto a destra del popup.

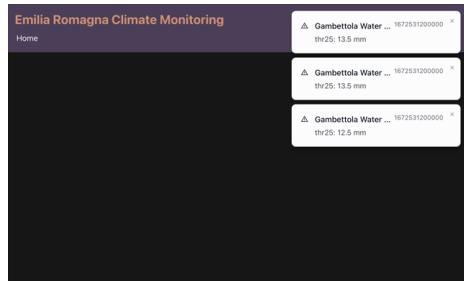


Figura 20: Notifiche pop-up nella all'interno dell'interfaccia utente

#### 4.1.6 Pagina Sensori

L’interfaccia utente per visualizzare tutti i sensori che sono attualmente in uso, abbiamo creato una view apposita contenuta all’interno del file: `ListSensorView`, realizzata sempre tramite le *Composition API*. Questa pagina permette inoltre di effettuare ricerche sui vari sensori che sono visualizzati a schermo.

**Struttura del Template** Il codice *HTML* all’interno di questa pagina si basa su due componenti principali:

1. *Ricerca e filtri*: sezione all’interno della quale viene realizzata la componente di ricerca e di filtraggio dei vari risultati;
2. *Lista dei sensori*: nella seconda componente dell’interfaccia viene visualizzata una lista di `<SensorItem>` i quali fanno riferimento ad un solo sensore.

Attraverso questa divisione delle responsabilità siamo riusciti a mantenere una migliore qualità ed ordine del codice. Ogni componente `SensorItem`, come descritto all’interno della sua sezione, emette due eventi: `removeSensor` e `updateSensor`. Tali eventi sono gestiti da due funzioni che verranno successivamente descritte.

**Variabili** All’interno di questa pagina, sono utilizzate diverse variabili e strutture dati per permettere di visualizzare ed interagire con i vari sensori:

- `sensors` e `displayedSensors`: queste due funzioni fungono da contenitore per tutti i sensori che sono scaricati dal backend, in particolare la variabile `sensors` contiene tutti i sensori, mentre la variabile `displayedSensors` contiene solamente i sensori che sono attualmente visualizzati a video;
- `filters` e `selectedFilter`: la funzione di queste due variabili è in relazione alla funzione di filtraggio che abbiamo previsto, più in particolare la variabile `filters` è una lista di nomi dei filtri che si possono utilizzare, mentre `selectedFilter` contiene il filtro che è selezionato dall’utente.

**Funzioni di gestione** Come precedentemente descritto, questa interfaccia contiene l’insieme di tutti i sensori che vengono scaricati e poi visualizzati all’utente. Per fare in modo che tutti i sensori vengano scaricati e poi visualizzati all’utente una volta che questa pagina è stata caricata, è stato necessario definire un *handler* all’evento `onMounted`. In particolare, all’interno di questa funzione viene effettuata una chiamata al backend per scaricare tutti i sensori e successivamente renderizzarli a video, nel caso in cui l’utente in questione non sia autorizzato a visualizzare questa pagina, verrà automaticamente rediretto verso la pagina di login. Oltre a questa funzione è stato necessario definire gli *handler* ai due eventi che il `SensorComponent` emette, per fare ciò dunque abbiamo definito:

- **removeSensorFromList**: questa funzione viene eseguita ogni qual volta il componente figlio emette l'evento **removeSensor**. All'interno di questa funzione viene rimosso il sensore passato in input assieme all'evento;
- **updateSensorName**: questa funzione viene chiamata solo ed esclusivamente, l'admin ha eseguito una modifica ad uno dei sensori. Questa funzione è l'*handler* dell'evento **updateSensorName** scatenato dal componente figlio. All'interno di questa funzione, vengono utilizzate le informazioni di input passate assieme all'evento per modificare uno specifico sensore.

Infine, abbiamo definito un **watcher** sulla variabile **searchQuery**, in questo modo ogni volta che l'utente inserisce o rimuove un carattere dal motore di ricerca i risultati vengono aggiornati e visualizzati in real-time.

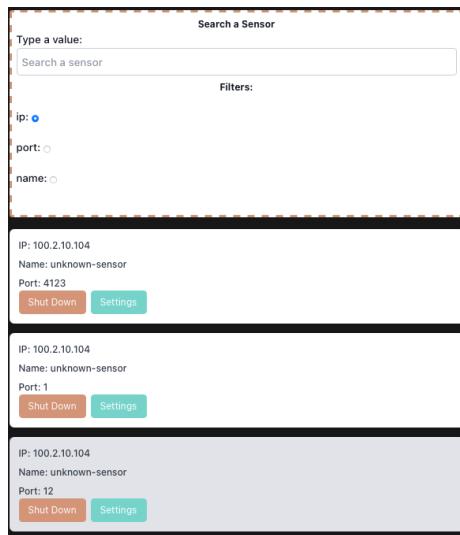


Figura 21: Implementazione della visualizzazione di tutti i sensori

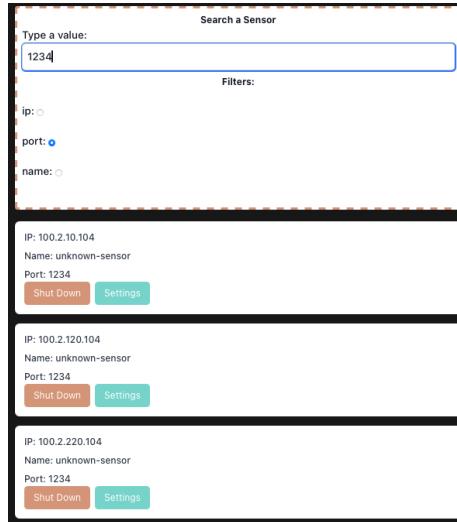


Figura 22: Esempio di ricerca di sensori che contengono un 4 all'interno della porta

#### 4.1.7 Componente singolo della lista

Come mostrato in precedenza, la pagina riguardante l'elenco di tutti i sensori 21, si compone di un'insieme di cards, all'interno delle quali vi sono le informazioni riguardo un singolo sensore. Per fare in modo di avere un singolo componente per ognuno dei sensori, abbiamo pensato di realizzare un componente apposito chiamato `ListComponent`. Questo componente ci permette di fare da tramite tra il componente principale, ovvero l'elenco di tutti i sensori e la pagina riguardanti le impostazioni di un singolo sensore.

**Struttura del Template** Il codice *HTML* definito all'interno del tag `<template>` si comprende di un insieme di tag `h3` per visualizzare le informazioni principali del sensore e due bottoni per interagire con esso, il tasto 'shutDown' emette l'evento `removeSensor` verso il componente padre ed il bottone `settings` invece permette di visualizzare la finestra per interagire con il sensore stesso.

**Variabili** Questo componente definisce delle `props` che gli vengono passate dal componente padre, le proprietà di input sono il *nome* del sensore, l'indirizzo *ip* ed infine il numero di *porta*.

**Funzioni di gestione** Per operare con questo componente e renderlo reattivo, sono state definite alcuni handler per gestire gli eventi principali di tale componente. Ad esempio è stata realizzata la funzione `shutDownSensor` che viene chiamata appena il tasto `shutDown` viene cliccato. Il compito di questo bottone è quello di aprire un modale di conferma per avvertire l'utente se è o

meno sicuro di tale azione, in caso affermativo viene emesso l'evento `removeSensor` il quale successivamente verrà gestito dal componente padre. Per quanto riguarda invece il bottone 'settings' è stato realizzata la funzione `toggleModal` che permette di visualizzare la finestra delle impostazioni del sensore.



Figura 23: Singolo componente della lista

#### 4.1.8 Pagina Impostazioni singolo sensore

Infine, l'utente di tipo admin, oltre a poter visualizzare tutti i sensori è anche in grado di poterne modificare alcuni parametri. Per permettere ciò abbiamo realizzato una finestra modale, il cui compito, è quello di permettere all'utente di modificare alcuni dei parametri del sensore selezionato. In questo caso, tale interfaccia utente è stata sviluppata tramite le *Composition API*.

**Struttura del Template** Il codice *HTML* definito all'interno del tag `<template>` si compone di una struttura principale che ne definisce la finestra modale che verrà successivamente visualizzata a video. All'interno di questa finestra sono poi presenti tre sezioni differenti che fanno riferimento a 3 caratteristiche diverse che si possono modificare riguardanti un singolo sensore.

**Variabili** All'interno di questo componente, è definita una *props* che fa riferimento al nome del sensore stesso.

**Funzioni di gestione** Questo componente ha il compito di modificare alcuni dei parametri di lavoro del sensore stesso, per rendere ciò possibile, sono state definite un'insieme di funzioni per eseguire tali modifiche come ad esempio `changeName`, `changeTime` e `changeDays` i quali, emettono un evento di modifica al componente padre per ravvisarlo dell'avvenuta modifica. In questo caso il componente padre è il `ListComponent`, descritto in precedenza.

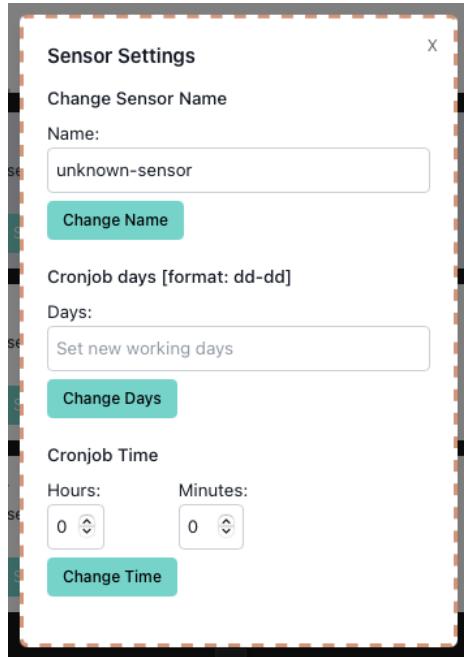
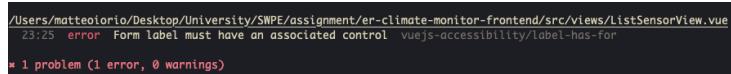


Figura 24: Finestra per le impostazioni di un singolo sensore

## 5 Test

### 5.1 Controllo qualità del codice

Durante lo sviluppo del nostro sistema, ci siamo impegnati costantemente a mantenere un'elevata qualità del codice, un aspetto fondamentale per garantire la robustezza e la manutenibilità del progetto. Per raggiungere questo obiettivo, abbiamo configurato un meccanismo di controllo automatizzato all'interno della nostra repository Git, implementando un *pre-commit hook* utilizzando la libreria *husky*. Un *pre-commit hook* consente di eseguire una serie di comandi predefiniti prima di consentire l'effettivo commit del codice. Se tutti i comandi restituiscono un codice di uscita pari a zero (indicando successo), il processo di commit può proseguire normalmente. In caso contrario, il processo viene interrotto e all'utente viene notificato l'errore riscontrato, spingendolo a correggere il problema prima di poter continuare. Grazie a questa configurazione, siamo stati in grado di introdurre controlli di qualità preliminari in modo sistematico e automatico. Uno dei controlli principali che abbiamo implementato riguarda la verifica della correttezza del codice tramite un linter. All'interno del nostro *pre-commit hook*, viene eseguito il comando `npm run linter`, che analizza il codice per rilevare eventuali errori di sintassi o violazioni delle regole di stile definite. Nel caso in cui vengano individuati problemi, il linter blocca il processo e informa lo sviluppatore, obbligandolo a risolvere tali errori prima di poter completare il commit. La scelta di utilizzare un linter si basa sulla sua grande flessibilità e configurabilità. Infatti, durante lo sviluppo, abbiamo la possibilità di aggiungere nuovi pacchetti e regole per eseguire controlli più specifici in base alle esigenze che emergono. Ad esempio, per garantire un'applicazione più inclusiva e accessibile, abbiamo deciso di integrare un pacchetto che effettua verifiche di accessibilità sui file che contengono sezioni HTML all'interno del tag `<template>`. Grazie a questa integrazione, il *pre-commit hook* notifica lo sviluppatore se alcune parti del codice non rispettano i criteri di accessibilità, offrendo così la possibilità di intervenire tempestivamente per risolvere tali problemi. Infine, il nostro *pre-commit hook* è stato configurato per eseguire un comando combinato: `npm run prelint-staged && npx lint-staged`. Questo comando consente di eseguire verifiche puntuali solo sui file che sono stati modificati, ottimizzando i tempi di esecuzione. In sintesi, grazie a questa soluzione, siamo in grado di mantenere un livello di qualità elevato del codice in ogni fase dello sviluppo, minimizzando errori e migliorando la qualità complessiva del nostro prodotto.



```
Users/matteotorio/Desktop/University/SNPE/assignment/er-climate-monitor-frontend/src/views/ListSensorView.vue
23:25  error  Form label must have an associated control  vuejs-accessibility/label-has-for
* 1 problem (1 error, 0 warnings)
```

Figura 25: Errore violazione di Accessibilità da parte di un componente

Inoltre per migliorare la leggibilità del sistema e fornire una maggiore semplicità nel suo utilizzo è stata utilizzata la tecnica di **dual coding** (figura 26).

Questa tecnica si basa sul collegare il canale *visivo* ed il canale *verbale* (il testo) permettendo al cervello di elaborare e ricordare meglio i concetti. In questo modo, come precisato anche in precedenza nel caso di utenti anziani, riusciamo a rendere più facile l'utilizzo del sistema anche a coloro che non sono pratici nell'utilizzo del computer o del telefonino. Durante la creazione del nostro sito web abbiamo voluto seguire alcune regole fornite da **Jacob Nielsen**, tra cui:

- *Design ed estetica minimalista*
- *Riconoscimento anziché ricordo*
- *Consistenza e standard*
- *Corrispondenza tra sistema e mondo reale*

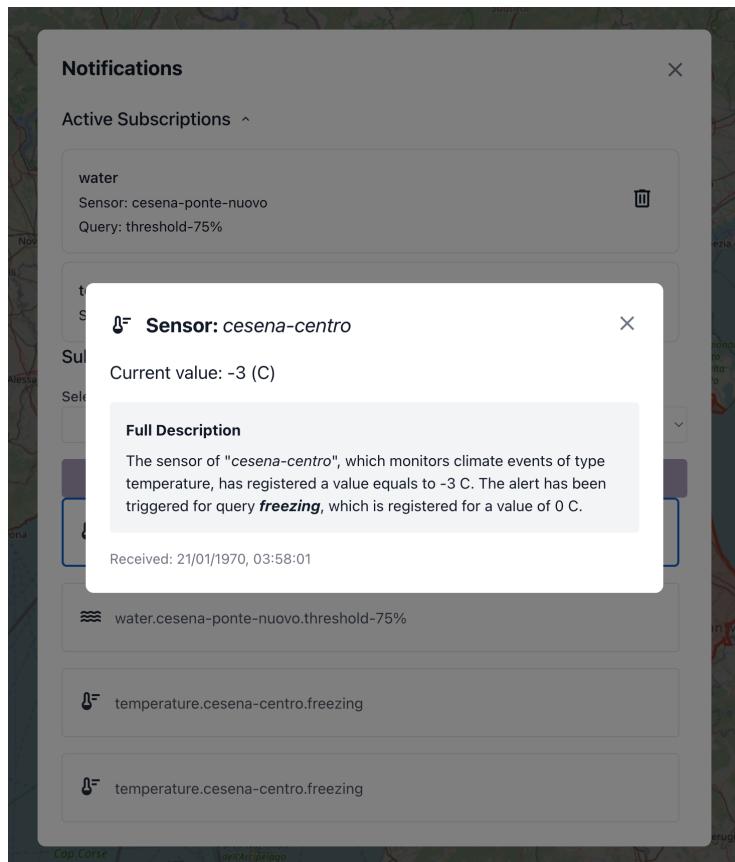


Figura 26: Tecnica di *Dual Coding* per la rappresentazione delle notifiche

## 5.2 Verifica UX

Il servizio che abbiamo realizzato, si interfaccia con un elevato bacino di utenza, a causa proprio degli ultimi calamita climatiche che si sono scatenate contro il territorio Emiliano Romagnolo (ER), provocando danni e vittime. La nostra missione è quella di provvedere un sistema facilmente utilizzabile da qualsiasi tipo di utente. Proprio per questo motivo, abbiamo voluto sviluppare un’interfaccia che sia estremamente semplice da utilizzare in modo che, ad esempio utenti anziani, siano in grado di orientarsi all’interno del nostro sito web. Per verificare che il nostro sito web sia utilizzabile abbiamo fatto utilizzare il nostro sito web ad una decina di utenti, ottenendo i seguenti pareri (per motivi di privacy, i nomi verranno occultati).

- *Utente 1:* il primo utente che siamo andati ad intervistare svolge come attività lavorativa la mansione di *guida alpina*. Essendo il suo lavoro a diretto contatto con la natura, quando viene chiamato per svolgere delle missioni sulla catena degli Appennini, necessita di avere informazioni real-time per le piogge. Questo utente ci ha fornito delle informazioni molto importanti, tra cui la comodità di monitorare in tempo reale lo stato delle rilevazioni e la facilità con la quale si riesce ad accedere a tali informazioni.
- *Utente 2:* il secondo utente intervistato è un grande *amante dell’outdoor*, praticando attività come trekking, escursionismo e ciclismo in natura. Questo utente ha evidenziato l’importanza di un sistema che fornisca previsioni meteo precise e aggiornate, specialmente durante la pianificazione delle sue uscite. Ha anche sottolineato come un’interfaccia semplice e intuitiva sia fondamentale per accedere rapidamente alle informazioni necessarie, evitando distrazioni. Un difetto che questo utente ci ha permesso di individuare era legato alla tonalità dei colori che erano stati utilizzati all’interno del sito, seguendo il suo consiglio abbiamo cercato di utilizzare dei colori che aumentassero il contrasto tra le varie sezioni.
- *Utente 3:* il terzo utente intervistato è un *camionista*, il quale utilizza il sistema principalmente tramite telefono durante le sue pause lavorative. Questo utente necessita di aggiornamenti meteo affidabili, soprattutto per prevenire condizioni di guida pericolose, come piogge intense o nebbia. Durante il test, il camionista ha apprezzato la facilità di utilizzo dell’app su dispositivi mobili e la chiarezza delle informazioni fornite.
- *Utente 4:* il quarto utente è un *impiegato* che vive in città ma si sposta spesso per motivi personali e lavorativi. Questo utente ha evidenziato la necessità di avere una panoramica generale delle condizioni meteo, soprattutto per pianificare spostamenti giornalieri o brevi viaggi. Ha sottolineato l’importanza di notifiche personalizzate, che permettono di ricevere avvisi specifici senza dover controllare continuamente l’applicazione.
- *Utente 5:* il quinto utente è uno *studente di filosofia* che si sposta frequentemente tra casa e università. Ha testato il sistema principalmente

per organizzare le sue giornate di studio e svago. Lo studente ha sottolineato l'utilità di un'interfaccia semplice e la possibilità di visualizzare rapidamente le temperature nelle zone circostanti, in modo da pianificare le sue attività senza preoccupazioni legate al maltempo.

## 6 Deployment

Le istruzioni illustrate all'interno di questa sezione rispecchiano le istruzioni contenute all'interno del file `README.md` della repository del progetto. Consigliamo caldamente di utilizzare quelle istruzioni per i comandi per via dei problemi del copia-incolla dei documenti PDF.

### 6.1 Backend

L'intero progetto è stato sviluppato all'interno di 3 repository differenti, rispettivamente una per la simulazione dei sensori, una per il backend e una per il frontend. Utilizzando i `submodules` di git è quindi possibile clonare il repository come segue:

```
1 git clone --recurse-submodules https://github.com/MatteoIorio11/er-climate-monitor.git
```

All'interno del progetto è possibile dapprima avviare tutti i servizi necessari su quattro docker container attraverso lo script `deploy-services.py`, per il quale è necessario installare i *requirements* attraverso `pip install -r requirements.txt`. Lo script permette di avviare tutti i servizi tramite l'argomento `up`, stoppare tutti i servizi con `down` e stoppare tutti i servizi e rimuovere le risorse (immagini, volumi, reti) tramite l'argomento `downrmi`.

Infine, lanciare l'istanza dell'*api-gateway*, spostandosi all'interno della cartella `api-gateway`, tramite il comando: `npm i && npm run build && npm run start`.

### 6.2 Costruire e Lanciare i sensori

Inizializzati i servizi e lanciata l'istanza dell'*api-gateway*, è possibile avviare i sensori situati all'interno di `er-climate-monitor-sensors/sensor`. In questa directory sono specificati vari sensori attraverso una serie di file di configurazione YAML. Per creare ed avviare tutti i sensori utilizzare lo script `build_mockup_sensors.py`, per il quale è necessario installare (come sopra, oppure utilizzando `uv` tramite `uv sync` o `uv pip install -e .`) i *requirements*. Lanciando il comando vengono generati i file di configurazione all'interno della cartella `sensors_config`, e successivamente è possibile lanciare lo stesso script con l'argomento `create`. In questo modo verranno generati una serie di script python, ognuno per ogni sensore. Una volta terminata l'operazione, tornare alla directory `er-climate-monitor-sensors` e avviare i sensori tramite: `python -m sensor.sensor_<NomeSensoreSenzaEstensione>`. Osservando i messaggi di log è possibile capire come l'operazione è andata a buon fine. Di seguito un piccolo riassunto dei comandi da lanciare partendo dalla radice della repository:

```
1 # Directory corrente: er-climate-monitor
2 cd er-climate-monitor-sensors/sensor
3 ./build_mockup_sensors.py create
4 cd ..
5 python -m sensor.sensor_<NomeSensoreSenzaEstensione>
```

È possibile ripulire la directory corrente da tutti i sensori generati tramite l'argomento `clear`: `./build_mockup_sensors.py clear`.

### 6.3 Frontend

Lanciare `npm install` e `npm run dev`. Una volta fatto ciò bisognerà seguire le indicazioni che verranno visualizzate a schermo, così facendo sarà possibile raggiungere il nostro sito web. Successivamente, per navigare all'interno di esso, non basterà altro che seguire le indicazioni fornite all'interno del capitolo 3. È importante sottolineare che qualsiasi utente che visita il nostro sito web viene trattato come un utente anonimo, privo di alcuna informazione. Se questo utente dovesse cliccare su una delle tipologie di evento atmosferico posizionato vicino la mappa, verrà automaticamente reindirizzato verso la pagina di login, dal momento in cui solamente gli utenti registrati possono accedere alle informazioni che il nostro sito fornisce.

## 7 Conclusioni

### 7.1 Sviluppi Futuri

Ora che abbiamo terminato lo sviluppo del nostro sito web, non ci resta che fare delle considerazioni su ciò che si potrà aggiungere nel futuro:

- Sicuramente sarà necessario, cercare di standardizzare tutti i sensori che si vorranno collegare, in questo modo informazioni come giorno e orario di campionamento saranno facilmente visualizzabili tramite il menu delle impostazioni sul singolo sensore;
- La seconda funzionalità che si potrebbe andare a realizzare nel futuro riguarda la possibilità di notifiche tramite messaggio su dispositivi come smartphone, in questo modo l'utente non avrà la necessità di rimanere costantemente sul nostro sito web;
- La terza funzionalità che si potrà implementare riguarda il perfezionamento dei grafici relativi ai singoli sensori. In particolare, sarà possibile aggiungere una nuova schermata dedicata, dove visualizzare dati tabulari rappresentativi delle diverse categorie di fenomeni atmosferici.
- La quarta funzionalità riguarda l'aggiunta di un'altra pagina, all'interno della quale saranno contenute le news riguardanti gli ultimi eventi atmosferici che si sono abbattuti sul territorio Emiliano Romagnolo.
- La quinta funzionalità prevede l'opportunità di scegliere differenti tipologie di grafici.
- La sesta funzionalità riguarda la raccolta di dati storici in modo tale da fornire informazioni riguardanti i valori per la giornata corrente in una finestra temporale, i massimi storici, i minimi storici, ...; tutto questo non

farà altro che incrementare l’“awarness” degli utenti per quanti riguarda gli eventi che caratterizzano il cambiamento climatico nel corrente periodo storico.