



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA E
DELL' AUTOMAZIONE

**Realizzazione di un'interfaccia Web basata
sulla tecnologia VUE per un'applicazione di
gestione del benessere ambientale in spazi
chiusi**

**Creation of a web interface based on VUE technology
for an indoor environmental wellness management
application**

Relatore:
Prof. Alessandro Cucchiarelli

Candidato:
Francesco Voto

Anno Accademico 2020-2021

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE
Via Brecce Bianche – 60131 Ancona (AN), Italy

Indice

Introduzione	1
1 Contesto Applicativo	4
1.1 Il CPMS	4
2 Obiettivi del progetto	8
2.1 Il Mockup dell'interfaccia	9
2.2 Elementi caratterizzanti	10
3 Strumenti utilizzati	12
3.1 Vue.js	12
3.1.1 Caratteristiche dei componenti	15
3.1.2 Lifecycle di un'istanza Vue	16
3.1.3 La Reattività di Vue	18
3.1.4 Data binding bidirezionale	19
3.2 Principali Direttive Vue	20
3.3 Vue e Laravel	22
3.4 Database Mysql	23
3.5 I componenti	24
3.5.1 Balkan OrgChartJs	24
3.5.2 Fusion Chart	26
3.5.3 Highcharts	29
4 Applicazione sviluppata	31
4.1 Controller	31
4.1.1 On/Off	32
4.1.2 Step	34
4.2 Subsystems	35
4.2.1 Status	36
4.2.2 FCU0-WIN0	38

Indice

4.2.3	SHA0-Boiler-Chiller-Summer	39
4.3	Diagrams	43
4.4	KPI	49
4.5	Messages	54
4.6	History	58
4.7	Il Controller	64
4.8	Il Layout	65
5	Conclusioni	67
5.1	Miglioramenti	70

Elenco delle figure

1	Architettura Olonica	7
2	Mockup Interfaccia	9
3	Logo Vue.js	12
4	Angular,React e Vue a confronto	13
5	Struttura file single component	14
6	Tag <script> di un file vue.js	15
7	Lifecycle di un istanza Vue	16
8	Come vengono gestioni i dati da Vue	18
9	Esempio binding bidirezionale	19
10	Versione demo dell'albero della libreria Balkan OrgChartJs	25
11	Sezioni di un indicatore angolare della libreria fusionChart	26
12	Esempio di indicatore angolare	28
13	Sezioni che compongono un grafico della libreria highChart	29
14	Layout del controller dell'interfaccia	31
15	Codice html per la realizzazione dell'On/Off	32
16	Regole css del pulsante On/Off	33
17	Codice html per la realizzazione della select	34
18	Regole css del pulsante di selezione	34
19	Layout del componente subsystems	35
20	Codice html del pulsante status	36
21	Regole css per la realizzazione del pulsante status	37
22	Codice per la realizzazione della select del sistema FCU0	38
23	Codice html del sistema Boiler	39
24	Regole css per la realizzazione del pulsante status	39
25	Tabella subsystem del database	40
26	Model del componente subsystems	41
27	Codice per l'acquisizione dei dati	42

Elenco delle figure

28	layout componente diagrams	43
29	Attributo chart dell'albero	44
30	Finestra popup per la visualizzazione di dati aggiuntivi .	45
31	Recupero dati dal database	45
32	Tabella diagrams del database	47
33	Model del componente diagrams	48
34	Layout componente KPI	49
35	Codice html per la realizzazione della select	50
36	Attributi dell'elemento fusionChart	51
37	Elementi che caratterizzano l'attributo dataSource	51
38	Recupero dati dal database	52
39	Tabella KPI del database	53
40	Model della tabella kpi	53
41	Layout del componente message	54
42	Codice html del componente	55
43	Funzione per il recupero dei dati dal database	55
44	tabella message del database	56
45	Model della tabella kpi	56
46	Regole css casella messaggi	57
47	Layout componente history	58
48	Codice html per la realizzazione del grafico	59
49	Funzione chooseChart per lo switch fra i KPI	60
50	Attributo chartOption per la costruzione del grafico . . .	60
51	Model tabella kpihistory	61
52	recupero dati dal database	62
53	tabella kpihistory	63
54	Il Cotroller dell'interfaccia	64
55	codice della vista dell'interfaccia	65
56	Dichiarazione tag html dei componenti vue	66
57	Interfaccia Finale	67
58	Importanza User Experience	69

Introduzione

Questo progetto nasce dall'esigenza di costruire un'interfaccia in grado di gestire e visualizzare lo stato di sistemi, utilizzati nel controllo e nel miglioramento del benessere ambientale di edifici che hanno appena superato la fase di ristrutturazione.

L'intero sistema è formato da un insieme di diverse parti, le quali permetteranno la realizzazione del CPMS (Construction Process Management Service) di cui parleremo nella sezione 1.1. Il nostro compito sarà quello di realizzare la dashboard, ossia l'interfaccia, che permetterà all'utente la gestione dei sistemi e di tutte le sue funzionalità.

Il sistema controllerà l'illuminazione, il riscaldamento dell'ambiente, la ventilazione, il riscaldamento dell'acqua ecc.

L'utente, ovvero uno dei conduttori dell'immobile appena ristrutturato, avrà la possibilità di visualizzare feedback specifici sullo stato di comfort dell'ambiente che lo circonda. Gli stessi feedback supporteranno l'utente nella scelta delle azioni da apportare al fine di potenziare le prestazioni dell'edificio, migliorando il comfort e minimizzando gli sprechi.

Introduzione

La nostra interfaccia può essere vista come il “ponte di comunicazione” fra l’utente e il sistema. Sarà formata da sei componenti differenti: Controller, Subsystems, Diagram, KPI, Message, History.

Discuteremo dei componenti in seguito, per il momento immaginiamo ognuno di essi come un blocco separato, indipendente l’uno dall’altro. Tutti saranno parte integrante di un’unica interfaccia. Quest’ultima permetterà la gestione ottimale dei dati rilevati e la visualizzazioni delle azioni che il sistema ci suggerisce, al fine di migliorare il benessere di vita all’interno dell’immobile, diminuendo i consumi di energia e di conseguenza le emissioni di CO₂ dannose per il nostro pianeta.

Per una corretta analisi dei dati ambientali verranno tenuti in considerazione anche parametri che non sono dominanti sul sistema ma che allo stesso tempo possono condizionarlo, come ad esempio sarà rilevante il momento in cui i dati vengono analizzati poiché verrà tenuto in considerazione il tasso di occupazione dell’edificio.

Il nostro compito sarà quello di creare un’interfaccia, avvalendoci dell’utilizzo della tecnologia vue.js, che ci aiuterà e ci semplificherà il processo di implementazione. Verrà sviluppata un’interfaccia responsive in cui l’utente vedrà sempre gli elementi importanti della pagina e potrà navigare con facilità da una sezione all’altra a prescindere se stia utilizzando il proprio telefono, un computer o un tablet.

Introduzione

L'intero sistema nasce con l'idea di funzionamento autonomo senza l'intervento dell'uomo. Esso, sarà in grado di migliorare le sue prestazioni e sarà sempre più accurato nel prevedere eventi che potrebbero verificarsi.

La nostra interfaccia sarà un insieme di oggetti grafici rappresentati in una forma strutturata, che permettono di rendere accessibili a colpo d'occhio informazioni di diversa natura e complessità.

Lo scopo principale della dashboard sarà la visualizzazione di tutti i sistemi e il loro relativo stato, però questo non toglie all'utente la facoltà di scegliere alcune delle azione suggerite dal sistema al fine di contribuire al raggiungimento degli obiettivi di benessere definiti.

Lascerà quindi la possibilità di prediligere l'azione che si desidera, ma allo stesso tempo sarà resistente ai comportamenti errati dell'utente, limitando le scelte tra quelle che il sistema valuta vantaggiose.

Nelle prossime sezioni descriveremo dettagliatamente il contesto nel quale si colloca il progetto che abbiamo sviluppato, con i relativi requisiti che dovrà soddisfare. Allo stesso tempo illustreremo gli obiettivi e gli scopi della nostra interfaccia, ovvero per cosa sarà utile. Verranno descritte tutte le tecnologie utilizzate per la realizzazione e come questa si è articolata. La descrizione sarà sufficientemente puntuale per ognuno dei sei componenti che ne faranno parte e verranno infine commentati gli aspetti che potranno essere migliorati all'interno di essa.

Lo scopo del lavoro illustrato in questa tesi è quindi quello di costruire un'interfaccia che permetta di mettere in relazione l'utente con il sistema (CPMS) al fine di migliorare le condizioni di vita e di benessere all'interno dell'edificio.

Capitolo 1

Contesto Applicativo

In questa sezione cercheremo di illustrare l'utilità e i motivi per i quali è essenziale una dashboard. Cercheremo di capire il contesto nel quale si colloca il nostro progetto e quali problematiche risolve.

L'interfaccia di cui tratteremo da qui in avanti è riferita ad un sistema che permetterà il miglioramento del benessere ambientale in edifici che hanno appena completato la loro fase di ristrutturazione. Questo verrà fatto attraverso il rilevamento e l'analisi dei fattori ambientali al fine di migliorare il comfort degli abitanti.

1.1 II CPMS

Il sistema di cui stiamo parlando prenderà il nome di CPMS(Construction Process Management Service) appartenente alla piattaforma ENCORE.

Il servizio CPMS fornisce dei suggerimenti che possono essere adottati non solo per una migliore conduzione dell'edificio ma anche per il monitoraggio delle evoluzioni che avvengono dopo i lavori di ristrutturazione. L'edificio verrà controllato e continuamente saranno migliorati i fattori ambientali secondo dei parametri ideali per il benessere dell'abitante.

Per una corretta valutazione dei parametri ideali verranno presi in considerazione i diversi orari di occupazione dell'edificio e il sistema si occuperà della gestione di illuminazione, riscaldamento e raffreddamento, riscaldamento dell'acqua e infiltrazioni. Il tutto servirà anche per migliorare le prestazioni energetiche dell'intero edificio.

Il prototipo del sistema è realizzato attraverso l'utilizzo di un modello semplificato, caratterizzato da parametri concentrati riguardanti sia il comfort che le prestazioni energetiche. Viene associato a questo insieme

Capitolo 1 Contesto Applicativo

di parametri un adeguato set di indicatori di prestazione . Il modello sarà in grado di sollecitare il cambiamento dei parametri attuali al fine di migliorare il comfort e il comportamento energetico dell'intero edificio.

IL CPMS permetterà una gestione intelligente dell'edificio ristrutturato per tutta la durata di vita. Verrà effettuata costantemente una valutazione dei parametri di comfort e sarà mantenuto uno storico della qualità dei servizi che sono forniti all'edificio nel suo ciclo di vita.

Nel dettaglio l'intero prototipo è formato da:

- **Un modello** semplificato con parametri concentrati riguardanti le prestazioni energetiche e i valori di comfort che consentono la rappresentazione del sistema. Il fine è quello di consentire il monitoraggio e le azioni che possano influenzare positivamente i comportamenti di comfort ed energia;
- **Un'applicazione Web**, ossia la nostra interfaccia, che viene utilizzata per supportare gli abitanti dell'edificio ristrutturato nella gestione dei sistemi, suggerendo loro azioni per migliorare le prestazioni o fornendo informazioni sullo stato del sistema così da valutare eventuali rettifiche.
- **Un protocollo di comunicazione REST-API** per acquisire i dati ambientali e i dati dai sensori o dai sistemi BMS presenti nell'edificio e quindi permettere ai modelli di valutare le prestazioni e alla Web app di elaborare questi dati;
- **Una registrazione delle azioni** che vengono eseguite dall'utente durante una sessione, così da rendere più facile al proprietario e a tutte le parti interessate, il confronto dei risultati attesi con le prestazioni ottenute.

I servizi offerti dal CPMS costituiscono una parte essenziale nella ristrutturazione, poiché racchiuderà i servizi necessari per eseguire correttamente il lavoro e controllare le prestazioni dell'edificio sotto vincoli di sostenibilità.

Il sistema supporterà gli abitanti nel momento in cui prenderanno possesso dell'immobile, fornendo una gestione intelligente del comfort attraverso una valutazione in tempo reale dei parametri ambientali.

Capitolo 1 Contesto Applicativo

L'obiettivo del CPMS è quello di un sistema intelligente basato sul paradigma olonico, ossia un sistema auto-organizzante che perfeziona le sue prestazioni nel corso del tempo attraverso l'analisi dei dati correnti al fine di migliorare le sue capacità. Può lavorare sia autonomamente sia che come un sistema di supporto che suggerisce all'utente le migliori azioni o le migliori pratiche nella fase di conduzione dell'edificio ristrutturato. Le sue caratteristiche gli permettono di supportare l'utente per ridurre il degrado delle prestazioni dell'edificio, essendo resiliente alle azioni errate dell'utente o al degrado dei componenti.

Fondamentalmente, il CPMS viene attivato alla fine dei lavori di ristrutturazione, qualora essi siano stati svolti rispettando i parametri e le regole esistenti per l'integrazione del servizio. L'obiettivo perseguito da CPMS sarà quindi quello di massimizzare il comfort minimizzando il consumo di energia e le emissioni di carbonio.

IL CPMS permetterà quindi di gestire il ciclo di vita dei lavori di ristrutturazione, definendo un meccanismo che fornisce continuamente feedback specifici. Inoltre suggerirà le azioni che potranno essere effettuate per influenzare il comportamento sia a breve che a lungo termine durante l'intero ciclo di vita della ristrutturazione. Grazie all'architettura olonica adottata per questo servizio, i sottosistemi saranno trattati come un insieme di oloni (una parte a se stante che però è parte integrante di un sistema più grande e complesso, vedi Figura 1). In particolare, uno degli obiettivi principali del sistema sarà la costruzione di un albero deterministico di sotto-obiettivi che rileva le opportunità per il miglioramento continuo della struttura. Questo costituirà un sistema diagnostico agile per il rilevamento in tempo reale di problemi e mancanza di efficienza. Potrebbe anche dar origine a un database di lezioni apprese per l'analisi e la gestione delle informazioni a lungo-medio termine.

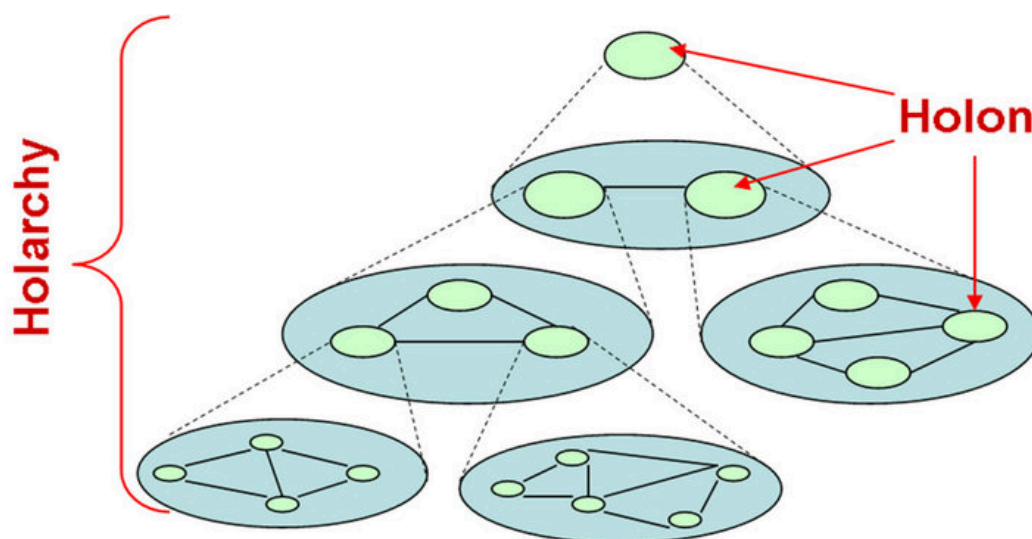


Figura 1: Architettura Olonica

La caratteristica ideale e più sofisticata dell'intero sistema è quella di agire in modo autonomo. In particolare, CPMS è orientato a introdurre la filosofia e il concetto di *"kaizen"* (miglioramento continuo) nel settore delle costruzioni e la sua automatizzazione.

Nelle prossime sezioni non ci dilungheremo sugli aspetti tecnici e sulle varie parti che compongono il CPMS ma ci soffermeremo sulla realizzazione della dashboard che sarà essenziale per la comunicazione diretta fra il sistema e l'utente, ossia quella parte che si occuperà di fornire i suggerimenti delle azioni al fine di migliorare le prestazioni e allo stesso tempo di una visualizzazione immediata e intuitiva dello stato dei sistemi. I dati che vengono rappresentati saranno prima analizzati dal CPMS stesso.

Capitolo 2

Obiettivi del progetto

L'obiettivo del progetto è incentrato sulla realizzazione di un'interfaccia Web per la realizzazione di un'applicazione per la gestione del benessere ambientale negli spazi chiusi. Il tutto tramite l'utilizzo della tecnologia VUE.js, un framework che permette la realizzazione di interfacce web reattive che quindi sfruttano il binding tra modello dati e vista, attraverso la creazione di singoli componenti che potranno essere aggiunti alla nostra pagina web con estrema semplicità.

Lo scopo del progetto è quindi lo sviluppo di una **dashboard**, ossia una singola pagina definita come "area di disegno" in cui verranno contenuti alcuni componenti che aiutano a riportare le informazioni chiave di un sistema con una presentazione dei dati molto intuitiva.

Le diverse sezioni mostrate in una dashboard vengono chiamate riquadri, i quali, anche se contenuti nella stessa pagina, agiscono come oggetti separati l'uno dall'altro. Ognuno di essi si occuperà distintamente della gestione di un set di dati.

La dashboard è un ottimo strumento per monitorare le attività di un sistema, è utile per avere risposte e controllare istantaneamente i parametri più significativi del sistema preso in considerazione.

Diversamente da quel che si può pensare, una dashboard non è un'immagine inanimata contenente dei dati da visualizzare, ma bensì è un oggetto interattivo nel quale i dati potrebbero variare a seguito di eventi generati dall'utente o dall'ambiente esterno.

E' proprio grazie all'utilizzo di Vue.js che si è stato in grado di sviluppare i sei componenti che caratterizzano l'intera interfaccia; essa ricaverà i valori di stato dall'ambiente e tramite l'utilizzo di grafici, indicatori angolari, diagrammi e pulsanti permetterà una visione fruibile, semplice e ottimale dei dati stessi e garantirà l'esecuzione di funzioni in grado di sollecitare il cambiamento del sistema.

I componenti utilizzati dovranno soddisfare i requisiti concepiti in fase di progettazione.

2.1 Il Mockup dell'interfaccia

Il seguente mockup descrive a livello visivo come l'interfaccia era stata ideata prima dell'implementazione(Figura 2):

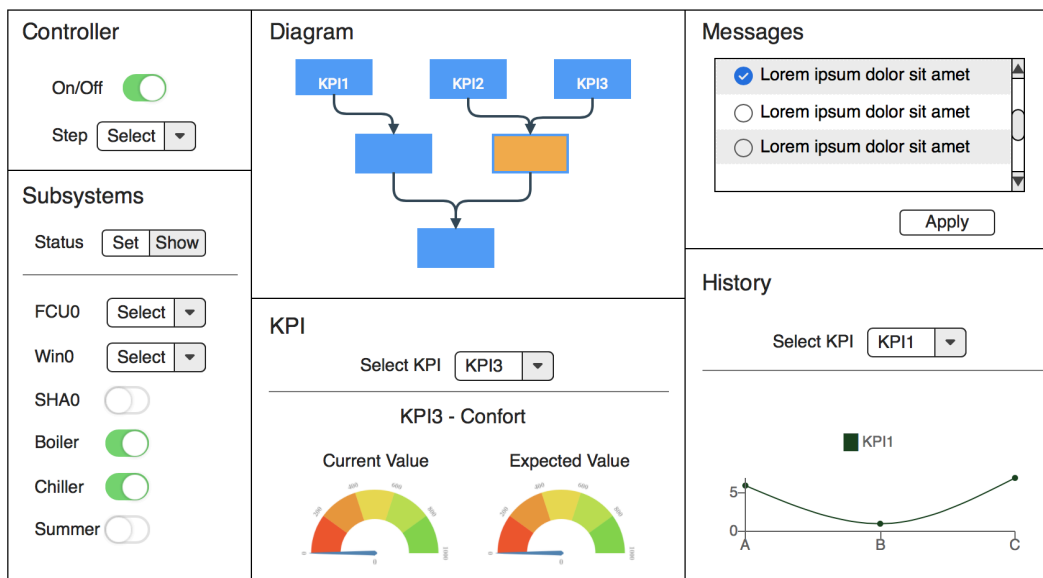


Figura 2: Mockup Interfaccia

L'interfaccia sarà quindi formata dal:

- **Controller:** si occuperà dell'avvio e dello spegnimento del sistema nonchè della possibilità di scegliere il relativo Step.
- **Subsystems:** permetterà o la semplice visualizzazione dello stato oppure la possibilità di modifica dei sistemi FCU0, WIN0, SHA0, Boiler, Chiller e Summer;

- **Diagrams:** tramite l'utilizzo di un albero garantirà la visione dello stato del sistema e la possibilità di visualizzare il messaggio del relativo nodo preso in considerazione tramite una finestra popup;
- **KPI:** componente che permetterà la selezione del particolare KPI(key performance indicator) che si vuole analizzare e quindi visionare lo stato attuale e il valore atteso dallo stesso;
- **Messages:** conterrà i messaggi con i possibili cambiamenti da apportare al sistema, con la possibilità di selezionarne uno;
- **History:** visualizzerà i valori dei KPI che si vuole prendere in considerazione e quindi mapperà i valori forniti dal sistema stesso in funzione del tempo.

L'intera interfaccia elaborerà attraverso i suoi componenti i dati rilevati, che verranno contenuti all'interno di una database relazionale e di conseguenza elaborati al fine di garantire una semplice e immediata analisi dello stato del sistema. L'utente sarà quindi in grado di trarre, tempestivamente e intuitivamente, le proprie considerazioni solamente attraverso la visualizzazione ottimale dei relativi dati.

I sei componenti dovranno essere suddivisi in sei sezioni differenti, distinte e marcate. I dati che costituiscono l'interfaccia saranno caricati in modo indipendente da ogni componente, il quale estrapolerà i dati contenuti in una tabella del database.

Per la realizzazione dell'interfaccia è richiesto l'utilizzo di Laravel 8.0, un framework php che permette facilmente lo sviluppo del codice in team, nonché facilità un'implementazione modulare e standardizzata. Approfondiremo l'argomento nella sezione 3.3

2.2 Elementi caratterizzanti

Per la realizzazione dell'intera interfaccia, come si può intuire dalla Figura 2, verranno utilizzati alcuni elementi molto comuni nel mondo dell'informatica.

Capitolo 2 Obiettivi del progetto

Saranno implementati dei:

- **toggleButton**: elemento che viene utilizzato per visualizzare lo stato selezionato o non selezionato di un pulsante. ToggleButton è fondamentalmente un pulsante di accensione/ spegnimento con un indicatore luminoso che indica lo stato attuale del pulsante di attivazione/ disattivazione.
- **selectButton**: è un elemento che viene utilizzato per creare un elenco a discesa. Spesso viene utilizzato in un modulo per raccogliere l'input dell'utente.
- **Albero**: è un elemento composto da due tipi di sottostrutture fondamentali: il nodo, che conterrà le informazioni e l'arco, che invece stabilirà un collegamento gerarchico fra due nodi. Parleremo di un nodo padre dal quale esce un arco orientato che lo collega ad un nodo figlio.
- **Indicatore Angolare**: elemento che visualizza i valori dei dati su una scala radiale, la quale è contrassegnata da un limite inferiore e da un limite superiore, ovvero i valori di massimo e minimo che possono essere tracciati. Internamente all'elemento, si possono creare varie sezioni per classificare i dati: ogni parte può avere un colore di sfondo, un colore del bordo, linee di separazione, ecc.
- **messageBox**: può essere definita come una casella di messaggi, in cui l'utente può visualizzare un messaggio. È possibile visionare le didascalie dei messaggi all'interno della casella e i pulsanti di comando in ognuna delle sezioni presenti.
- **Grafico xy**: è un elemento utilizzato per la rappresentazione di coppie di valori. Nella nostra interfaccia, sarà utilizzato per una descrizione in funzione del tempo dello stato del KPI. I valori dei KPI verranno mostrati nell'asse delle ordinate, mentre l'ascissa rappresenterà la sequenza temporale nella quale i dati vengono rilevati.

Capitolo 3

Strumenti utilizzati

In questa sezione verranno illustrate tutte le tecnologie che sono state utili per la realizzazione del progetto. In successione vedremo le caratteristiche di ognuna di esse e le potenzialità che le caratterizzano, evidenziando quanto realmente possano semplificare lo sviluppo di applicazioni.

3.1 Vue.js



Figura 3: Logo Vue.js

Per lo sviluppo dell'intera applicazione, come annunciato nelle sezioni precedenti, ci si è avvalsi del framework vue.js(Figura 3).

“Vue, anche conosciuto come vue.js, è un framework Javascript open source per la creazione di interfacce utente (UI, User Interface). Si tratta di un framework progressivo, ovvero è pensato per essere adottabile in modo incrementale - a differenza dei framework monolitici.

Vue è usato principalmente per lo sviluppo front-end, ed è focalizzato sul livello di visualizzazione ViewModel all'interno di una struttura MVVM. Le funzionalità di Vue consentono di estendere l'HTML, creare modelli e componenti riutilizzabili e progettare interfacce utente reattive e mobile-first grazie ad un'API semplice e flessibile. [1]."

Capitolo 3 Strumenti utilizzati

Vue.js, come descritto in precedenza, viene definito un framework progressivo. Con progressivo si intende l'idea di suddividere il codice in parti distinte per poi riassembrarle, rispecchiando il concetto del "dividi et impera".

Vue.js è specificamente utilizzato per creare viste HTML, non fornisce tante funzionalità e librerie native come accade ad esempio per Angular, ma consente una facile integrazione con altre librerie esterne e componenti del progetto.

Tuttavia, ancora non è presente una grande azienda che sostiene Vue.js. Nonostante questo, il framework cattura ancora molta attenzione da parte della comunità di sviluppatori web, come ad esempio su GitHub, dove è uno dei più apprezzati.

Detto questo, si può dedurre che anche se la maggior parte delle aziende sta ancora usando React, le persone stanno rivolgendo la loro attenzione verso Vue.js e, in futuro, potrebbe diventare sempre più popolare.

Di seguito, verranno illustrate le percentuali di utilizzo di vue rispetto ai principali competitors(Figura 4):

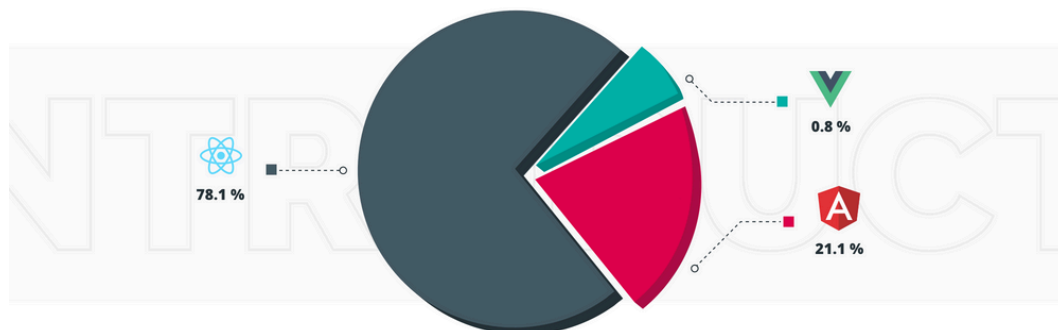


Figura 4: Angular,React e Vue a confronto

"Vue.js è stato rilasciato nel 2014 da Evan You, che stava lavorando per Google utilizzando Angular prima di concentrarsi sullo sviluppo di un framework tutto suo [2]". Vue.js è un framework basato su componenti da cui gli sviluppatori possono creare elementi personalizzati, che possono riutilizzare in tutto il progetto. Questa caratteristica non è esclusiva di Vue.js poiché anche React e Angular hanno questa peculiarità.

Capitolo 3 Strumenti utilizzati

La particolarità di Vue.js sta nel fatto che un componente può includere HTML, CSS e JavaScript insieme, formando il cosiddetto “single file component”. In esso, il codice HTML è scritto all’interno del tag `<template>`, mentre il codice JavaScript e CSS sono contenuti, rispettivamente, nei tag `<script>` e `<style>`.

La Figura 5 mostra come è strutturato un single file component.



Figura 5: Struttura file single component

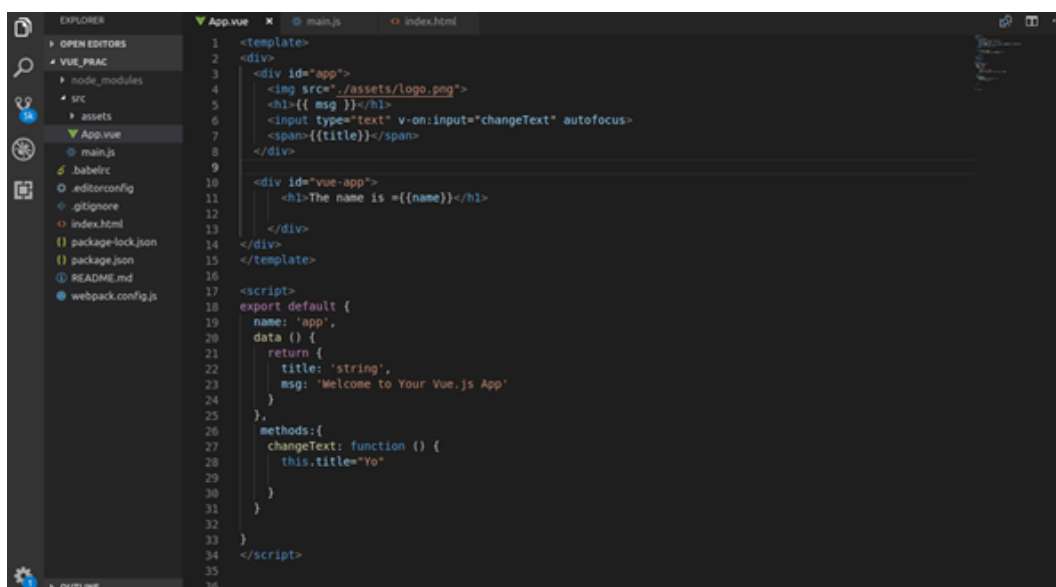
Nei progetti più complessi, potrebbe risultare molto vantaggioso sviluppare diversi componenti autonomi e integrarli solo in un successivo momento, poichè questo processo favorisce una modularizzazione del codice e quindi una gestione molto più semplice e ottimale. Per la creazione di un singolo componente basterà sviluppare il codice relativo all’oggetto che si sta realizzando all’interno di un file con l’estensione “.vue”. Ogni componente verrà poi definito globalmente attraverso un nome univoco che permetterà di richiamarlo in un modo estremamente semplice.

Ora daremo un’occhiata più da vicino ad alcune caratteristiche interessanti che rendono Vue.js brillante e che gli permettono di guadagnarsi popolarità.

3.1.1 Caratteristiche dei componenti

Ciò che crea le differenze tra i framework è la struttura dei componenti. Un componente è un elemento riutilizzabile fatto su misura con i suoi metodi, logica e stile. I componenti possono comunicare tra loro passandosi i dati (props) o ascoltando l'evento di un altro componente e quindi innescare particolari cambiamenti.

L'immagine riportata in Figura 6, mostra il contenuto all'interno del tag `<script>` di un single file component.



```
1 <template>
2 <div>
3 <div id="app">
4 
5 <h1-{{ msg }}</h1>
6 <input type="text" v-on:input="changeText" autofocus>
7 <span-{{title}}</span>
8 </div>
9
10 <div id="vue-app">
11 <h1>The name is -{{name}}</h1>
12 </div>
13 </div>
14 </template>
15
16 <script>
17 export default {
18   name: 'app',
19   data () {
20     return {
21       title: 'string',
22       msg: 'Welcome to Your Vue.js App'
23     }
24   },
25   methods: {
26     changeText: function () {
27       this.title="Yo"
28     }
29   }
30 }
31 }
32 }
33 }
34 </script>
35
36
```

Figura 6: Tag `<script>` di un file `vue.js`

La figura illustra alcune opzioni di base e alcune delle più usate. Alcune opzioni possono essere molto semplici, come ad esempio "name" per dichiarare il nome da associare al componente, "data" per dichiarare i dati del componente stesso e l'opzione "methods" che è usata per dichiarare i metodi. Vue.js però, oltre alle semplici proprietà descritte in precedenza, mette a disposizione numerose funzionalità native che sono di aiuto per gestire particolari situazioni che si potrebbero presentare. Nelle sezioni successive approfondiremo questi aspetti che non sono del tutto immediati.

3.1.2 Lifecycle di un'istanza Vue

Inizieremo inanzitutto a soffermare la nostra attenzione su come ha inizio la creazione di un oggetto vue. Ora osserveremo il meccanismo con il quale vue permette la costruzioni di singoli componenti che poi verranno iniettati nella vista principale.

La Figura 7 descrive il ciclo di vita di un'istanza vue e di tutti i metodi che la caratterizzano. Vedremo in seguito passo dopo passo come avviene la creazione.

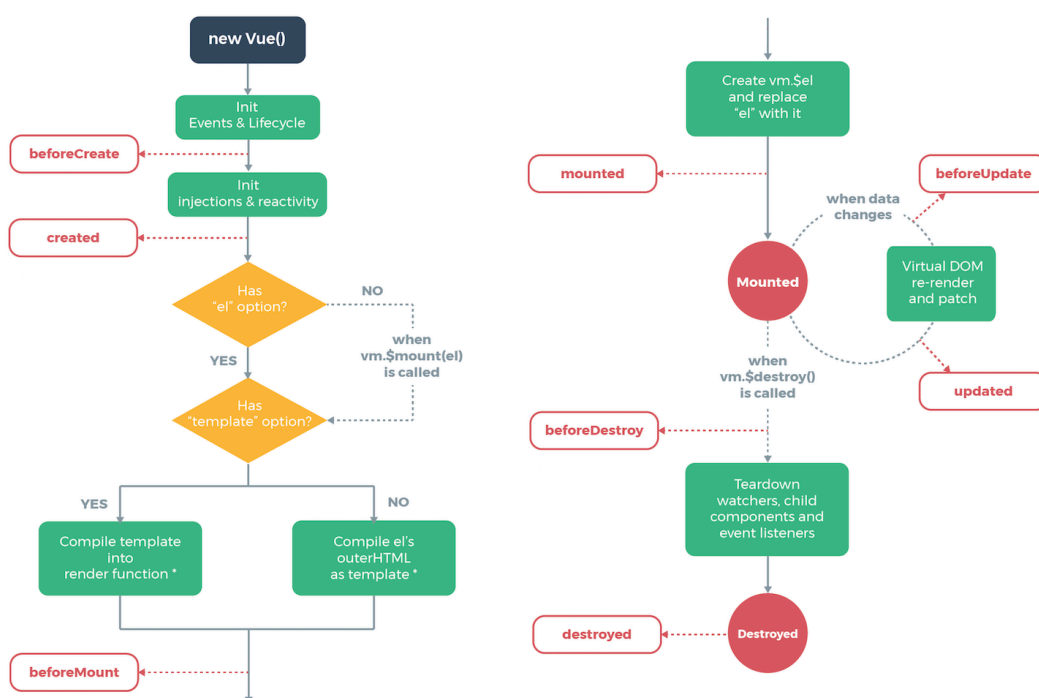


Figura 7: Lifecycle di un istanza Vue

Ogni istanza Vue, come detto in precedenza, segue un ciclo di vita specifico e offre la possibilità di richiamare funzioni javascript in relazione allo stato dell'istanza.

Capitolo 3 Strumenti utilizzati

Di seguito verranno elencate le funzioni caratterizzanti per ogni istanza di vue e il loro funzionamento:

- **beforeCreate**: metodo chiamato in modo sincrono immediatamente dopo l’inizializzazione dell’istanza, prima che i dati inizino ad essere osservati.
- **Created**: metodo chiamato in modo sincrono dopo la creazione dell’istanza. A questo punto l’istanza ha terminato l’elaborazione, il che significa che sono state impostate le seguenti opzioni: osservazione dei dati, metodi, funzioni callback e gli handler degli eventi. Tuttavia, la fase di montaggio non è stata avviata e `$el` (il riferimento all’elemento div del componente) non sarà ancora disponibile.
- **beforeMount**: chiamato poco prima dell’inizio del montaggio
- **Mounted**: chiamato dopo che l’istanza è stata montata, dove l’elemento `$el` viene effettivamente costruito ed è pronto all’utilizzo. La funzione `mounted` però non garantisce che tutti i componenti figli sono stati montati. Se ci si vuole assicurare che l’intera vista sia stata costruita, puoi usare `this.$nextTick` all’interno di `mounted`.
- **beforeUpdate**: chiamato quando i dati cambiano, prima che il DOM venga cambiato. Questo è il modo per accedere al DOM esistente prima di un aggiornamento, ad esempio per rimuovere i listener di eventi aggiunti manualmente.
- **Updated**: chiamato dopo una modifica dei dati, il DOM virtuale viene nuovamente aggiornato e corretto. Qui è possibile eseguire operazioni dipendenti dal DOM. Così come la funzione “`Mounted`”, anche essa non garantisce che tutti i componenti figli siano stati aggiornati.
- **beforeDestroy**: viene chiamato prima che l’istanza di Vue venga distrutta.

3.1.3 La Reattività di Vue

All'interno del tag `<script>` c'è una funzione "data", in cui gli sviluppatori possono definire i dati del componente. L'oggetto restituito da questa funzione diventa poi un oggetto reattivo del componente, il che significa che ogni volta che una proprietà di questo oggetto viene cambiata, il componente attiva il "render" e aggiorna la vista.

In altre parole, quando gli sviluppatori dichiarano dati all'interno della funzione data, il componente converte l'oggetto con i relativi setter e getter. Ogni componente ha il suo watcher per tracciare i cambiamenti dei dati con l'aiuto dei setter e getter.

Quando viene rilevato un cambiamento, il watcher innesca un "re-render", che aggiorna la vista dell'utente con gli ultimi dati.

L'immagine in Figura 8, fornita dal sito ufficiale di Vue.js [3], mostra come funziona il processo di reattività nel componente Vue.js.

Questa caratteristica di reattività permette che la realizzazione del data binding sia facile e controllabile.

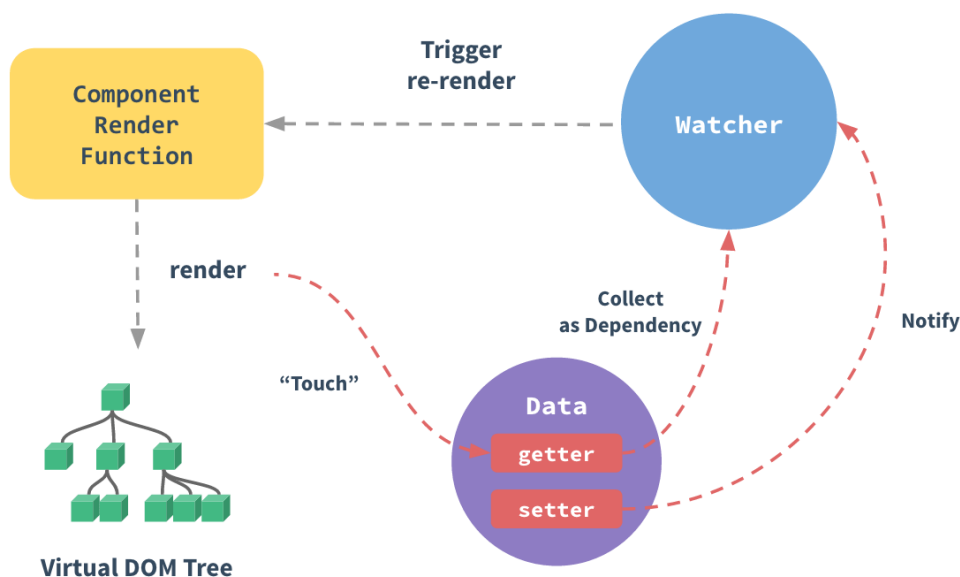


Figura 8: Come vengono gestiti i dati da Vue

3.1.4 Data binding bidirezionale

Come detto in precedenza, il data binding bidirezionale è una caratteristica del framework Vue.js. Questa si è dimostrata molto utile nel legare i dati degli input nelle form attraverso la direttiva v-model.

La Figura 9 mostra un esempio di un data binding bidirezionale in una form utilizzando la direttiva v-model.



Figura 9: Esempio binding bidirezionale

La direttiva v-model lega qualsiasi cosa sia scritta in essa con la proprietà corrispondente nell'oggetto "data". Ciò significa che, ogni volta che scriviamo qualcosa all'interno dell'input, la proprietà "message" dell'oggetto "data" viene automaticamente aggiornata a quel valore e viceversa. Questo processo permette una gestione dello stato dei componenti molto più comoda e meno dispendiosa in termini di tempo.

In questo esempio, abbiamo anche introdotto un nuovo elemento sintattico: le doppie parentesi graffe. La parola "message" all'interno di queste parentesi graffe è sostituita con il valore della corrispondente nell'oggetto "data". Così ogni volta che il valore della proprietà "message" viene cambiato, anche il message viene sostituito con il rispettivo valore.

Nella Figura 9, la parola sotto l'input mostra il suo valore, che risulta da questo processo. Il processo di reattività può essere spiegato così: la direttiva V-model dell'elemento input permette alla proprietà corrispondente nell'oggetto "data" di aggiornarsi con lo stesso valore di input ogni volta che gli utenti digitano qualcosa nell'input. Dopo aver osservato il cambiamento della proprietà nell'oggetto "data", l'osservatore del componente attiva il re-rendering per aggiornare la vista con gli ultimi dati.

3.2 Principali Direttive Vue

In questa sezione descriveremo alcune delle direttive principali che caratterizzano il framework vue. Esse aiuteranno a semplificare il classico meccanismo di implementazione del linguaggio javascript.

- **v-if**: permette la visualizzazione dell'elemento in maniera condizionata. Questa direttiva favorisce un cambiamento dinamico nel momento in cui la variabile cambia e soddisfa la condizione.

Nell'esempio di seguito, se la variabile "awesome" assumerà un valore true, allora il tag <h1> sarà visualizzato.

```
1 <h1 v-if="awesome">Vue is awesome!</h1>
```

- **v-else**: direttiva ammessa solo nel caso in cui sia presente precedentemente una direttiva v-if e permette la sua esecuzione solo nel caso in cui la condizione v-if non viene verificata.

```
1 <h1 v-if="awesome">Vue is awesome!</h1>  
2 <h1 v-else>Oh no </h1>
```

- **v-else-if**: è necessario che l'elemento precedente abbia una direttiva v-if ed è utile per avviare l'esecuzione di una parte di codice qualora l'esecuzione del v-if non sia stata verificata.

```
1 <div v-if="type === 'A'"> A </div>  
2 <div v-else-if="type === 'B'"> B </div>
```

- **v-for**: Esegue il codice dell'elemento contenuto al suo interno più volte in base ai dati d'origine. Il valore della direttiva deve utilizzare una sintassi speciale, così da fornire l'elemento corrente rispetto all'elemento che stiamo iterando

```
1 <div v-for="item in items">  
2   {{ item.text }}  
3 </div>
```

Capitolo 3 Strumenti utilizzati

- **v-on:** Associa un listener di un evento all'elemento. Il tipo di evento è indicato all'interno dell'argomento. Verrà poi specificato il nome della funzione o l'istruzione da attivare nel momento in cui l'evento si verifica. Possono essere gestiti con questa direttiva tutti gli eventi nativi del DOM.

```
1 <button v-on:click="doThis"></button>
```

- **v-bind:** Associa in modo dinamico uno o più attributi o passa un dato a la variabile prop di un componente.

```
1 
2
3 <!-- abbreviazaione -->
4 
5
6 <my-component :prop="someThing"></my-component>
```

Tutte le informazioni riportate in questa sezione sono state derivate consultando parte della documentazione messa a disposizione da vue.js all'interno del sito web [3].

3.3 Vue e Laravel

Il successo di Vue.js è legato alla scelta di Laravel di consigliarlo come framework front-end, che ha portato al rilascio della versione 2.0 del framework (30 settembre 2016), seguita da diverse versioni fino alla 3.2 il 9 Agosto 2021 [2].

Laravel è un framework PHP, basato sulla programmazione orientata agli oggetti e caratterizzato dal pattern architetturale MVC. Grazie ad esso è possibile sviluppare applicazioni sempre più complesse e allo stesso tempo sempre più facili da mantenere.

"La sua storia è iniziata nel 2011, quando Taylor Otwell creò Laravel nel tentativo di fornire un'alternativa più avanzata al framework CodeIgniter [4]". Il progetto è cresciuto nel tempo con numerose versioni di aggiornamento ed è arrivato all'ottava versione.

In effetti, Laravel possiede alcune caratteristiche che sottolineano i suoi punti di forza.

Possiede:

- sistemi di autenticazione facili da usare,
- strumenti da riga di comando integrati (Artisan), che aiutano gli sviluppatori fornendogli file con del codice già strutturato che saranno utili allo sviluppo del applicazione e quindi saranno già pronti per essere estesi. Grazie ad esso, si può creare file di migrazione, un modello contenente le estensioni, un controller già strutturato con i relativi metodi in cui poi lo sviluppatore dovrà semplicemente inserire la parte logica del software.

Laravel ci fornisce anche un ORM(Object-relational mapping) chiamato Eloquent, ossia uno strumento che consente di astrarre le tabelle presenti nel database e trasformarle in modelli per facilitarne l'interazione con le strutture dati utilizzando una sintassi facile ed intuitiva.

In breve, può aiutare gli sviluppatori a interagire con database senza scrivere query molto complesse in SQL, utilizzando solo PHP. VUE può essere integrato in laravel attraverso un processo di configurazione estremamente semplice.

3.4 Database Mysql

MySQL è un sistema per la gestione dei database relazionali gratuito e open source che utilizza un linguaggio SQL. MySQL è scritto in C e C++ ed è compatibile con tutti i principali sistemi operativi. MySQL è stato lanciato nel 1998 ed è il DBMS relazionale open source più popolare e utilizzato al mondo. E' scaricabile gratuitamente e ampiamente utilizzato nelle applicazioni Web.

Il linguaggio di Interrogazione SQL, utilizzato dal DBMS MySQL, è facile da usare, organizza, interroga, manipola ed elimina i dati in modo efficiente (funzioni CRUD) e possiede numerose funzionalità per ricavare dati specifici fra grandi quantità di dati strutturati, in un modo semplice e veloce.

SQL è oggi il principale linguaggio di interrogazione dei database relazionali.

3.5 I componenti

Per la creazione dei componenti vue.js utili alla realizzazione dell'interfaccia ci si è avvalsi dell'utilizzo di alcune librerie, che facilitano lo sviluppo perché ci forniscono una collezione di entità di base pronte per l'uso e quindi favoriscono il riuso di codice, evitando al programmatore di dover riscrivere stesse funzioni, e allo stesso tempo garantiscono un'efficiente manutenzione.

Verranno elencate in questa sezione le diverse librerie di cui si è fatto uso per lo sviluppo dell'intera interfaccia.

3.5.1 Balkan OrgChartJs

La prima libreria di cui tratteremo è la Balkan OrgChartJs che consente agli sviluppatori di creare organigrammi semplici, flessibili e personalizzabili per presentare i dati strutturali in modo chiaro ed elegante, e ha permesso la realizzazione del componente che è stato concepito con una struttura ad albero (Diagrams), per permettere la rappresentazione dello stato e dei messaggi degli indicatori di prestazione (KPI).

OrgChart JS può essere eseguito su qualsiasi server che supporti HTML, non dipende da alcuna libreria JavaScript di terze parti, funziona in tutti i browser moderni e utilizza SVG per il rendering grafico.

Per la creazione di un organigramma basterà creare un oggetto orgChart in cui verranno specificati alcuni attributi, ad esempio “nodes” dove verranno inseriti l'insieme dei dati come: “id”(obbligatorio), “pid” ossia l'id genitore che rappresenta la connessione tra due nodi, il nome e tutte le informazioni supplementari.

L'attributo **nodeBinding** servirà per permettere la visualizzazione dei dati sull'interfaccia dell'albero tramite la specifica esplicita fra gli attributi 'field_0..n'(espressi in ordine crescente) e i dati contenuti all'interno del nodo, in caso contrario verranno visualizzati in una finestra popup quando si effettua un click sul nodo.

Riassumendo esistono due tipi di campi: campi del nodo e i campi visualizzati. È possibile definire un numero illimitato di campi, i quali possono essere sovrascritti all'interno dell'attributo **nodeBinding** per permettere la visualizzazione sul nodo stesso, senza necessità di una finestra popup.

Capitolo 3 Strumenti utilizzati

La libreria OrgChart JS possiede una serie di modelli predefiniti, infatti è possibile modificare il modello impostando l'attributo **template** tra quelli presenti (ana, ula, olivia, belinda, rony, mery, polina, mila, diva, base, isla, deborah).

È possibile creare un modello personalizzato: semplicemente ciò che bisogna fare è ereditare uno dei modelli predefiniti e sovrascriverne alcune proprietà. Sarà possibile modificare l'aspetto dei componenti che caratterizzano l'organigramma come il Nodo, i Campi, l'Immagine, i Pulsanti Espandi/Comprimi, così come le relative dimensioni, forme e colori.

Inoltre, si può utilizzare un CSS personalizzato per sovrascrivere o aggiungere nuove proprietà agli stili del grafico, attraverso alcuni selettori CSS presenti all'interno della documentazione.

L'immagine Figura 10 è una demo del componente che evidenzia la grande potenzialità e la flessibilità che possiede questa libreria.

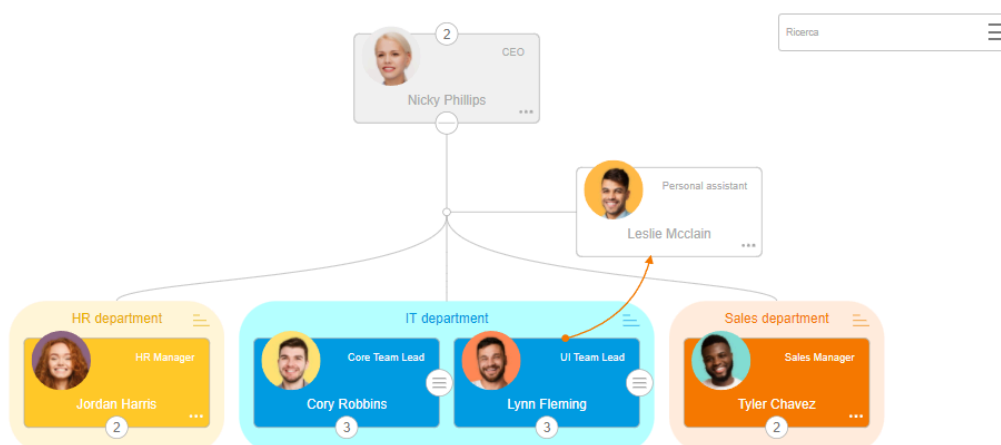


Figura 10: Versione demo dell'albero della libreria Balkan OrgChartJs

Al fine di una corretta realizzazione del componente ci si è avvalsi della sua documentazione reperibile sul sito web:

<https://balkan.app/OrgChartJS/Docs/GettingStarted>

3.5.2 Fusion Chart

La libreria FusionCharts permette di creare grafici animati e interattivi per le applicazioni web.

Tra le tante funzionalità presenti in questa libreria, Fusion Chart consente agli sviluppatori di creare degli indicatori angolari simili agli speedometers, con i quali è possibile visualizzare un dato specifico, utilizzando un quadrante su scala radiale con limiti ben definiti. È possibile associare colori e valori alle sezioni così da limitare intervalli specifici al fine di permettere una rapida considerazione sullo stato del sistema, ad esempio verde per soddisfacente, giallo per cautela e rosso per allarme.

Fusion Chart permette una personalizzazione puntuale degli aspetti grafici: Didascalia, Sottotitolo, Area del grafico, Limiti inferiore e superiore, il fattore di scala, l'indicatore, la visualizzazione dei dati.

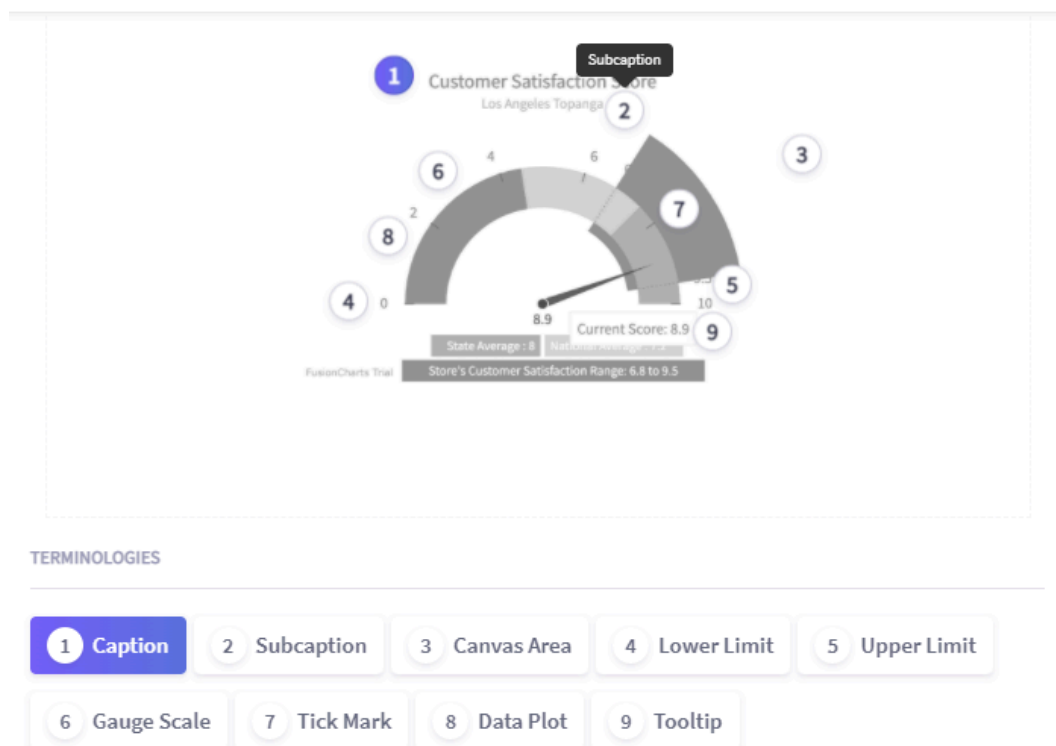


Figura 11: Sezioni di un indicatore angolare della libreria fusionChart

Capitolo 3 Strumenti utilizzati

La Figura 11 mostra le grandi potenzialità e funzionalità della libreria. È possibile accedere rapidamente alle sezioni del grafico che vogliamo personalizzare con esempi chiari ed espliciti delle modifiche che si possono realizzare sovrascrivendo alcuni attributi in fase di implementazione del componente.

Come anticipato, un oggetto `fusionChar` visualizza i valori dei dati su una scala radiale. La scala radiale è contrassegnata da un limite inferiore e da un limite superiore, ovvero i valori di massimo e minimo che possono essere tracciati. All'interno si possono creare varie sezioni per classificare i dati: ogni sezione può avere un colore di sfondo, un colore del bordo, ecc. Per un corretta visualizzazione sono presenti gli indicatori che vengono utilizzati per puntare al valore desiderato sulla scala radiale. È possibile puntare a più di un valore utilizzando più indicatori. Ogni indicatore può avere le sue proprietà individuali come: colore, larghezza, raggio e valore.

Tutti gli attributi riguardanti le caratteristiche grafiche saranno contenute in un attributo `dataSource`: sarà possibile specificare il linguaggio con il quale sono formattati i dati tramite l'attributo `dataFormat` (che nel nostro caso saranno `json`) e di specificare gli eventi ai quali il componente deve rispondere.

Tutte le informazioni relative alla personalizzazione e all'utilizzo di questa libreria sono estremamente illustrate all'interno delle sua documentazione che è reperibile sul sito:

<https://www.fusioncharts.com/dev/chart-guide/gauges-and-widgets/angular-gauge>

Capitolo 3 Strumenti utilizzati



Figura 12: Esempio di indicatore angolare

Nella Figura 12 viene rappresentata una delle possibili configurazioni di questo componente, in cui sono stati personalizzati alcuni aspetti grafici. La documentazione presente al sito web sopracitato illustra le innumerevoli configurazioni che si possono realizzare in fase di implementazione.

3.5.3 Highcharts

La libreria Highcharts è una moderna libreria per le rappresentazioni grafiche basata su SVG. Semplifica l'aggiunta di grafici interattivi a progetti web. È molto utilizzata e rimane un'ottima soluzione per lo sviluppo di grafici grazie al suo solido set di funzionalità, alla sua facilità d'uso e alla documentazione completa.

Per analizzare il funzionamento di Highcharts, bisogna comprendere le varie parti e i concetti che caratterizzano un grafico.

La Figura 13 illustrerà le parti principali che compongono il grafico.

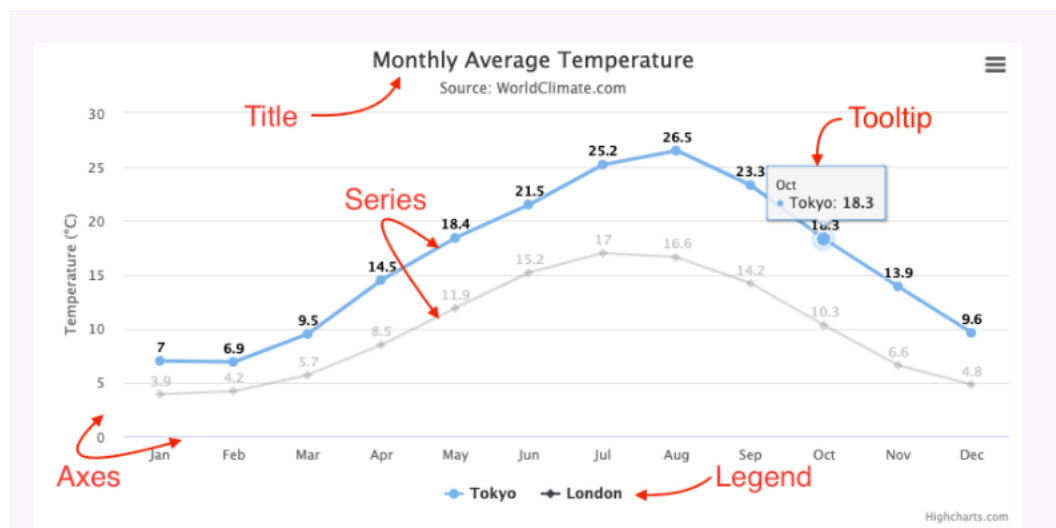


Figura 13: Sezioni che compongono un grafico della libreria highChart

Come si può evincere dalla figura, il grafico è suddiviso in più sezioni.

La sezione definita come "Title" rappresenta il testo che descrive il grafico, che di solito si trova nella parte superiore, dando la possibilità di fornire in aggiunta un sottotitolo per ulteriori informazioni.

La sezione "Series" rappresenta la vera e propria mappatura dei dati che vogliamo rappresentare, con la possibilità di inserimento di una o più serie di dati da visualizzare all'interno grafico.

Inoltre, sono presenti descrizioni puntuali per ogni dato rappresentato, che vengono visualizzate al passaggio del mouse su un punto del grafico, attraverso una piccola finestra di visualizzazione che descrive i valori di quel particolare punto.

Capitolo 3 Strumenti utilizzati

Nella parte inferiore verrà presentata la legenda che mostra le serie di dati contenuti nel grafico e consente di abilitare e disabilitare la visualizzazione di una o più serie.

Gli assi possiedono un etichetta al fine di specificare il tipo di dato che viene graficato.

E' possibile inoltre personalizzare la visualizzazione con l'utilizzo di animazioni che rendono il grafico più accattivante e interessante, migliorando quindi la User Experience.

Capitolo 4

Applicazione sviluppata

In questa sezione verranno descritti tutti i sei componenti che formano l'intera interfaccia e verrà descritto il modo in cui sono stati progettati.

4.1 Controller

Il primo componente che prenderemo in considerazione è il Controller, quella parte dell'interfaccia che si occupa della gestione dei sistemi, ossia la sezione che avrà il compito di avviare o spegnere l'intero processo e che darà la possibilità di selezionare il relativo "Step" che si vuole utilizzare.

La Figura 14 mostra il layout del Controller.

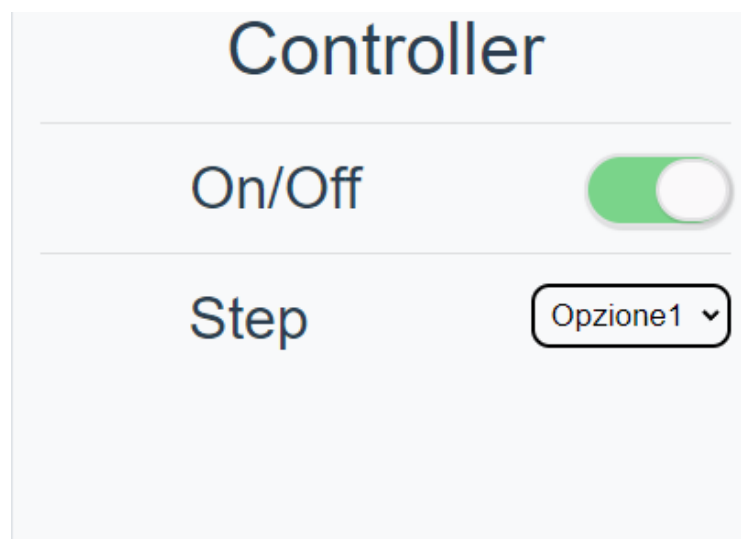


Figura 14: Layout del controller dell'interfaccia

Come si evince dalla figura, il controller ha due funzionalità, di cui tratteremo.

4.1.1 On/Off

Il pulsante On/Off è stato implementato con la logica del "toggle button", ossia un bottone di controllo all'interno di un'interfaccia grafica, che quando viene cliccato cambia il suo stato, da una condizione all'altra e viceversa, esempio verde acceso e bianco spento.

```
<h2>On/Off
  <div class="move">
    <div class="wrapper" :class="{ 'active-wrapper': status}" @click="changeState">
      <div class="toggle" :class="{ 'active-toggle': status}"/>
    </div>
```

Figura 15: Codice html per la realizzazione dell'On/Off

Il pulsante è realizzato attraverso l'utilizzo di due tag div (vedi Figura 15), uno contenitore, contrassegnato con l'attributo class "wrapper" e un secondo div con attributo class "toggle". Entrambi presentano delle istruzioni condizionali per il cambiamento del proprio attributo class in relazione allo stato in cui si trova il sistema: quando cambia il valore della variabile "status" a sua volta cambierà l'attributo class, e di conseguenza il css per la visualizzazione.

Il cambiamento della variabile status da "true" a "false" avverrà grazie al listener dell'evento click, il quale richiamerà la funzione changeState, definita nella sezione methods, all'interno dell'tag <script>.

Capitolo 4 Applicazione sviluppata

I colori, le forme e le animazioni del bottone vengono racchiuse nel css del componente stesso. In Figura 16 sono riportate le direttive utilizzate per la rappresentazione.

```
.wrapper,
.toggle {
  box-shadow: 1px 1px 1px 1px rgba(0, 0, 0, 0.1);
  transition: all 300ms ease;
}
.wrapper {
  width: 80px;
  height: 40px;
  display: flex;
  align-items: center;
  background: #faf9fa;
  border: 2px #e3e2e4 solid;
  border-radius: 50px;
}
.active-wrapper {
  background: #7ad48a;
}
.toggle {
  position: relative;
  width: 40px;
  height: 34px;
  background: #faf9fa;
  border: 2px #e3e2e4 solid;
  border-radius: 50px;
  margin: 3px;
}
.active-toggle {
  transform: translateX(85%);
}
```

Figura 16: Regole css del pulsante On/Off

Quando la variabile status assumerà un valore “true” verranno aggiunte le regole dei selettori css `active_wrapper` e `active_toggle` le quali aggiungeranno un colore di sfondo verde all’interruttore per rappresentare lo stato di accensione e verrà traslato sul lato sinistro il bottone ovale rappresentato dalla classe `toggle`.

4.1.2 Step

Il pulsante Step permette, con l'aiuto di un menù a tendina, la selezione di una delle opzioni tra quelle definite. L'implementazione risulta notevolmente semplice tramite l'utilizzo del tag `<select>` nativo del linguaggio HTML (Figura 17).

```
<select :value="'Opzione1'" @change="selection($event.target.value)">
  <option :value="option" :key="option" v-for="option in this.options">
    {{ option }}
  </option>
</select>
```

Figura 17: Codice html per la realizzazione della select

All'avvio del componente verranno caricate all'interno della select tutte le opzioni selezionabili e tramite l'aiuto della direttiva `v-for` che caratterizza il framework Vue, si semplifica notevolmente la sua implementazione. Verrà inoltre associato un eventListener di tipo "change", che avvierà la funzione `selection`, alla quale verrà passato il valore "value" dell'opzione selezionata.

Il css definirà poi le forme, i colori e le dimensioni da associare al tag di selezione attraverso l'utilizzo della keyword `select`(Figura 18).

```
select{
  border: 2px solid □black;;
  outline: □black;
  background: transparent;
  border-radius: 10px;
  margin: 0;
  display: block;
  padding: 5px 5px 5px 5px;
  font-size: 17px;
  color: □black;
}
```

Figura 18: Regole css del pulsante di selezione

Definiremo un bordo(2px) e il suo relativo raggio di curvatura, un colore di sfondo, le dimensioni del padding, il colore e la dimensione dei caratteri.

4.2 Subsystems

Il secondo componente di cui discuteremo è Subsystems che si occuperà della gestione dei sistemi:

FCU0, WIN0, SHA0, Boiler, Chiller, Summer con la possibilità di settare la modalità "show", che consentirà solamente la visualizzazione e la modalità "set", che permetterà la visualizzazione e la modifica dello stato dei sistemi citati in precedenza.

In Figura 19 viene mostrato il componente nella sua integrità.

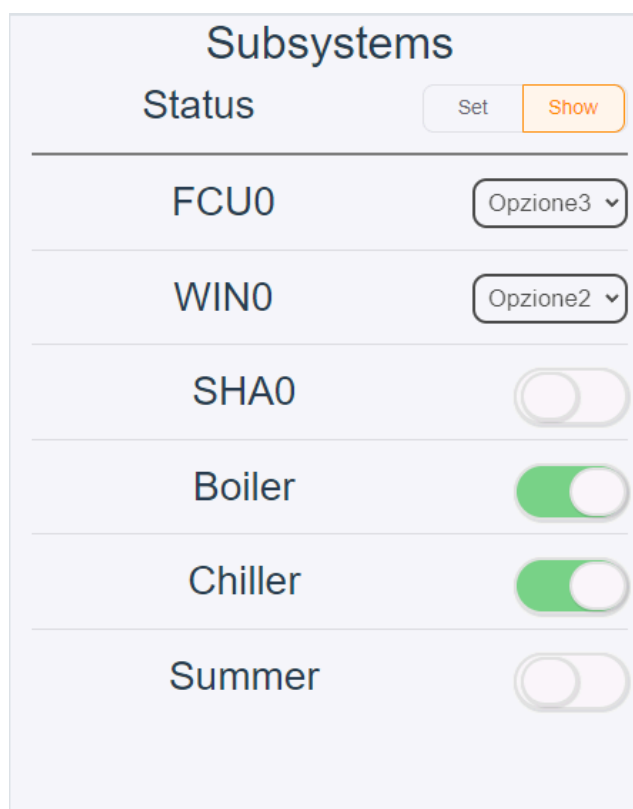


Figura 19: Layout del componente subsystems

Vedremo ora nello specifico come sono implementati i singoli elementi.

4.2.1 Status

Il pulsante Status sarà quello che si occuperà di attivare e disattivare gli elementi presenti nella parte inferiore.

La logica utilizzata per la realizzazione è mostrata in Figura 20.

```
<h3>Status
  <div class="move">
<div class="btn-group">
  <span
    :class="{ 'unlock-btn': status === false }"
    @click="chooseStatus(false)"
    class="status-btn"
  >Set</span>
  <span
    :class="{ 'lock-btn': status === true }"
    @click="chooseStatus(true)"
    class="status-btn"
  >Show</span>
</div>
</div>
</h3>
```

Figura 20: Codice html del pulsante status

Lo sviluppo del bottone è avvenuto attraverso l'utilizzo di un tag `<div>` contenente due tag figli ``. Al tag padre viene assegnato l'attributo `class btn-group`, che verrà definito nel foglio di stile per determinare il layout generale del bottone. Per i tag figli la logica è leggermente differente: l'attributo `class` è condizionato dal valore della variabile `status`, la quale può assumere solo due valori `true` o `false`. Nel caso in cui il valore della variabile fosse `true` allora al corrispettivo figlio verrà assegnato l'attributo `class unlock -btn`, nel caso contrario il valore `lock-btn`. Grazie a questo meccanismo sarà possibile realizzare un cambiamento della grafica di visualizzazione così da rendere esplicito e comprensibile in quale modalità si sta visualizzando il sistema e a quali funzionalità è possibile accedere. Il click all'interno di una delle due sezioni delimitate dal tag `span` richiamerà la funzione `chooseStatus`, la quale cambierà il valore della variabile **"status"** (variabile che ritroveremo in seguito poichè artefice della disabilitazione degli elementi sottostanti). Più avanti vedremo come è stata possibile la realizzazione di questo meccanismo. Per ora ci concentreremo sulla parte grafica dell'elemento.

Capitolo 4 Applicazione sviluppata

La Figura 21 riporta le regole css caratterizzanti dell'oggetto che stiamo considerando.

```
.unlock-btn {
  background: ■ rgba(255, 246, 236, 1);
  border-radius: 8px 0px 0px 8px;
  position: relative;
  color: ■ rgba(255, 140, 8, 1);
  &:after {
    content: "";
    display: block;
    position: absolute;
    top: -1px;
    bottom: -1px;
    left: -1px;
    right: -1px;
    border-radius: 8px 0px 0px 8px;
    border: 1px solid ■ rgba(255, 140, 8, 1);
  }
}
```

Figura 21: Regole css per la realizzazione del pulsante status

La (Figura 21) riporta le regole css Soffermeremo l'attenzione su `&:after`, un selettore che permette l'inserimento di qualcosa dopo il contenuto di ogni elemento selezionato. La proprietà "content" è richiesta e serve per specificare il contenuto da inserire; è possibile inserire in aggiunta il selettore `&:before` per inserire qualcosa prima del contenuto.

Questa parte di codice sarà essenziale per la visualizzazione marcata della modalità che è stata selezionata SET o SHOW. Nella Figura 21 sono presenti solo le regole css di uno dei due bottoni, ossia quello con l'attributo class "unlock-btn"; cosa analoga avviene per l'altro elemento(lock-btn), a meno di piccoli cambiamenti sull'arrotondamento del bordo al fine di una visualizzazione conforme al layout del pulsante.

4.2.2 FCU0-WIN0

In questa sezione definiremo il funzionamento degli elementi relativi ai sistemi FCU0 e WIN0.

La Figura 22 riporta l'implementazione di solo uno dei due, considerata la loro analogia.

```
<h3>{{this.subsystems[0].name}}
<div class="move">
  <select :value="options[this.subsystems[0].status]" :disabled="!this.status" @change="selectionFCU0($event.target.selectedIndex)">
    <option :value="option" :key="option" v-for="option in options">
      {{ option }}
    </option>
  </select>
</div>
</h3>
```

Figura 22: Codice per la realizzazione della select del sistema FCU0

Per la realizzazione ci si è avvalsi di una classica select che abbiamo già avuto l'occasione di discuterne nella sottosezione 4.1.2 dedicata allo "Step".

Le opzioni preselezionabili sono definite nella funzione "data" del componente, che grazie all'utilizzo del v-for(for implementato all'interno di vue), scandisce le opzioni selezionabili e le visualizza in un menu a tendina. La particolarità di questo elemento rispetto al pulsante "Step" definito in precedenza è l'attributo "disabled", che è condizionato dalla variabile status di cui abbiamo già parlato nella sottosezione 4.2.1.

Grazie alla potenzialità del framework vue la variabile "status" resterà osservata per tutto il ciclo di vita della schermata. Questo significa che, quando verrà modificato il valore della variabile status, automaticamente il cambiamento ripercuoterà sul attributo disabled attivando e disattivando l'elemento. Qualora l'elemento si trovasse in una situazione di disabled=false, al cambiamento di una delle opzioni presenti all'interno delle select verrà richiamata la rispettiva funzione a cui verrà passato indice dell'elemento selezionato tramite l'utilizzo dell'oggetto \$event.target.selectedIndex e all'interno di essa sarà implementato il codice da eseguire in relazione alla scelta effettuata.

Per le regole css relative alla visualizzazione di questo elemento si può far riferimento alla Figura 18, in quanto gli elementi FCU0/WIN0 e Step condividono la stessa grafica. Sebbene condividano le stesse regole css, andranno reinserite nella sezione <style> dato che ogni componente è indipendente e non condivide il proprio foglio di stile con gli altri.

4.2.3 SHA0-Boiler-Chiller-Summer

Ora definiremo come sono stati realizzati i bottoni di stato dei sistemi: SHA0, Boiler, Chiller e Summer. Per la rappresentazione si è utilizzato un toggle button, considerata l'esigenza di dover rappresentare due condizioni: attivo/inattivo.

Considerata la similitudine fra i quattro elementi ne analizzeremo uno campione e tutte le considerazioni fatte saranno automaticamente estendibili agli altri.

La Figura 23 mostra l'implementazione per la realizzazione dell'elemento Boiler.

```
<h3>{{this.subsystems[3].name}}
  <div class="move">
    <div class="wrapper" :class="{ 'active-wrapper': statusboiler}" @click="changeBoiler">
      <div class="toggle" :class="{ 'active-toggle': statusboiler}"/>
    </div>
  </div>
</h3>
```

Figura 23: Codice html del sistema Boiler

La creazione è avvenuta in maniera analoga all'interruttore On/Off presente all'interno del controller, per cui è possibile far riferimento alla sottosezione 4.1.1 per comprendere meglio lo sviluppo.

Soffermeremo l'attenzione su come gli elementi passino dalla modalità show alla modalità set. Nella Figura 23 è possibile vedere come l'intero componente sia stato realizzato tramite l'utilizzo di due tag <div>, gli stessi non presentano fra i loro attributi predefiniti l'attributo disabled, non essendo uno strumento nativo del linguaggio per la scelta di una opzione. Considerato quanto detto, il controllo sullo stato della variabile "status" è avvenuto all'interno della funzione changeBoiler, la quale prima di cambiare il valore della variabile status e di conseguenza mettere in esecuzione il codice che dovrà essere avviato al click sul pulsante, effettuerà la verifica della modalità di visualizzazione dell'intero componente(Figura 24).

```
},
changeBoiler(){
  if(this.status==true){
    this.statusboiler=!this.statusboiler;
  }
},
```

Figura 24: Regole css per la realizzazione del pulsante status

Capitolo 4 Applicazione sviluppata

Quando avverrà la costruzione del componente, di default verranno visualizzati gli stati in cui i sistemi si trovano in quel determinato momento, e si attiverà la modalità "show" per cui non sarà possibile apportare modifiche.

Tutti i dati caratterizzanti di questo componente sono contenuti all'interno della tabella "subsystem" del database MySQL. Attraverso l'aiuto di Laravel e dell'ORM che ci viene messo a disposizione, sono stati realizzati i modelli per l'astrazione dei dati dal database.

LA struttura della tabella "subsystem" è mostrata in Figura 32).

subsysld	description	name	status
1	testo descrizione	FCU0	2
2	testo descrizione	WIN0	1
3	testo descrizione	SHA0	0
4	testo descrizione	Boiler	1
5	testo descrizione	Chiller	1
6	testo descrizione	Summer	0

Figura 25: Tabella subsystem del database

La tabella sarà è formata da:

- un **id** (univoco) che sarà la chiave primaria,
- una **descrizione**,
- suo relativo **il nome del sottosistema**,
- lo **stato** in cui si trova.

Come accennato in precedenza, sarà poi il nostro model che astrarrà i dati dalla tabella che verranno acquisiti dal componente per la costruzione dell'interfaccia.

Capitolo 4 Applicazione sviluppata

Il model conterrà una sola funzione `getAllSubsystem`, la quale con una semplice query scritta con la sintassi semplificata di Eloquent permetterà di recuperare i dati dal database(Figura 26).

```
class subsystems {  
  
    public function getAllSubsystem() {  
        $subsystem = subsystem::get();  
        return $subsystem;  
    }  
}
```

Figura 26: Model del componente subsystems

Per recuperare i dati il componente si avvale dell'utilizzo di una rotta. Le rotte rappresentano un indirizzo che determina cosa deve accadere quando vengono richiamate attraverso una richiesta http.

Nel nostro caso verrà definita nella rotta la funzione da richiamare dell'Controller, che a sua volta avrà un riferimento alla variabile model della tabella subsystems e tramite la funzione `getAllSubsystem()` riacquisirà i dati.

Per semplificare la gestione delle rotte e quindi delle chiamate ajax asincrone per il recupero dei dati, ci si è avvalsi dell'utilizzo della libreria axios.

AXIOS Lo scopo della libreria AXIOS è quello di semplificare la definizione di richieste AJAX (richieste asincrone); è facile da utilizzare ed è una libreria molto piccola e leggera. Può effettuare le richieste get, post, put, delete ecc. Possiede due principali metodi, il metodo `then()` che viene avviato in caso di successo, ossia quando sono stati correttamente acquisiti i dati della richiesta, e il metodo `catch()` che viene avviato in caso di un'eccezione.

Il metodo `get()` prevede due parametri, il primo è l'URL a cui verrà effettuata la richiesta AJAX e il secondo è facoltativo, ovvero i dati che devono essere inviati.

Vediamo in seguito come i dati vengono acquisiti (Figura 27).

```
data() {
  return {
    subsystems: [],
    statussho: "",
    statusboiler: "",
    statuschiller: "",
    statussummer: "",
    status: false,
    options: ["Opzione1", "Opzione2", "Opzione3", "Opzione4"],
  };
},
mounted(){
  axios.get('./subsystem').then((response) => {
    // handle success
    console.log(response.data);
    this.subsystems = response.data;
    this.statussho = response.data[2].status;
    this.statusboiler = response.data[3].status;
    this.statuschiller = response.data[4].status;
    this.statussummer = response.data[5].status;
  })
},
}
```

Figura 27: Codice per l'acquisizione dei dati

Come si evince dalla Figura 27, l'acquisizione dei dati avviene all'interno della funzione `mounted`, che ci assicura che tutti le parti del componente siano state create. Se la richiesta fatta attraverso `axios` avrà esito positivo, e questo ci viene assicurato dalla funzione `then()`, i dati definiti all'interno dell'attributo "data" saranno sostituiti dai dati presenti nel database. Questi, grazie al meccanismo offerto da `vue`, verranno osservati e tutto il componente risponderà alle modifiche delle variabili avvenute in questa fase.

4.3 Diagrams

Il terzo componente che verrà descritto è **Diagrams**, caratterizzato da una struttura ad albero, che sarà utile per una chiara e semplice visualizzazione del modello di gestione del benessere ambientale attraverso i KPI.

Per lo sviluppo di questo componente si è ricorso all'utilizzo della libreria *Balkan OrgChartJs* di cui abbiamo parlato nella sottosezione 3.5.1

La Figura 28 illustra un esempio di applicazione del componente.

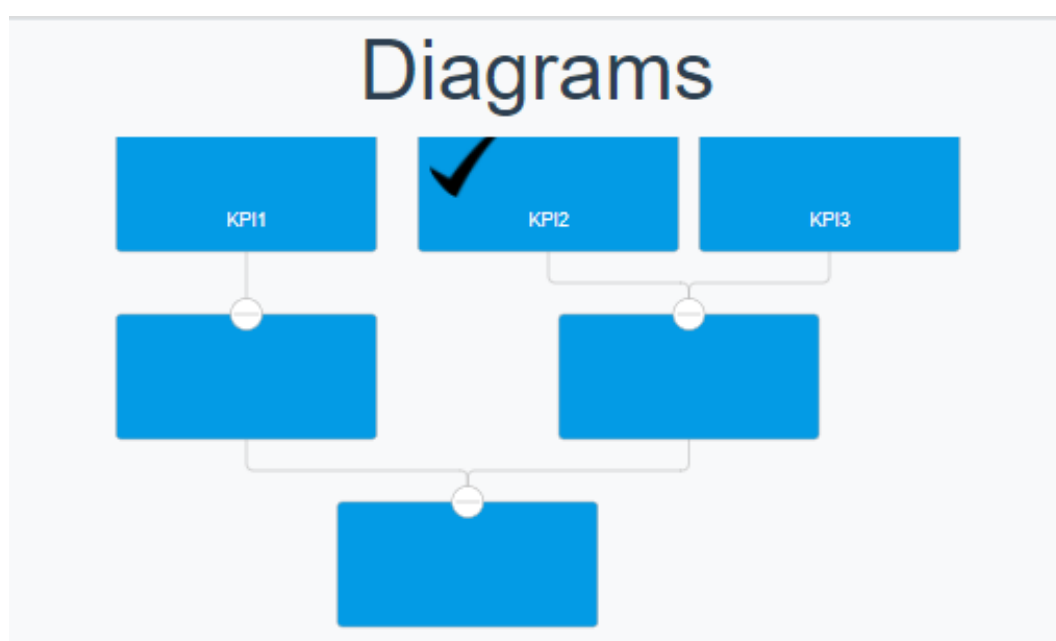


Figura 28: layout componente diagrams

L'intero componente è stato pensato come un albero rovesciato, costituito da 6 nodi.

I nodi genitori sono rappresentati dai tre rispettivi KPI1, KPI2 e KPI3. Il primo (KPI1) possiede un proprio nodo figlio, gli altri due (KPI2, KPI3) ne condividono uno, si ricongiungono nell'ultimo nodo presente nell'albero.

L'implementazione avviene tramite la costruzione di un'istanza di *OrgChart*, oggetto definito all'interno della libreria che all'avvio viene importata.

La libreria contiene numerosissimi attributi personalizzabili. Ora tratteremo quelli utilizzati per lo sviluppo(Figura 29).

```
computed: {
  getChart() {
    return {
      chart: new OrgChart(document.getElementById("orgchart"), {
        enableSearch: false,
        mouseScroll: OrgChart.action.none,
        enableKeyNavigation: false,
        min : true,
        scaleInitial : 0.5,
        orientation: OrgChart.orientation.bottom,
        nodeBinding: {
          field_0: "nome",
          img_0: "img"
        },
        nodes: this.nodi,
      })
    };
  }
}
```

Figura 29: Attributo chart dell'albero

Considereremo innanzitutto l'attributo "orientation", che permette di invertire la direzione dell'albero tramite l'utilizzo delle parole chiave: top, botton, right e left. E' possibile quindi scegliere il verso di orientamento dell'albero.

La libreria possiede una barra di ricerca già implementata per la ricerca testuale all'interno dei nodi: questa funzionalità non è nel nostro caso necessaria, pertanto setteremo enableSearch a "false". Imposteremo mouseScroll a "none" per non permettere l'ingrandimento e il rimpicciolimento del componente e allo stesso modo verrà impostato enebleKeyNavigation a "false" per non tener traccia dell'ultimo nodo che è stato selezionato.

L'attributo "nodeBinding" associa i dati del nodo ai parametri del modello dello stesso; nell'esempio presente nella Figura 29 il campo "nome" viene assegnato a field_0 del modello, così come avviene per il campo img. Sarà possibile definire n campi field_0..n, permettendo la visualizzazione sul nodo.

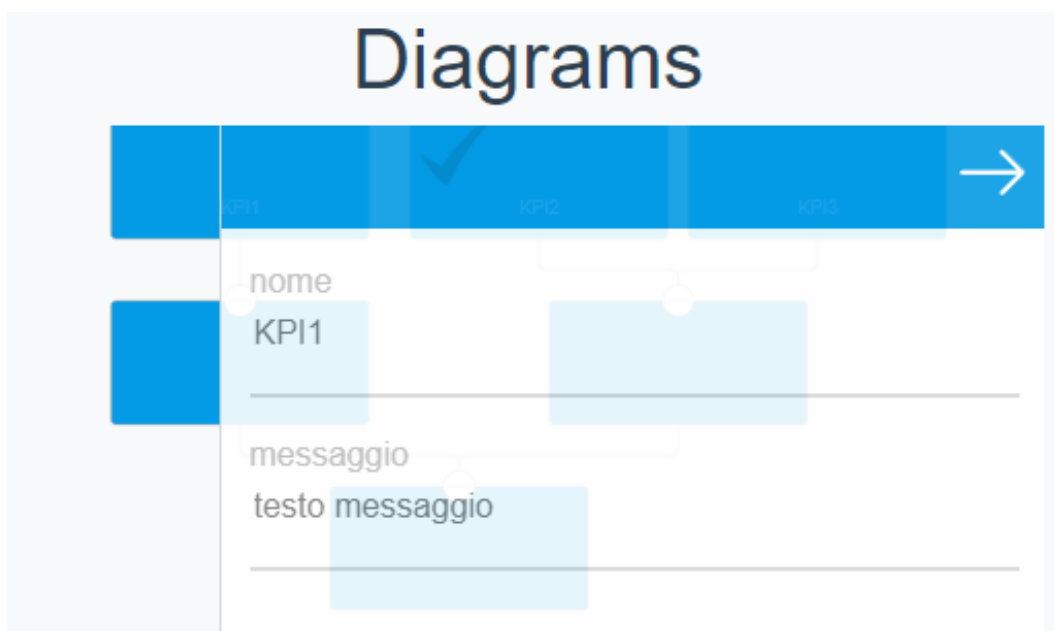


Figura 30: Finestra popup per la visualizzazione di dati aggiuntivi

Tutti gli altri parametri che non saranno definiti all'interno dell'attributo "nodeBinding" automaticamente verranno visualizzati in una finestra popup, che comparirà nel caso in cui l'utente effettui un click all'interno di uno dei 6 nodi(Figura 30). Nel caso mostrato in figura viene visualizzato il campo messaggio, che però non era presente sull'interfaccia principale.

La costruzione dell'albero è resa possibile associando a l'istanza di *OrgChart* i dati che si vogliono rappresentare, effettueremo perciò un'associazione tra l'attributo "nodes" e la variabile "nodi" da noi definita, la quale ospiterà i dati recuperati dal database che però dovranno essere strutturati con una forma ben specifica(Figura 31)

```

mounted() {
  axios.get('./albero').then((response) => {
    let idModified = response.data.map(
      obj => {
        var img="";
        if(obj.status==1){
          img="https://png2.cleantpng.com/sh/65c948fd340b17185e07f25712a7e392/L0KzQYm3VsA0W51xR91yc4Pzfr10gBhma5wyhdN7az32ib7pjBwua51ui39qcoSwc";
        }
        return {
          "id" : obj.nodeId,
          "pid":obj. parentId,
          "img": img,
          "messaggio":obj.description,
          "nome":obj.name,
        }
      }
    );
    this.nodi = idModified;
    this.getChart();
  })
},

```

Figura 31: Recupero dati dal database

Capitolo 4 Applicazione sviluppata

Il recupero dei dati avviene tramite l'utilizzo di axios (di cui abbiamo parlato nella sezione 4.2), il quale effettuerà una chiamata alla rotta “/albero”, che avvierà una funzione del controller che a sua volta si interfacerà con il relativo model per l'acquisizione dei dati dal database. Successivamente, grazie all'utilizzo della funzione "map" nativa nel linguaggio javascript, verranno rimappati i dati in un formato json, nel quale verranno specificati il campo “id” identificativo del nodo, il campo “pid”(parent id) che specifica il genitore del nodo preso in considerazione e l'attributo “img”. Quest'ultimo ci permetterà, tramite l'utilizzo di un'istruzione condizionale, la visualizzazione a livello di interfaccia di una spunta che rappresenterà quale dei KPI ha il suo stato settato a 1. In aggiunta, saranno presenti gli attributi “nome” e “messaggio” che descrivono rispettivamente il nome associato al nodo e la sua descrizione.

Successivamente avverrà l'assegnazione alla variabile “nodi” precedentemente dichiarata nella sezione “data”, la quale è collegata dinamicamente all'attributo "nodes" dell'oggetto OrgChart. Infine verrà richiamata la funzione getChart() per la creazione (vedi Figura 31).

La costruzione avviene tramite l'utilizzo di SVG (Scalable Vector Graphics), una tecnologia in grado di visualizzare oggetti di grafica vettoriale. La realizzazione della grafica avviene mediante un insieme di figure geometriche che definiscono punti, linee, curve e poligoni ai quali possono essere attribuiti colori e sfumature.

Capitolo 4 Applicazione sviluppata

I dati costituenti del componente sono memorizzati in una tabella sql all'interno del database, e sono strutturati come riportato in Figura 32.

nodeId	parentId	name	description	status
1	0		testo messaggio	0
2	1		testo messaggio	0
3	1		testo messaggio	0
4	2	KPI1	testo messaggio	0
5	3	KPI2	testo messaggio	1
6	3	KPI3	testo messaggio	0

Figura 32: Tabella diagrams del database

- La proprietà “**nodeId**” rappresenterà univocamente il nodo e a sua volta sarà la chiave primaria della tabella.
- Il “**parentId**” servirà per definire il rapporto di parentela con gli altri nodi, ovvero sarà poi l'attributo pid che abbiamo definito precedentemente(Figura 31).
- Il “**name**”, colui che definirà il nome stesso del nodo, e come si evince dalla figura potrà assumere il valore nullo.
- La “**description**” ovvero la descrizione esplicativa del nodo.
- Come ultimo la proprietà “**status**” che servirà a identificare quale tra i sei nodi presenti è in uno stato attivo.

Capitolo 4 Applicazione sviluppata

I dati vengono estratti, come già visto per gli altri componenti, tramite l'utilizzo di Eloquent.

Nella Figura 33 possiamo vedere il modo in cui è strutturato il model.

```
use App\Models\Resources\diagram;  
  
class diagrams {  
  
    public function getAllNodes() {  
        $nodes = diagram::get();  
        return $nodes;  
    }  
}
```

Figura 33: Model del componente diagrams

Il model quindi mapperà la tabella, grazie alle direttive presenti nel file "diagram" contenuto nella cartella Resources (debitamente importata), e attraverso una semplice chiamata `get()` recupereremo i dati da iniettare.

Possiamo notare quanto sia espressiva e semplice la sintattica che caratterizza Eloquent, facilitando notevolmente il recupero dei dati rispetto alle classiche query caratterizzanti del linguaggio sql.

4.4 KPI

Il quarto componente preso in considerazione, gestirà la visualizzazione del valore di comfort attuale e atteso dal KPI selezionato. Il componente sarà quindi formato da una select con la quale si potrà scegliere l'elemento che si intende visualizzare. Dopo la selezione, istantaneamente, verranno rappresentati sugli "speedometers" i valori del corrispettivo KPI, tramite gli indicatori che modificheranno i loro valori nel tempo.

La Figura 34 mostra come il componente si presenta all'interno dell'interfaccia.

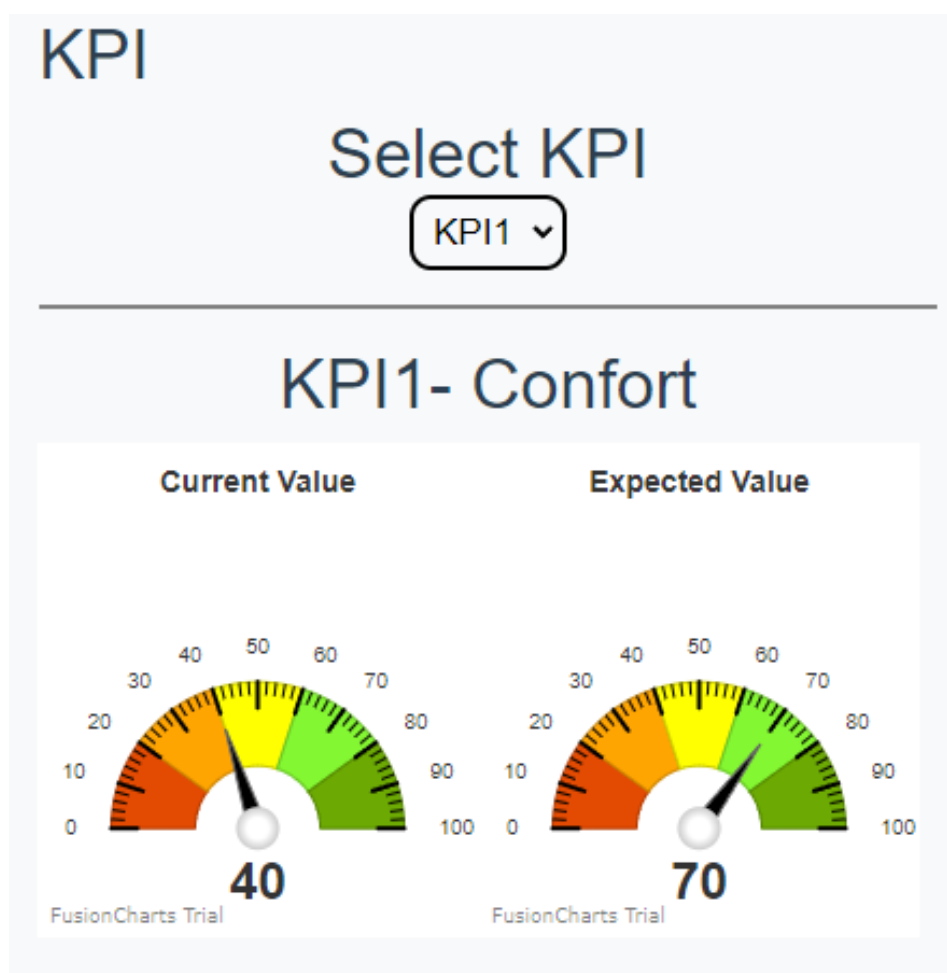


Figura 34: Layout componente KPI

Capitolo 4 Applicazione sviluppata

Gli indicatori visualizzeranno il valore alla base del grafico, e si posizioneranno in una delle 5 zone precedentemente definite suddivise in un intervallo tra 0 e 100, le quali permetteranno di fare le giuste considerazioni sullo stato nel quale il KPI si trovi. Attraverso l'utilizzo di apposite colorazioni che variano dal rosso al verde, si identificherà se il KPI considerato è: in uno stato soddisfacente(verde), in uno stato di allarme (rosso) o se si trovi in uno stato intermedio.

Analizziamo ora come è avvenuta l'implementazione. Verrà descritta la logica utilizzata per la realizzazione dell'elemento select e degli indicatori angolari, per i quali è stata utilizzata la libreria fusionchart di cui abbiamo parlato nella sezione degli Strumenti Utilizzati(sottosezione 3.5.2).

```
<h2>Select KPI
<select @change="chooseChart($event.target.value)">
  <option :value="index" :key="val.name" v-for="(val, index) in this.kpis">
    {{ val.name }}
  </option>
</select>
```

Figura 35: Codice html per la realizzazione della select

La Figura 35 rappresenta il codice delle select: assomiglia molto alle precedenti che abbiamo già avuto modo di commentare(sottosezione 4.2.2).

L'unica differenza, oltre alla diversità dei dati che trattiamo, risulta nell'utilizzo del v-for, tiene traccia del valore dell'opzione da rappresentare e il suo corrispettivo indice.

Al momento della creazione del menù a tendina verrà assegnato alla variabile "value", il valore dell'indice dell'opzione presa in considerazione, e non appena l'utente seleziona il KPI che ha interesse a visualizzare, la funzione chooseChart passerà il valore dell'indice dell'opzione selezionata. Di conseguenza verrà aggiornata l'interfaccia con i valori del KPI preselezionato.

Capitolo 4 Applicazione sviluppata

Per la realizzazione dei due indicatori angolari ci si è serviti di due elementi di tipo fusionchart, la Figura 36 ne riporta uno dei due al fine esplicativo.

```
<h2>{{this.nome}}</h2>
<div>
  <fusioncharts style="float:left;"
    :type="typeWidget"
    :width="285"
    :height="238"
    :dataFormat="dataFormat"
    :dataSource="dataSourceWidget"
    :events="events"
    ref="fc"
  ></fusioncharts>
```

Figura 36: Attributi dell'elemento fusionChart

Tralasciando momentaneamente sia le proprietà "width" e "height", utili al ridimensionamento del componente e sia l'attributo "dataFormat", che specifica il formato dei dati(json nel nostro caso), poniamo ora l'attenzione sull'attributo "dataSource" che fungerà da contenitore per la specifica delle proprietà.

All'interno di esso saranno contenute tutte le specifiche di rappresentazione: i raggi per la costruzione delle sezioni(gaugeOuterRadius, gaugeInnerRadius); il raggio dell'indicatore(pivotRadius); l'intervallo delle linee per la suddivisione delle sezioni(majorTMNumber, majorTMNumber); la descrizione(caption); i limiti del grafico (lowerLimit, upperLimit); il tema che si intende visualizzare fra quelli predefiniti(theme); i colori e i limiti da utilizzare per le diverse sezioni (colorRange); cosa più importante l'attributo "dials" dove sarà presente la proprietà "dial" che conterrà il valore che sarà rappresentato.

La Figura 37 renderà facile la comprensione di quello che è stato detto fin ora.

```
dataSourceWidget: {
  chart: {
    gaugeOuterRadius: "68",
    gaugeInnerRadius: "29",
    pivotRadius: "10",
    majorTMNumber: "9",
    minorTMNumber: "4",
    valueBelowPivot: "1",
    showValue: "1",
    caption: "Current Value",
    captionHorizontalPadding: "2",
    lowerLimit: "0",
    upperLimit: "100",
    theme: "zune"
  },
  colorRange: { ...
  },
  dials: [ ...
  ]
}
```

Figura 37: Elementi che caratterizzano l'attributo dataSource

Capitolo 4 Applicazione sviluppata

Sarà possibile inoltre definire gli eventi ai quali il componente dovrà rispondere attraverso la specifica di essi all'interno dell'attributo "events" associando ad esso funzioni che saranno richiamate al verificarsi dell'evento stesso.

Come accade per gli altri componenti del sistema, bisogna mettere in piedi un meccanismo di recupero dati dal database. Utilizzando la stessa logica applicata agli altri componenti, definiamo una rotta "/kpi" che richiama una funzione del controller che a sua volta farà accesso al model, che estrae i dati dalla tabella del nostro database e che, tramite l'utilizzo di una chiamata ajax, verranno acquisiti dal componente e iniettati nell'interfaccia.

Nella fase "mounted" recupereremo i dati e li rimapperemo in un oggetto utilizzabile dal componente.

Questo processo utilizza ancora la libreria axios che ci facilita notevolmente la realizzazione di chiamate (Figura 38).

```
mounted() {
  axios.get('./kpi').then((response) => {
    let idModified = response.data.map(
      obj => {
        return {
          "name" : obj.name,
          "vAtteso":obj.especValue,
          "vAttuale":obj.currValue,
        };
      });
    this.kpis=idModified;
    this.dataSourceWidget.dials.dial[0].value=this.kpis[0].vAttuale;
    this.dataSourceWidget2.dials.dial[0].value=this.kpis[0].vAtteso;
  })
}
```

Figura 38: Recupero dati dal database

L'oggetto contenente i dati acquisiti, recuperati attraverso la funzione get(), vengono ridefiniti in un nuovo oggetto json e memorizzati nella variabile "kpi" dichiarata in "data". Verranno poi caricati per default i dati del primo KPI presente nel database, attraverso l'assegnazione della variabile "dial" di cui abbiamo già parlato in questa sezione.

Capitolo 4 Applicazione sviluppata

I dati vengono recuperati da una tabella sql presente nel database strutturata come descritto nella Figura 39.

kpid	name	description	currValue	especValue
1	KPI1	testo messaggio	40	70
2	KPI2	testo messaggio	50	80
3	KPI3	testo messaggio	60	90

Figura 39: Tabella KPI del database

La chiave univoca della tabella è rappresentata dall'attributo "kpiId", a cui segue il nome, la sua descrizione, il valore attuale in cui si trova il KPI e per ultimo il valore atteso.

L'estrazione degli elementi dal database avviene definendo il solito model che ci aiuta nel recupero dei dati, tramite la funzione get() implementata dall'ORM(Figura 40).

```
use App\Models\Resources\kpi;

class kpi {

    public function getAllKpi() {
        $kpi = kpi::get();
        return $kpi;
    }
}
```

Figura 40: Model della tabella kpi

Come accade anche per gli altri componenti, il meccanismo di recupero dei dati è reso possibile attraverso l'importazione della classe kpi, la quale estende la classe Model implementata da Eloquent, astruendo la tabella sql grazie alla specifica della variabile "\$table"(nome della tabella) e "\$primaryKey"(nome della chiave primaria).

4.5 Messages

Il componente "messages" rappresenta il penultimo componente caratterizzante dell'interfaccia. La sua funzione è quella di mostrare una serie di messaggi informativi contenenti suggerimenti di azioni possibili per sollecitare il sistema.

La Figura 41 mostra il layout del componente.

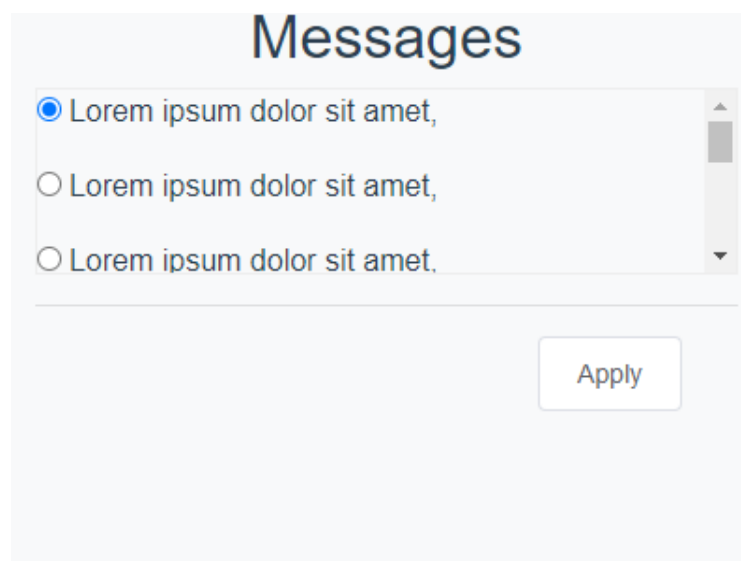


Figura 41: Layout del componente message

La logica di implementazione è semplice. Abbiamo utilizzato un radio button per garantire la scelta esclusiva fra le opzioni presenti, e un pulsante "Apply" per avviare l'azione selezionata. Verranno generati un numero n di messaggi contenenti le azioni, in rapporto a quanti ne saranno presenti nel database. Questo vuol dire che dinamicamente sono caricati nell'interfaccia.

Osserviamo nel dettaglio come è avvenuta la creazione del componente. Il tag `<div>` sarà il contenitore dell'intera "box", il quale ospiterà il tag `` necessario per la creazione di elenchi non ordinati. In seguito, attraverso la specifica dei singoli elementi ``, saranno generati dinamicamente degli input di tipo radio in relazione al numero di messaggi presenti nella variabile "message"(vedi Figura 42).

Capitolo 4 Applicazione sviluppata

```
<div class="box">
  <ul :key="message.messageId" :value="message.content" v-for="message in listmsg">
    <li >
      <input type="radio" v-model="msgselected" name="msgselected" :value="message.messageId"> {{message.content}}
    </li>
  </ul>
</div>
<hr>
<div>
  <el-button id="move" @click="Selection()">Apply</el-button>
</div>
</div>
```

Figura 42: Codice html del componente

La selezione di una delle opzioni avviene tramite utilizzo della direttiva `v-model`, caratteristica del framework `Vue.js`, utile per associare i dati degli input nelle form a delle variabili dichiarate nel codice.

Nella creazione dinamica degli input, attraverso la direttiva `v-for`, verrà associato all'attributo "value" del singolo input il valore univoco "id" del messaggio, il quale verrà memorizzato istantaneamente nella variabile "msgselected" al click dell'opzione scelta.

Per concretizzare la scelta effettuata, utilizzeremo un bottone definito con il tag `<el-button>` implementato dalla libreria, "vue-el-element" che è stata debitamente importata nel file "app.js".

Al click sul pulsante, l'handler dell'evento avvierà la funzione `Selection()`, la quale avrà accesso alla variabile `msgselected` con il valore aggiornato all'ultima scelta dell'utente.

I dati degli input da inserire nell'interfaccia sono contenuti in una tabella sql del nostro database, utilizzando la solita chiamata ajax per il caricamento dei dati all'interno del componente(vedi Figura 43).

```
mounted() {
  console.log("response");
  axios.get('/message').then((response) => {
    console.log(response.data);
    this.listmsg = response.data;
  })
}
```

Figura 43: Funzione per il recupero dei dati dal database

Verrà quindi definita una rotta `"/message"` all'interno del file delle rotte, che si collegherà al nostro controller richiamando la rispettiva funzione che avrà accesso al model della tabella "message" del database, che recupererà i dati e li trasferirà al componente. Se la richiesta avrà successo verranno memorizzati nella variabile "listmsg".

Capitolo 4 Applicazione sviluppata

I dati, come si evince dalla Figura 44, sono strutturati in diverse tuple contenenti le seguenti proprietà: “messageId” per la rappresentazione univoca dell’elemento nonché chiave primaria e “content” che rappresenta il messaggio dell’azione.

messageId	content
1	Lorem ipsum dolor sit amet,
2	Lorem ipsum dolor sit amet,
3	Lorem ipsum dolor sit amet,
4	Lorem ipsum dolor sit amet,
5	Lorem ipsum dolor sit amet,
6	Lorem ipsum dolor sit amet,
7	Lorem ipsum dolor sit amet,

Figura 44: tabella message del database

Per estrapolare i dati dalla tabella utilizzeremo il meccanismo di astrazione messo in piedi da Eloquent, definendo il model per semplificare il recupero dalla tabella, che sarà in un certo senso analogo agli altri componenti(vedi Figura 45).

```
use App\Models\Resources\message;

class messages {

    public function getAllMessages() {
        $messages = message::get();
        return $messages;
    }
}
```

Figura 45: Model della tabella kpi

La rappresentazione grafica e quindi il relativo css del componente risulta estremamente semplice; attraverso l’utilizzo di poche istruzioni è stata possibile la sua realizzazione.

La Figura 46 mostra il css che lo caratterizza.

```
.box {
  margin-left: 0;
  border:1px solid #EEE;
  overflow-y: auto;
  height: 100px;
}
ul{
  text-align: left;
  padding-inline-start:0px
}
#move{
  float: right;
  margin:0 30px;
}
```

Figura 46: Regole css casella messaggi

La class “box” definita nel tag <div> garantirà l’assegnazione del bordo evidenziato e l’aggiunta della proprietà “overflow-y”, che consentirà lo scroll in verticale della message box a cui abbiamo impostato una larghezza pari a 100px e un margine pari a 0. Le regole “text-align” e “padding-inline-start” mostreranno il contenuto dei messaggi a ridosso del bordo sinistro.

Per ultimo abbiamo dichiarato l’id “move” al tag <el-button>, così da centrare il pulsante in basso a destra utilizzando le proprietà “float” e “margin”.

4.6 History

In questa sezione definiremo il sesto componente dell'interfaccia nonché l'ultimo: mostreremo come è stata realizzata la costruzione del grafico xy, che sarà necessaria per la visione degli stati dei KPI in relazione al tempo(Figura 47).

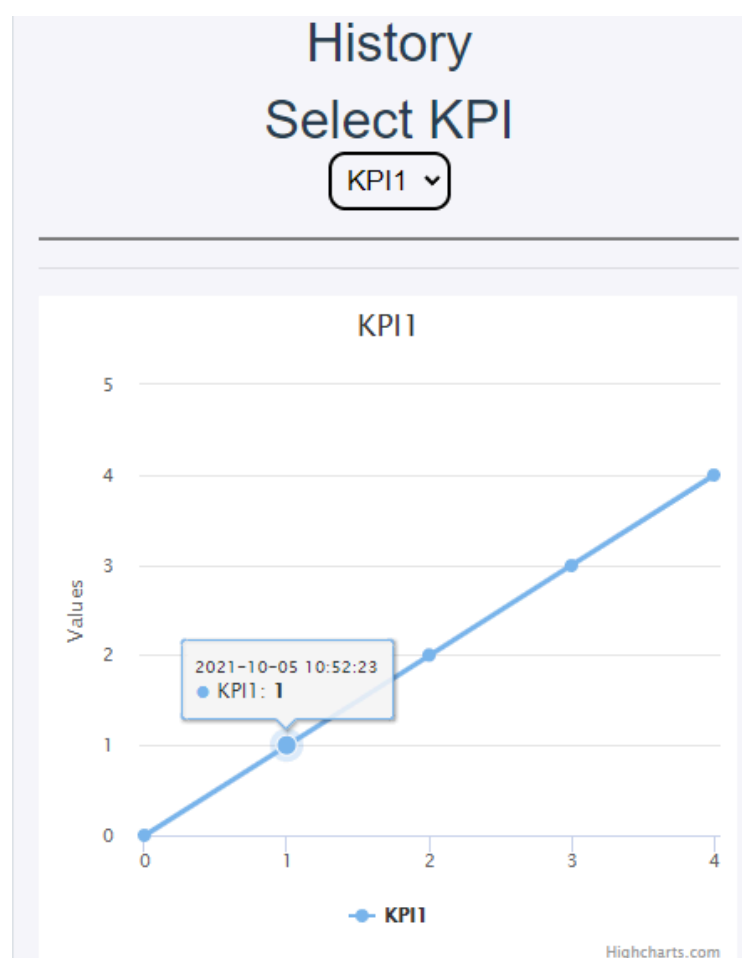


Figura 47: Layout componente history

La linea temporale è rappresentata dall'asse delle x, che sarà formato da numeri in successione (0..n) che scandiscono gli intervalli di tempo nei quali i valori vengono misurati.

Sull'asse delle y, invece, saranno graficati i valori effettivi per ogni istante di tempo.

Capitolo 4 Applicazione sviluppata

Una piccola finestra informativa comparirà al passaggio del puntatore su un particolare punto del grafico, fornendo la data e l'ora esatta dell'istante e il suo relativo valore.

Per permettere la visualizzazione dei diversi KPI presi in analisi, si è utilizzata una select, che gestirà la scelta attraverso un menù a tendina. A seguito della selezione verrà mostrato il grafico contenente i nuovi valori.

Per lo sviluppo è stata utilizzata la libreria Highcharts (illustrata nella sottosezione 3.5.3): anch'essa si avvale dell'utilizzo di SVG (Scalable Vector Graphics), che ci permette di realizzare oggetti grafici su base vettoriale per evitare i problemi di risoluzione.

Vedremo nel dettaglio come è avvenuta la realizzazione del componente (Figura 48).

```
<h2>Select KPI
  <select :value="'KPI1'" @change="chooseChart($event.target.value)">
    <option :value="chart.name" :key="chart.name" v-for="chart in this.points">
      {{ chart.name }}
    </option>
  </select>
  <hr style="border: 1px grey solid;">
</h2>
<hr>
<div>
  <highcharts
    class="chart"
    :options="chartOptions"
    :updateArgs="updateArgs"
    ref="chart"
  ></highcharts>
</div>
```

Figura 48: Codice html per la realizzazione del grafico

Come prima cosa osserveremo il comportamento della select, la quale genererà la lista dei KPI ricavata dal database attraverso la quale effettueremo la scelta del elemento che si vuole consultare.

L'implementazione avviene attraverso la composizione del tag <select> come involucro delle tag <option>, le quali vengono generate dinamicamente tramite il meccanismo del v-for messo in piedi da Vue. In poche parole verrà assegnato all'attributo "value", di ogni singola opzione, il nome del rispettivo KPI.

Tramite l'handler per la gestione dell'evento change (vue permette di definirlo rapidamente tramite l'utilizzo di @change), richiameremo la funzione chooseChart a cui passeremo il valore dell'attributo "value"

dell'opzione scelta, in modo da sollecitare visualizzare il grafico del rispettivo KPI (vedi Figura 49).

```
methods: {
  chooseChart(type) {
    this.chartOptions.series.pop();
    this.chartOptions.series.push(this.points[type]);
    this.chartOptions.title.text=this.points[type].name;
  }
}
```

Figura 49: Funzione chooseChart per lo switch fra i KPI

Nella figura, si può osservare la logica che permette la visualizzazione dei diversi grafici. Essenzialmente il tutto si basa sull'utilizzo di due funzioni:

1. La funzione **pop()** per eliminare il grafico attualmente plottato: questo è memorizzato nell'attributo "series" contenuto a sua volta in "chartOptions" che è essenziale per la costruzione del componente.
2. La funzione **push()** che si occuperà di reinserire i dati del KPI scelto, all'interno dell'attributo "series".

Infine verrà riassegnato al grafico il nome del KPI scelto.

Come detto, l'attributo "chartOptions", conterrà tutte le informazione del grafico (vedi Figura 50).

```
chartOptions: {
  chart: {
    type: "line",
  },
  title: {
    text: ""
  },
  plotOptions: {
    series: {
      animation: {
        duration: 2000,
      },
    },
  },
  series: [],
},
};
```

Figura 50: Attributo chartOption per la costruzione del grafico

Una sezione "chart" definirà il tipo di grafico che si sta realizzando, attraverso la specifica dell'attributo "type" impostato a "line".

Capitolo 4 Applicazione sviluppata

La sezione “title” comprende l’attributo “text”, che conterrà il nome del KPI rappresentato e inserito dinamicamente in fase di costruzione del grafico.

“PlotOptions” conterrà le opzioni aggiuntive al fine di una rappresentazione più accattivante per l’utente, come ad esempio l’attributo “animation”, che farà sì che il grafico si costruisca gradualmente nel tempo stabilito di 2000ms.

Come ultimo troviamo la proprietà “series”, che conterrà i valori assunti dal KPI in funzione del tempo che abbiamo interesse a rappresentare. I dati dovranno essere strutturati in formato json secondo una logica precisa:

```
{  "name": "KPI1",
  "data": [ ["2021-10-05 10:49:15", 0],
            ["2021-10-05 10:52:23", 1],
            ["2021-10-05 10:53:02", 2],
            ["2021-10-05 10:54:11", 3],
            ["2021-10-05 10:58:26", 4]
        ]
}
```

Ci sarà un attributo “name” contenente il nome del KPI e un attributo “data” contenente un array di array composto dai valori di x e y che dovranno essere rappresentati.

Vediamo come è stato realizzato il meccanismo per la conversione dei dati (Figura 51).

```
public function getKpiHistory() {
    $allkpi=array();
    $kpi=kpi::get()->pluck('name','kpiId');
    foreach($kpi as $key => $val){
        $data=array();
        $array=array();
        $kpihistory = kpihistory::where('kpiId','=',$key)->pluck('value','time');
        foreach( $kpihistory as $key => $val) {
            $array[] = array($key,$val);
        }
        $data["data"]=$array;
        $data["name"]=$val;
        $allkpi[$val]=$data;
    }
    return json_encode($allkpi);
}
```

Figura 51: Model tabella kpihistory

Come prima cosa vengono recuperati tutti i nomi e i relativi id univoci dei KPI presenti. Gli id recuperati verranno utilizzati per la ricerca

Capitolo 4 Applicazione sviluppata

dei valori dei rispettivi KPI, attraverso l'utilizzo della query "where" caratteristica della sintassi di Eloquent.

Attraverso un ciclo for verranno recuperati i dati da graficare di ogni singolo KPI e rimappati all'interno di un array di array.

Gli stessi verranno iniettati in un array associativo contenente gli indici "data" e "name".

Stesso ragionamento verrà riproposto per tutti i kpi e tutti i dati saranno memorizzati in un array contenitore, il quale sarà trasformato in un oggetto json tramite la funzione "json_encode" presente nelle librerie del linguaggio php.

Sarà poi il controller dell'interfaccia a richiamare la funzione del getKpiHistory() contenuta nel model.

Il controller verrà richiamato dal componente tramite il solito meccanismo delle rotte attraverso una chiamata ajax, che garantirà l'acquisizione dei dati mediante l'utilizzo della libreria axios.

Gli stessi dati già formattati dal model verranno memorizzati nella variabile "points" e sarà poi la select a occuparsi della visualizzazione dinamica dei dati, in relazione alla scelta effettuata.

Vediamo come avviene il caricamento dei dati(Figura 52).

```
mounted() {
  axios.get('./kpihistory').then((response) => {
    this.points=response.data;
    this.chartOptions.series.push(this.points[Object.keys(this.points)[0]]);
    this.chartOptions.title.text=this.points[Object.keys(this.points)[0]].name;
  });
};
```

Figura 52: recupero dati dal database

Come succede per gli altri componenti presenti nell'interfaccia, il recupero dei dati avviene nella fase mounted, che ci assicura che tutte le parti che costituiscono il nostro oggetto vue siano state effettivamente montate.

La chiamata alla rotta "/kpihistory" ci restituirà tutti i dati dei KPI presenti nel formato definito.

Capitolo 4 Applicazione sviluppata

Per default all'avvio dell'interfaccia verrà rappresentato il primo KPI presente nel database. Questo avverrà grazie all'utilizzo della funzione "push()", la quale passerà i valori all'interno dell'attributo "series" (definito dalla libreria highchart).

I dati da graficare sono contenuti nel nostro database in una tabella MySQL, così come accade per tutti gli altri dati della nostra interfaccia.

La Figura 53 illustra come è strutturata la tabella e mostra alcune tuple che la compongono.

kpihisId	kpid	time	value
1	1	2021-10-05 10:49:15	0
3	1	2021-10-05 10:52:23	1
4	1	2021-10-05 10:53:02	2
5	1	2021-10-05 10:54:11	3
6	1	2021-10-05 10:58:26	4
7	2	2021-10-14 12:59:42	12.2
8	2	2021-10-14 13:01:32	13.3
9	2	2021-10-14 13:09:57	17.7
10	2	2021-10-14 13:10:58	19.3

Figura 53: tabella kpihistory

La proprietà "kpihisId" rappresenterà univocamente i punti che descriveranno i nostri KPI e sarà quindi la chiave primaria.

Il "kpid" esprime univocamente il KPI al quale si riferisce quel particolare dato, ricordando che la tabella kpihistory e kpi sono legate da un vincolo di integrità referenziale tramite l'attributo kpid, in quanto i dati contenuti in figura si riferiscono ai KPI dichiarati nella tabella "kpi" mostrata nella Figura 39.

Gli attributi "time" e "value" rappresentano rispettivamente i valori dell'ascissa x e dell'ordinata y che saranno plottati all'interno del nostro grafico.

4.7 Il Controller

Il controller rappresenta in parte il gestore di tutta l'applicazione, che richiamerà la vista della nostra interfaccia e si occuperà di gestire tutti i model definiti per il recupero dei dati. Si occuperà quindi di inizializzare tutte le istanze dei model, le quali estrarranno i dati dal nostro database e conterrà tutte le funzioni che verranno richiamate dai componenti, utilizzando il meccanismo delle rotte per l'acquisizione dei dati. Vedremo nello specifico come è strutturato il controller dell'interfaccia (Figura 54).

```
class InterfaceController extends Controller
{
    public function __construct() {
        $this->messages = new messages;
        $this->diagrams = new diagrams;
        $this->kpis = new kpis;
        $this->subsystems = new subsystems;
        $this->kpihistorys = new kpihistorys;
    }
    public function index()
    {
        return view(['interface']);
    }
    public function getMessage(){
        $themessages=$this->messages->getAllMessages();
        return $themessages;
    }
    public function getAlbero(){
        $theNodes=$this->diagrams->getAllNodes();
        return $theNodes;
    }

    public function getKpi(){
        $theKpi=$this->kpis->getAllKpi();
        return $theKpi;
    }

    public function getSubsystem(){
        $theSubsystem=$this->subsystems->getAllSubsystem();
        return $theSubsystem;
    }
    public function getKpiHistory(){
        $theKpiHistory=$this->kpihistorys->getKpiHistory();
        return $theKpiHistory;
    }
}
```

Figura 54: Il Controller dell'interfaccia

Vediamo come nella funzione costruttrice "`__construct`" vengono iniziate le 5 istanze dei rispettivi model, attraverso l'utilizzo della keyword "`new`". Verranno definite le 5 funzioni che richiameranno i metodi del

model per il recupero dei dati. E' definita nella funzione "index" la vista da richiamare, ossia la pagina html contenente i sei componenti dell'intera interfaccia.

Quest'ultima sarà richiamata da una rotta definita nel file "web.php".

4.8 Il Layout

I componenti sono stati organizzati con un layout a griglia, disposti in due righe e tre colonne. Per semplificare la realizzazione si è utilizzato Bootstrap, un framework efficiente per lo sviluppo di interfacce web e per rendere i siti internet responsive: permette quindi di creare pagine con un layout che si adatta automaticamente al tipo di dispositivo usato.

Il suo scopo è proprio quello di raggruppare le caratteristiche grafiche comuni agli elementi del linguaggio html, per renderli riutilizzabili con facilità. Vediamo in seguito come è stata strutturata la pagina html dell'interfaccia (Figura 55).

```
<div class="container-fluid ">
  <div class="row">
    <div class="col border bg-light">
      <togglevs ></togglevs>
    </div>

    <div class="col border bg-light">
      <albero></albero>
    </div>

    <div class="col border bg-light">
      <messagebox></messagebox>
    </div>
  </div>
  <div class="row">
    <div class="col border bg-light" >
      <subsystems ></subsystems>
    </div>

    <div id="cruscotto" class="col border bg-light" >
      <cruscotto ></cruscotto>
    </div>

    <div class="col border bg-light">
      <grafico></grafico>
    </div>
  </div>
</div>
```

Figura 55: codice della vista dell'interfaccia

I due tag div con l'attributo class "row" conterranno rispettivamente 3 componenti cadauno, i quali saranno contenuti in tre div tag con l'attributo "col border bg-light". Grazie all'importazione di bootstrap, a questi tag saranno definite regole css per la visualizzazione di un layout a

Capitolo 4 Applicazione sviluppata

forma di griglia bordato(boder) e con un leggero colore in sottofondo(bg-light), così come è descritto all'interno della sua documentazione a cui è possibile attingere dal sito web:

<https://getbootstrap.com/docs/4.0/layout/grid/>

Tutti i nomi dei tag html dei componenti che caratterizzano l'interfaccia sono definiti all'interno del file "app.js" e dovranno essere racchiusi all'interno di un tag <div> con id "app".

```
Vue.component('cruscotto', require('./components/cruscotto.vue').default);
Vue.component('messagebox', require('./components/MessageBox.vue').default);
Vue.component('grafico', require('./components/grafico.vue').default);
Vue.component('togglevs', require('./components/togglevs.vue').default);
Vue.component('albero', require('./components/composealbero.vue').default);
Vue.component('subsystems', require('./components/subsystems.vue').default);

const app = new Vue({
  el: '#app',
});
```

Figura 56: Dichiarazione tag html dei componenti vue

La Figura 56 descrive come avviene l'associazione tra i <tag> html dei componenti e i file vue contenenti il codice degli stessi. È definito anche l'id del <div>, al quale vue dovrà far riferimento per iniettare i componenti che sono stati definiti in precedenza, qualora si presentassero i loro tag identificativi. Questo avviene nella dichiarazione dell'attributo "el" dell'istanza vue.

Capitolo 5

Conclusioni

Lo studio, l'analisi e lo sviluppo di ogni singolo componente ha permesso la realizzazione dell'intera interfaccia, che oggi si presenta con un layout che si avvicina molto a quello definito in fase di progettazione.

La Figura 57 mostra il risultato finale dell'interfaccia nella sua completezza.

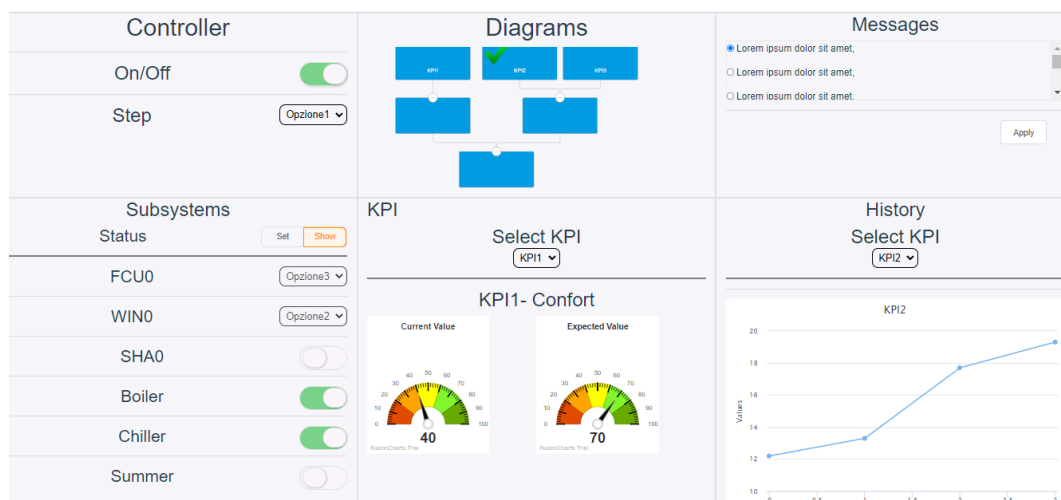


Figura 57: Interfaccia Finale

Il layout finale si presenta di una forma molto simile a quella ideata inizialmente, soddisfa i requisiti funzionali e non funzionali del progetto, garantendo una buona visualizzazione dello stato dei sistemi che stiamo trattando.

La nostra interfaccia soddisfa le esigenze di: utilità, facilità ed efficienza d'uso.

Capitolo 5 Conclusioni

Questo vuol dire:

- **Utilità** in termini di utilizzo: l'utente dovrebbe trarre beneficio e quindi vantaggio nell'utilizzo dell'interfaccia che deve soddisfare i requisiti per i quali è stata ideata.
- **Facilità** nel suo utilizzo: l'utente poter utilizzare l'interfaccia in maniera immediata e soddisfacente, senza uno eccessivo sforzo di apprendimento del suo funzionamento.
- **Efficiente** nel funzionamento: l'interfaccia deve garantire la veridicità dei dati mostrati e quindi l'affidabilità nella visualizzazione dei dati

Tutti questi concetti sono quelli che definiscono una buona User Experience che definisce le percezioni e le impressioni di una persona che emergono dall'utilizzo di un prodotto.

Sicuramente può essere ritenuta un qualcosa di pressoché soggettivo, ma allo stesso tempo, se si soddisfano le caratteristiche definite poc'anzi, si potrà notare la differenza in termini di soddisfazione dell'utente.

Atmail riporta alcuni dati statistici che ribadiscono l'importanza della User Experience e di quanto essa possa fare la differenza (far riferimento a [5]) (Figura 58).

Capitolo 5 Conclusioni

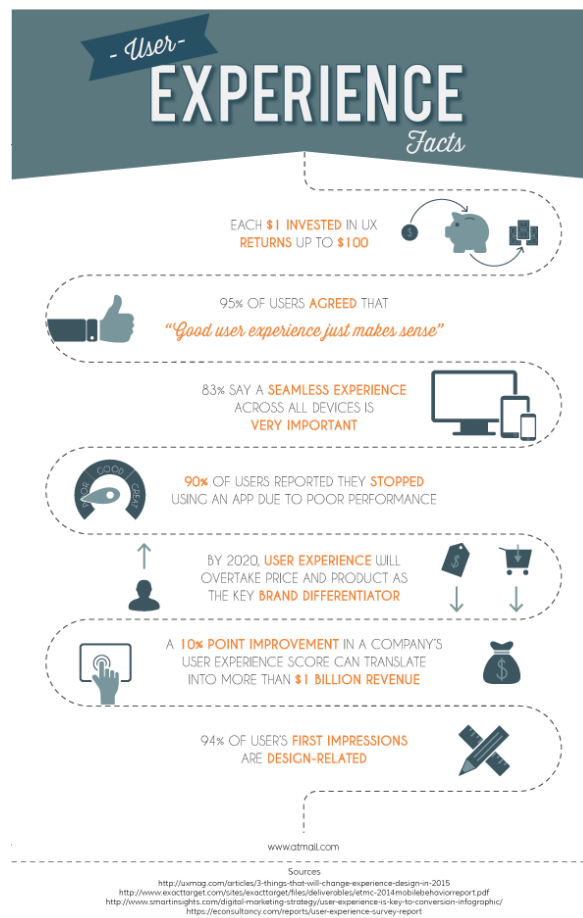


Figura 58: Importanza User Experience

- Ogni 1\$ investito in UX restituisce fino a 100 \$.
- L'85% degli utenti ha riconosciuto che una buona esperienza utente ha senso.
- L'83% afferma che un'esperienza senza interruzioni su tutti i dispositivi è importantissima.
- Il 90% degli utenti ha dichiarato di aver smesso di utilizzare un'app a causa delle scarse prestazioni.
- Un miglioramento del 10% nel punteggio dell'esperienza utente di un'azienda può tradursi in oltre 1 miliardo di dollari di entrate.
- Il 94% delle prime impressioni dell'utente è legato al design.

5.1 Miglioramenti

In relazione alla nostra interfaccia ora ci soffermeremo sugli aspetti che potranno essere migliorati.

Per prima cosa si potrebbe ottimizzare la gestione dei dati. I dati all'interno dell'interfaccia vengono caricati nel momento della creazione della pagina e rimarranno invariati per tutto il ciclo di vita. Nel caso in cui la pagina venisse ricaricata, i dati saranno aggiornati.

Quindi un miglioramento potrebbe essere quello di modificare la nostra interfaccia in un sistema real-time.

*“un **sistema real-time** (in italiano "sistema in tempo reale") è un sistema in cui la correttezza del risultato delle sue computazioni dipende non solo dalla correttezza logica ma anche dalla correttezza temporale. Quest'ultima è spesso espressa come tempo massimo di risposta. Alle computazioni eseguite dai sistemi real-time ci si riferisce con il termine inglese real-time computing o meno frequentemente con il termine italiano computazioni in tempo reale. A questi sistemi sono spesso associati anche requisiti di affidabilità e interazione con l'ambiente.[6]”*

Così come definito dal sito Wikipedia, un sistema real-time permette in poche parole l'aggiornamento della nostra interfaccia in tempo reale.

Con tempo reale in realtà stiamo definendo un intervallo di tempo sufficientemente piccolo in cui i dati si aggiornano, così da consentire una approssimazione della correttezza temporale dei dati rappresentati.

Un ulteriore aspetto che potrebbe essere migliorato è la rappresentazione nel grafico dei dati temporali sull'asse delle ascisse. Proiettare i dati sull'asse con una formattazione del tipo “12/maggio” potrebbe facilitare una consultazione rapida e immediata, evitando l'utilizzo di una successione di punti che descriva i vari istanti a cui si riferiscono i particolari dati.

Bibliografia

- [1] <https://www.geekandjob.com/wiki/vue>. consultato a ottobre 2021.
- [2] <https://en.wikipedia.org/wiki/Vue.js>. consultato a ottobre 2021.
- [3] <https://vuejs.org/v2/guide/>. consultato a ottobre 2021.
- [4] <https://en.wikipedia.org/wiki/Laravel>. consultato a ottobre 2021.
- [5] <https://www.atmail.com/blog/the-importance-of-ux/>. consultato a ottobre 2021.
- [6] https://it.wikipedia.org/wiki/Sistema_real_time. consultato a ottobre 2021.