

Università degli studi di Genova

Corso di Laurea in Informatica

Accessibility by design, una libreria per lo sviluppo web accessibile

by

Mattia Dapino

Ottobre 2021

Relatore Prof.ssa Marina Ribaudo
Co-relatore Ing. Giampiero Granatella

Dibris

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Indice

1 Introduzione **3**
1.1 Cosa significa accessibilità? 4
1.2 Validazione di accessibilità 5
1.3 Esempi di elementi accessibili e non 6

2 Prototipazione della libreria **10**
2.1 Componenti 10
2.1.1 Search 10
2.1.2 Read 13
2.1.3 Create e Update 14
2.2 Personalizzazione 15
2.3 Model 15
2.4 Under the hood 16

3 Tecnologie **18**
3.1 Test accessibilità 18
3.1.1 AC Checker 18
3.1.2 WAVE Webaim 19
3.1.3 Voiceover 19
3.1.4 Lighthouse 20
3.2 Tecnologie 20
3.3 Automazione 21
3.3.1 Semantic versioning 21
3.3.2 Conventional commit 22

4 Conclusioni **23**
4.1 Sviluppi futuri 24
4.1.1 Accessibilità per le applicazioni mobili 24
4.1.2 Nuovi modelli 25
4.1.3 Upgrade dello stack tecnologico 25
4.1.4 Integrazione in Portofino 25

1 Introduzione

Negli ultimi anni, e ancor più negli ultimi mesi, molti servizi della pubblica amministrazione, di società private, bancarie e assicurative che una volta venivano offerti solo da sportello, sono diventati disponibili tramite il canale Internet attraverso applicazioni web o mobili. Con la diffusione massiva dei servizi online, accelerata dalla corrente pandemia, l'interazione fra l'uomo e la macchina è diventata sempre più importante e le interfacce tramite cui i servizi vengono fruiti dagli utenti (clienti, studenti, cittadini) non solo devono essere sempre più potenti, intelligenti e complete, ma anche di facile fruibilità e usabilità. Il concetto di inclusione assume quindi un ruolo fondamentale per permettere alle persone "fragili" di usufruire di servizi a loro ancor più necessari. Questa categoria di persone comprende non sono solo i disabili ma include fasce di popolazione (quali gli anziani) che prima non accedevano ai servizi online.

Ad oggi lo sviluppo di interfacce si è focalizzato più sulla potenza e la ricchezza di funzionalità, rispetto all'accessibilità e all'usabilità. Il mercato offre molte soluzioni che si propongono di risolvere svariati casi limite in cui è necessario mostrare una grande mole di dati (es. VueTable, D3JS) o fortemente legate ad uno specifico software, si vedano le librerie grafiche per SAP, oppure i pacchetti grafici il cui unico compito è quello di semplificare i vincoli grafici, senza prendere in considerazione gli aspetti di accessibilità e funzionalità (es. Bootstrap, Vuetify, Ant Design).

Da una parte ci troviamo ad avere sistemi molto complessi e poco personalizzabili, ma ricchi di funzionalità; dall'altra abbiamo componenti curate esteticamente e molto elastici su cui occorre implementare alcuni requisiti tecnici quali paginazione, ordinamento, ricerca e gestione delle entità nel database, mentre tutta la parte relativa all'accessibilità rimane, per lo più, a discrezione e sensibilità dello sviluppatore.

In questa prova finale abbiamo realizzato una libreria di componenti riutilizzabili che prendano in carico la risoluzione delle problematiche di accessibilità senza limitare le funzionalità dell'interfaccia.

Ma cosa significa per un'applicazione essere accessibile?

1.1 Cosa significa accessibilità?

Secondo la legge n.4 del 9 gennaio 2004 [4], per accessibilità si intende *la capacità di un sistema informatico di erogare servizi e informazioni in maniera fruibile anche a persone che necessitano configurazioni atipiche o che siano soggette a limitazioni tecnologiche, ambientali o fisiche.*

Questa legge, oltre a definire quali siano i requisiti di accessibilità, ha anche attribuito ad AgID, l'agenzia per l'Italia digitale controllata dalla Presidenza del Consiglio dei Ministri, numerosi compiti di vigilanza e monitoraggio [14] sulla Pubblica Amministrazione (PA) che, dal 23 settembre 2020, ha l'obbligo di garantire l'accessibilità delle piattaforme utilizzate per erogare i servizi al cittadino e di riportare periodicamente lo stato.

Al fine di definire parametri e procedure per garantire il rispetto della normativa, AgID ha emanato le linee guida in ambito accessibilità per gli strumenti informatici [15] della PA. Queste linee guida sono entrate in vigore il 10 gennaio 2020 e sono fondate sull'idea di un progressivo miglioramento delle piattaforme al fine di erogare servizi sempre più accessibili e usabili.

Oltre alle linee guida fornite da AgID anche il W3C, World Wide Web Consortium, ha creato le Web Content Accessibility Guidelines, di seguito WCAG [20], nell'ambito della Web Accessibility Initiative (WAI). Queste linee guida, tenendo conto di molti tipi di disabilità, come le disabilità visive, uditive, cognitive e fisiche o combinazioni di queste, forniscono importanti fondamenti sui principi di accessibilità e consentono di estrarre quattro principi fondamentali che un'applicazione deve avere per essere accessibile:

- **Percepibile**: Gli elementi presenti sulla pagina devono poter essere percepiti dall'utente.
- **Utilizzabile**: Gli elementi della pagina devono poter essere utilizzati dall'utente.
- **Comprensibile**: Il contenuto e le azioni che possono essere eseguite devono essere comprensibili.
- **Robusto**: Il passare del tempo non deve contrastare i precedenti fondamenti.

È immediato notare che questi quattro principi non sono ad esclusivo vantaggio degli utenti diversamente abili, ma si possono tranquillamente includere nell'insieme dei principi di usabilità di qualsiasi applicazione, quindi, rispettando il requisito di accessibilità si ha un miglioramento qualitativo dell'applicazione per *tutti* gli utenti.

1.2 Validazione di accessibilità

I test automatici sono un ottimo punto di partenza per avere un'idea di massima sull'accessibilità di un'applicazione e consentono di avere una visione di insieme del sistema, anche se in contesti di integrazione continua non sono sufficienti per determinare che il risultato sia effettivamente accessibile e fruibile nella sua interezza.

Si pensi ad esempio agli elementi che non sono semanticamente cliccabili, ma che vengono utilizzati come tali e, di conseguenza, sono "invisibili" sia agli strumenti di verifica automatica che alla persona non vedente che utilizza uno screen reader.

Al momento, esattamente come per i test di usabilità ampiamente descritti nel libro "Don't make me think" [10], siccome si deve analizzare l'interazione, è difficile per l'utente percepire gli elementi e le loro modalità di utilizzo (*affordance*), se questi vengono usati al di fuori del loro contesto di utilizzo standard.

Per la validazione di questo progetto di prova finale è stato preziosissimo il supporto dello Staff del Laboratorio per l'accessibilità del SIMAV, il Centro di servizio di Ateneo di Simulazione e Formazione Avanzata[1]. Lo Staff ha messo a disposizione le competenze acquisite nel tempo e le attrezzature per controllare l'effettivo livello di accessibilità che è possibile ottenere utilizzando le componenti software sviluppate in questa prova finale. Sfortunatamente, a causa dell'attuale condizione sanitaria, non è stato possibile effettuare prove più esaustive con utenti che utilizzano quotidianamente metodi di fruizione atipici.

Il risultato dei test eseguiti, sebbene abbia rivelato alcune criticità derivanti dalle personalizzazioni effettuabili dall'utente, è stato molto soddisfacente in quanto, in particolare nelle verifiche relative all'uso tramite screen reader, ha dimostrato un'ampia navigabilità dei contenuti con le componenti non personalizzate. Nel caso peggiore, ossia dove lo sviluppatore non ha prestato la minima attenzione alle interazioni atipiche, il risultato dei test è stato accettabile.

In merito alle personalizzazioni è stato immediato notare che lo sviluppatore potrebbe inserire elementi non accessibili nella libreria e, di conseguenza, limitare l'operabilità della componente. Questo argomento è meglio discusso nella sezione 2.2.

[1] https://simav.unige.it/

1.3 Esempi di elementi accessibili e non

Esistono molti elementi all'interno di un'applicazione che possono determinare o agevolare l'accessibilità ai contenuti. Uno di questi è la presenza di un link che consente di saltare l'header della pagina e di accedere direttamente ai contenuti, nella figura 1 possiamo vedere un esempio di questo link presente su Google. In questo modo si fornisce all'utente che utilizza uno screen reade un metodo agevole per evitare di dover iterare su tutte le voci di menù presenti nella pagina, migliorando notevolmente l'esperienza d'uso.

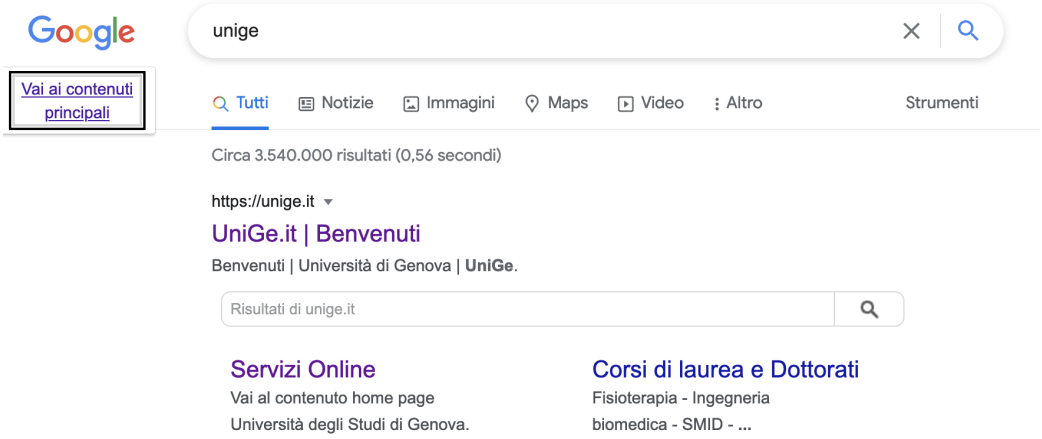


Figura 1: Collegamento presente sulla pagina delle risposte di Google per saltare al contenuto

La figura 2 mostra degli elementi (icone) tradizionalmente non usati per l'interazione che diventano cliccabili grazie all'uso dello stile CSS. Si può notare che le due immagini a destra e sinistra sono identiche, ma non il codice. Siccome l'elemento *b-icon* renderizza come *svg*, che per semantica non è cliccabile, nel primo caso (immagine a sinistra nella figura) l'elemento verrebbe giustamente escluso dalla navigazione perché identificato come elemento decorativo e non funzionale. Nel secondo caso, dove è presente un bottone con l'indicazione di come deve essere letto da uno screen reader, l'utente che lo utilizza sarà in grado di percepirlo e di interagirvi esattamente come si fa con un normale bottone.

Questo caso specifico è stato riscontrato in una personalizzazione fatta sul progetto di demo[2] durante una sessione di test presso il SIMAV, evidenziando

[2]Il progetto di demo utilizza una base di dati realizzata nel contesto del progetto di Basi Dati 2 che rappresenta la relazione n:m tra i Pokemon e gli allenatori che li possiedono

così quanto le operazioni di personalizzazione possano introdurre funzionalità non accessibili.



Figura 2: Esempio di bottoni identici, ma in un caso (sinistra) non accessibili

Un altro fattore molto importante, sia per l'accessibilità che per l'usabilità di un'applicazione, sono i contrasti degli elementi con lo sfondo che potrebbero rendere gli elementi invisibili o poco chiari non solo in contesti di ipovisione, ma anche semplicemente quando si accede alla rete con lo schermo sotto il sole. Escludendo i casi di daltonismo, che devono essere trattati con un altro approccio e richiedono cambiamenti nella palette dei colori dell'applicazione, un semplice cambiamento di tonalità delle tinte scelte, come si può vedere nella figura 3, può migliorare molto l'esperienza d'uso senza avere grosse ripercussioni sullo stile del linguaggio grafico dell'applicazione.

Se prendiamo l'immagine di esempio possiamo notare che la form di sinistra ha una tonalità di blu molto più chiara rispetto a quella di destra dove i testi risultano più leggibili. Si potrebbe obiettare che aumentando ulteriormente il contrasto si avrebbe un risultato ancora più accessibile ma, come ampiamente illustrato nell'articolo "The Aesthetic-Accessibility Paradox" [17], si peggiorerebbe l'estetica del sito, e quindi una via di mezzo che abbia un contrasto sufficiente diventa una soluzione accettabile.

Altri fattori invece sono determinanti per lo stile di una applicazione. Uno di questi casi è la scelta del font, alcune caratteristiche specifiche del testo possono ridurre la leggibilità del testo o addirittura renderlo illeggibile. Come si può vedere nella figura 4 i font nella colonna di destra sono molto meno leggibili rispetto a quelli di destra. Tra i font che possono essere considerati

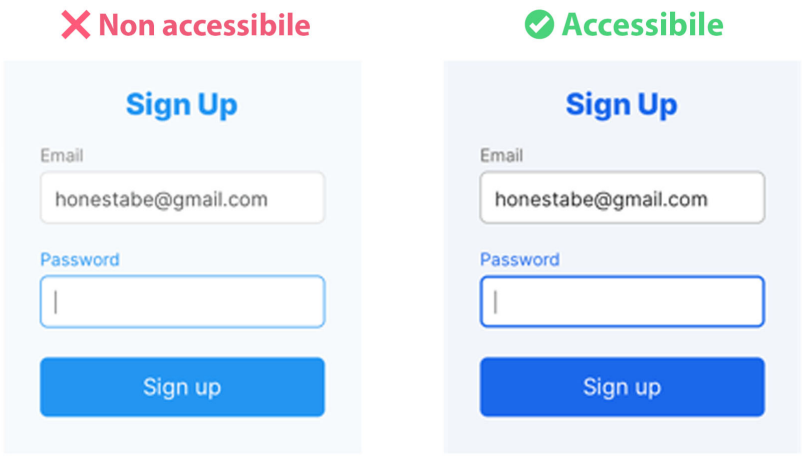


Figura 3: Due form con contrasti diversi

accessibili vanno inoltre distinte due categorie fondamentale, i font serif e i font sans-serif che, rispettivamente, vanno utilizzati per i paragrafi e per i titoli.

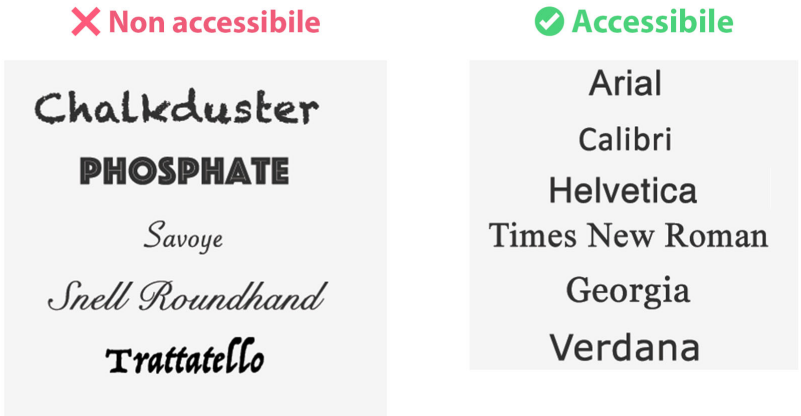


Figura 4: Diversi font divisi tra accessibili e non

Per le persone con problematiche cognitive o disturbi dell'attenzione l'allineamento del testo e gli spazi bianchi sono un grande aiuto nel sostenere l'attenzione durante la lettura. La figura 5 mostra chiaramente come il testo giustificato sia molto più leggibile rispetto a quello allineato a sinistra.

Un ultimo fattore determinante per l'accessibilità è il contrasto del testo con lo sfondo. Se osserviamo la figura 6 diventa chiaro notare che alcuni

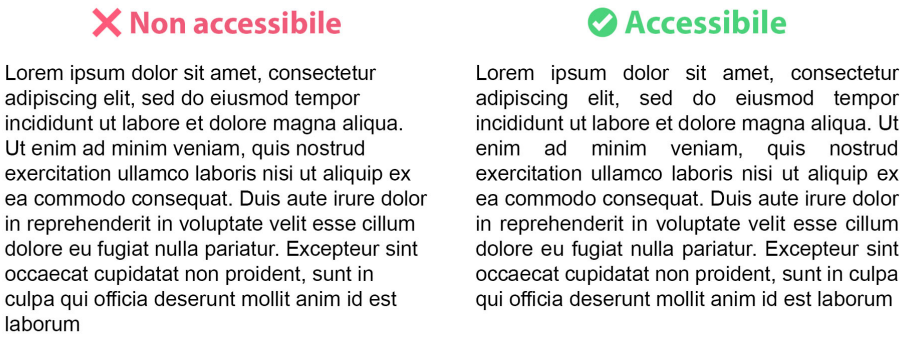


Figura 5: Due testi con diversi allineamenti del testo

colori sovrapposti ad altri sono meno leggibili rispetto ad altre combinazioni. Secondo le WCAG[20], per fare si che il testo sia sia accessibile, questo deve avere con lo sfondo un contrasto di almeno 4.5:1 e, calcolando che circa il 9% della popolazione mondiale soffre di difetti della vista, una buona porzione dell'utenza beneficerebbe dal rispetto di questo rapporto.

Foreground \ Background	Red	Orange	Yellow	Green	Blue	Violet	Black	White	Gray
Red		Poor	Good	Poor	Poor	Poor	Good	Good	Poor
Orange	Poor		Poor	Poor	Poor	Poor	Good	Poor	Poor
Yellow	Good	Good		Poor	Good	Poor	Good	Poor	Good
Green	Poor	Poor	Poor		Good	Poor	Good	Poor	Good
Blue	Poor	Poor	Good	Good		Poor	Poor	Good	Poor
Violet	Poor	Poor	Good	Poor	Poor		Good	Good	Poor
Black	Poor	Good	Good	Good	Poor	Good		Good	Poor
White	Good	Good	Good	Poor	Good	Good	Good		Good
Gray	Poor	Poor	Good	Good	Poor	Poor	Poor	Good	

Figura 6: Tabella che illustra il contrasto tra vari colori

2 Prototipazione della libreria

Questo progetto è una libreria JavaScript, disponibile su NPM, scritta in VueJS [21] per il framework Portofino[11], che ha lo scopo di agevolare gli sviluppatori nel creare applicazioni web che siano accessibili e nel contempo di ridurre il più possibile la duplicazione del codice, tramite un insieme di componenti preconfezionate, altamente personalizzabili e integrabili. Con l'uso di queste componenti si può velocizzare lo sviluppo e abbattere i costi, rispettando il requisito di accessibilità e ottenendo un effettivo miglioramento qualitativo dell'applicazione per tutti gli utenti.

La libreria si incentra sulla componente di interazione con i dati presentati nell'interfaccia per consentire l'esecuzione delle operazioni operazioni CRUD (Create, Read, Update, Delete) sulle risorse e sulle entità presentate da Portofino a partire dalle definizioni delle entità del framework. Le componenti sono indipendenti da librerie grafiche dato che è possibile fornire un set di componenti personalizzate o utilizzarne altre già pronti forniti con la libreria. Nello specifico, come vedremo nelle prossime sezioni, la libreria genera card di dettaglio, form per la modifica, e tabelle con filtro di ricerca, ordinamento e paginazione altamente personalizzabili. Le varie componenti vengono interamente generate dalla definizione delle azioni di Portofino, ma rimangono comunque ampiamente personalizzabili in ogni loro parte, tramite gli *slot*[3] di VueJS che consentono la sostituzione di parti del DOM (Document Object Model) della componente con altre iniettate nella componente padre dallo sviluppatore.

2.1 Componenti

La libreria fornisce quattro componenti, che rappresentano le operazioni di creazione, lettura, modifica e la ricerca delle entità, per interagire con i dati. Queste componenti sono liberamente utilizzabili dallo sviluppatore per comporre in totale libertà l'applicazione, inserendo anche più elementi su una singola pagina in modo che interagiscano tra di loro.

2.1.1 Search

La componente di ricerca è la più complessa, dato che incorpora la logica di strutturazione della tabella (figura 7) con le opzioni di ordinamento e di paginazione e la generazione della form per la ricerca delle entità (figura 8). Offre inoltre un ampio margine di personalizzazione sulla visualizzazione

[3] `https://vuejs.org/v2/guide/components-slots.html`

dei dati e sulle azioni che è possibile eseguire su di essi, sovrascrivendo la configurazione di Portofino.

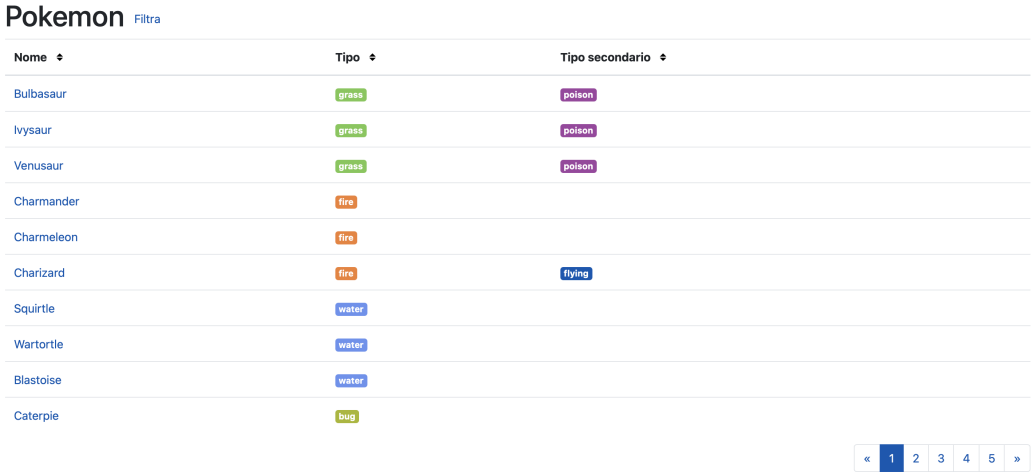


Figura 7: Tabella generata dalla libreria presa dal progetto di demo

Il filtro di ricerca viene generato, come per le componenti di creazione e modifica, mappando il tipo di dato che si desidera filtrare su un campo di inserimento ad hoc, con l'aggiunta di alcuni campi specifici per la ricerca, come la selezione multipla e il campo di ricerca avanzata su parole chiave, che consente di specificare se la parola ricercata deve iniziare, contenere, terminare o essere identica a quella inserita.

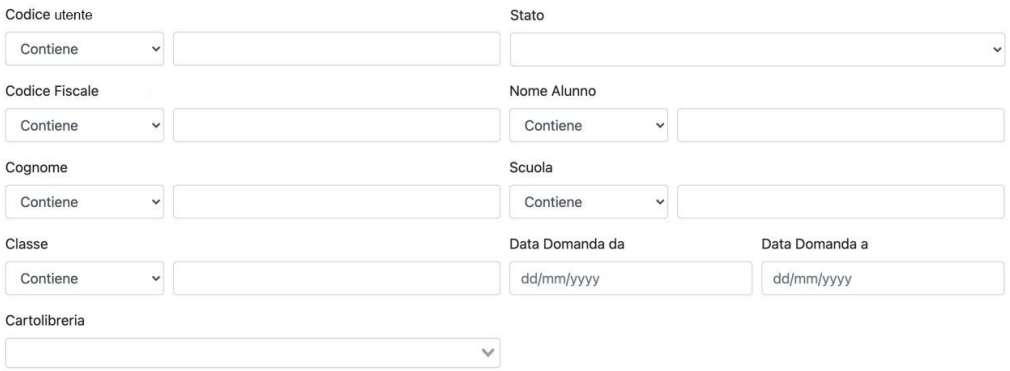


Figura 8: Form di ricerca di un progetto che utilizza questa libreria

Per la creazione della tabella in figura 7 sono sufficienti poche righe di codice, come si può vedere nella codice riportato nella figura 9. Il codice riportato rappresenta un vue-sfc (vue - single file component), ossia un

tipo di file che, tramite un *transpiler*[4], può essere convertito in codice JavaScript consumabile dal browser. Possiamo notare che questo file contiene principalmente due parti.

```
1  <template>
2    <crud-search :action="action" :title="true" :search="true">
3      <template #cell(name)="{ item, value }">
4        <n-link :to="`/pokemons/${item.key}`">{{ value }}</n-link>
5      </template>
6      <template #cell(primary_type)="{ value }">
7        <span class="badge badge-primary" :style="{ backgroundcolor: pillcolor(value) }">
8          {{ value }}
9        </span>
10     </template>
11     <!-- ... -->
12   </crud-search>
13 </template>
14 <script>
15 import Portofino from "@manydesigns/portofino";
16 import TypesColor from "~/utils/TypesColor";
17 export default {
18   async asyncData() {
19     const action = await Portofino.getAction("pokemons");
20     return { action };
21   },
22   methods: {
23     pillcolor: (type) => TypesColor[type],
24   },
25 };
26 </script>
```

Figura 9: Codice sorgente per ottenere la tabella mostrata in figura 7

La prima parte definisce la struttura del componente, il primo tag che incontriamo alla riga 2 è il tag *crud-search*, ossia il componente della libreria che stiamo discutendo! Gli attributi passati al componente servono per personalizzare il componente e per passargli i dati che gli sono indispensabili per generare il componente, in questo caso il primo attributo è la definizione di action che abbiamo precedentemente ottenuto, mentre gli altri due abilitano rispettivamente la visibilità del titolo del componente e del form di ricerca. All'interno del tag abbiamo due tag template figli, questi consentono di utilizzare una funzionalità molto potente di VueJS, gli slot, che ci consentono di definire delle porzioni di codice che andranno a sostituirne altre già presenti nel nostro componente. In questa istanza i due slot sostituiscono la cella della tabella contenente il nome con una che contiene un collegamento alla pagina con il dettaglio di quella entità alla linea 4 e la cella contenente il tipo principale del Pokemon in modo da mostrare il nome del tipo contenuto in una pill colorata alla riga 7.

[4]Uno strumento che prende in input del codice sorgente e lo trasforma in un altro sintatticamente differente ma la cui esecuzione produce lo stesso risultato

La seconda parte è il codice JavaScript che definisce la logica business del componente, il metodo asyncData si occupa di ottenere la definizione della action di portofino-js[12] [5], necessario al componente, mentre il metodo *pillColor*, si occupa semplicemente di ritornare un colore dato un tipo di Pokemon.

Ovviamente non è necessario che le componenti fornite da questa libreria siano utilizzate come primo elemento del componente, ma possono essere annidate e ne può essere utilizzata più di una nella stessa pagina per creare viste composte come nella figura 10. In questa figura nel tag *crud-detail*, che discuteremo in seguito, è presente l'attributo *fields* che permette di personalizzare vari aspetti degli attributi della nostra entità, in questo esempio si occupa esclusivamente di nascondere l'attributo *id* dalla vista, ma è anche possibile definire altri attributi non presenti nell'entità oppure alterare il testo dell'etichetta o disabilitarne la possibilità di ordinare le entità della vista.

```
<template>
  <div class="container-fluid">
    <div v-if="!$fetchState.pending" class="row">
      <div class="col-md-4">
        <crud-read :entity="$route.params.id"
                   :action="action"
                   :fields="{ id: { display: false } }"/>
      </div>
      <div class="col-md-8">
        <crud-search :action="trainerpoke">
          <template #cell(pokemon)="{ item }">
            <n-link :to="`/pokemons/${item.key}`">
              {{ item.getDisplayValue("pokemon") }}
            </n-link>
          </template>
        </crud-search>
      </div>
    </div>
  </div>
</template>
```

Figura 10: Codice di una pagina che contiene due delle componenti della libreria

2.1.2 Read

L'elemento di ricerca è una card (figura 11), che mostra, tramite una description-list, gli attributi dell'entità, gestendone il caricamento asincrono e la corretta

[5] `https://manydesigns.github.io/portofino-js/actions`

lettura da parte di uno screen reader, pur consentendo di alterare la visibilità degli attributi.

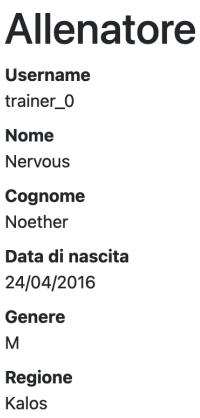


Figura 11: Componente di dettaglio di una entità del progetto di demo

2.1.3 Create e Update

Le componenti di creazione e modifica sono fondamentalmente uguali, entrambe generano un form (figura 12) che mappa i tipi di dato presenti nel database su un tipo input che può essere utilizzato dall'utente per inserire i valori necessari. Il form valida automaticamente i dati inseriti e si assicura che tutti i campi richiesti vengano compilati, fornendo un feedback visivo accessibile anche a persone che utilizzano uno screen reader.



Figura 12: Form di creazione generata dalla libreria

2.2 Personalizzazione

Per quanto riguarda le personalizzazioni grafiche delle componenti, per fare si che la libreria non sia vincolata ad una unica scelta stilistica, ho individuato due strade percorribili. La prima strada prevedeva di avere un'unica struttura del documento personalizzabile tramite la creazione di fogli di stile appositi, mentre la seconda consente allo sviluppatore di creare una componente ad-hoc per definire come viene visualizzata la componente.

Il primo metodo, seppur a primo impatto possa sembrare più semplice, si è rivelato il meno personalizzabile in quanto obbliga l'utente a riscrivere un foglio di stile per emulare quello di altre librerie. Spesso, tuttavia, per pigrizia o mancanza di tempo, questo non viene fatto e si mantiene lo stile di base della libreria, che differirà dal linguaggio stilistico del resto dell'applicazione.

Il secondo metodo invece, delegando all'utente o ad altri sviluppatori che possono pubblicare liberamente le componenti, consente di creare pacchetti di integrazione con librerie grafiche che implementano varie scelte stilistiche o di crearne di completamente personalizzate, strutturate sulle esigenze del progetto, senza nessuna limitazione sulla struttura dell'elemento.

Per le ragioni illustrate nella sezione successiva, e siccome una delle priorità che mi sono posto era mantenere un elevato livello di esperienza per lo sviluppatore, mi è risultato naturale optare per il secondo metodo, seppur tenendo presente che, per agevolarne la diffusione, è necessario ampliare l'ecosistema dei modelli grafici e non limitarsi a Bootstrap 4.

2.3 Model

I *model* rappresentano la raccolta di componenti grafiche necessarie alla libreria che devono essere specificate dallo sviluppatore in fase di registrazione del plugin. Queste possono essere fornite direttamente dalla libreria o importate come dipendenza esterna e inserite nella configurazione della libreria.

In questo modo potrebbero essere sviluppati modelli grafici non accessibili, che distruggerebbero il proposito di questa libreria, ma, come per il problema precedentemente discusso delle personalizzazioni grafiche, si lascia al buon senso dello sviluppatore che andrà a scrivere tali personalizzazioni. Rimane comunque importante notare che anche il resto dell'applicazione deve essere pensato per essere accessibile e che anche con il metodo precedente, nonostante si mantenga sempre la stessa struttura del modello, l'accessibilità dell'intera libreria potrebbe essere invalidata scegliendo, per esempio, colori con uno scarso contrasto tra di loro o font poco leggibili.

Questo approccio, rispetto a quello di fornire componenti statiche personalizzabili tramite CSS, consente di poter utilizzare direttamente librerie

grafiche, con tutte le implicazioni dell'uso di quelle librerie, piuttosto che replicarne lo stile, e consente di essere resilienti ad un aggiornamento dello stile libreria grafica siccome è sufficiente aggiornare la dipendenza, senza dover riscrivere il foglio di stile che replica quello della libreria. Un esempio concreto e molto recente di questo è stato il passaggio Material Design a Material You [18].

2.4 Under the hood

La libreria si basa sulla funzione di Portofino di esporre, tramite API, la definizione degli attributi e delle relazioni con altre entità del sistema che vengono poi gestite in maniera completamente trasparente per generare le viste delle varie componenti esposte. Il ciclo di vita, rappresentato nella figura 13, illustra come vengono caricati ed elaborati i dati utilizzati e in quale momento vengono eseguite le operazioni più significative, in particolare il momento in cui vengono registrati gli *event listener*. È importante avere conoscenza di questi aspetti del ciclo di vita della libreria siccome si potrebbe pensare di emettere un evento prima di completare il render della componente e, siccome il *listener* non sarebbe ancora stato registrato, l'evento andrebbe perduto e non si otterrebbe il risultato desiderato.

I dati necessari per la generazione delle componenti vengono ottenuti direttamente da Portofino che espone le informazioni sulle entità attraverso API RESTful.

La prima serie di dati ottenuti da Portofino sono le definizioni degli attributi e delle relazioni dell'entità su cui si vuole operare, che vengono elaborate in modo da poter essere utilizzate dalla libreria ed eventualmente unite ad altre iniettate dallo sviluppatore e il risultato utilizzato per creare i campi degli attributi delle entità.

La seconda chiamata alle API, presente in ogni componente, fuorché in quello di creazione, ottiene i dati dell'entità da mostrare all'utente.

Solo a questo punto entra in gioco il sistema di render delle componenti che si occupa di iniettare all'interno del modello fornito i dati precedentemente ottenuti e di sostituire gli elementi personalizzabili del modello con i frammenti di template inseriti all'interno dell'istanza della componente, in modo da generare la vista dei dati desiderata dallo sviluppatore.

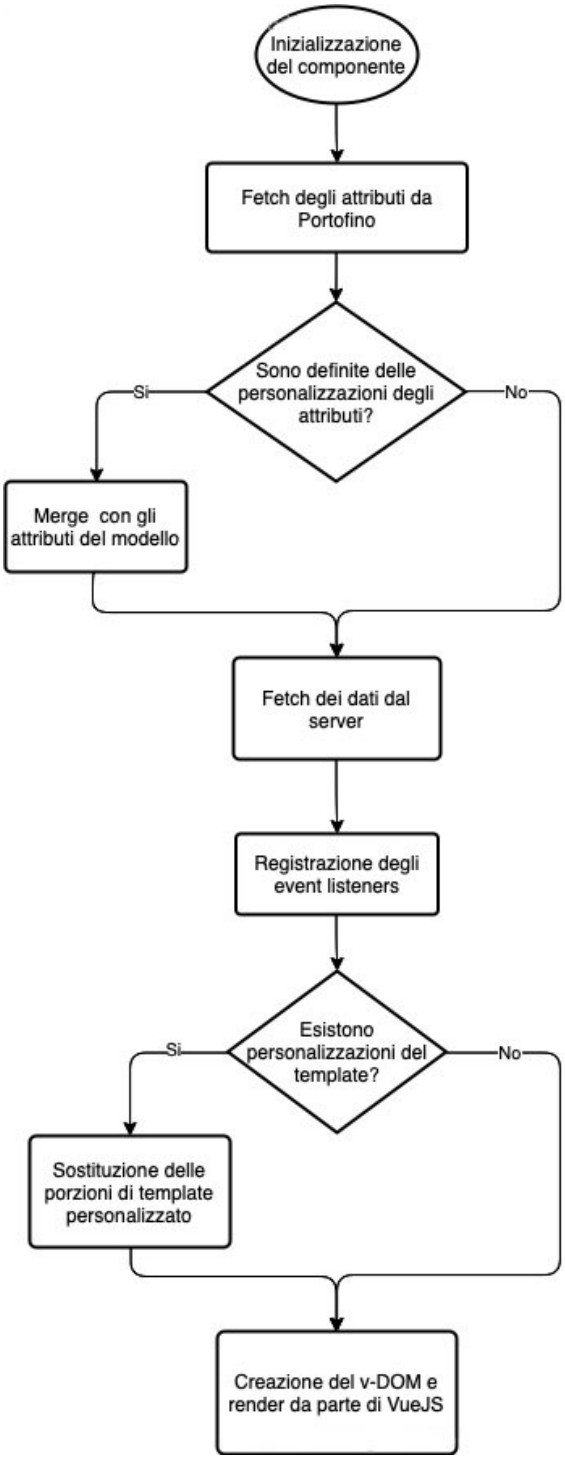


Figura 13: Ciclo di vita della libreria

3 Tecnologie

Per lo sviluppo di questa libreria sono stati necessari molti strumenti creati da altri sviluppatori, non solo per la verifica e i test di accessibilità dell'applicazione ottenuta, ma anche per la creazione delle fondamenta della libreria e per il funzionamento del progetto. Anche se la maggior parte degli strumenti fungono esclusivamente da supporto e possono essere trascurati, altri sono risultati indispensabili strumenti di ausilio, senza i quali lo sviluppo sarebbe stato molto più complicato, forse addirittura impossibile.

3.1 Test accessibilità

Al fine di validare l'effettiva accessibilità delle componenti generate dalla libreria, sono stati utilizzati degli strumenti di validazione automatica di pagine web, che verranno discussi successivamente. Poiché gli strumenti di validazione automatica si limitano alla validazione della fruibilità di un contenuto da parte di strumenti di ausilio, essi ignorano completamente elementi che non sono considerati interagibili per la loro semantica ma sono utilizzati come tali. Per esempio, come abbiamo già visto nel capitolo 1, nelle prime fasi di sviluppo avevo utilizzato delle icone SVG per indicare delle azioni sugli elementi della tabella di ricerca, ma senza inserirle all'interno di un elemento che, per la semantica HTML fosse cliccabile. Per questo motivo, secondo il sistema di validazione automatica, la tabella era perfettamente accessibile, ma per una persona non vedente questi bottoni non esistevano.

Date le suddette problematiche, è evidente che un controllo umano diventa essenziale per testare questi casi particolari. Con il supporto dello Staff del laboratorio per l'accessibilità del SIMAV, è stato possibile effettuare test più accurati sull'effettiva fruibilità delle componenti fornite dalla libreria, fermo restando che le personalizzazioni di ogni componente devono essere pensate per essere accessibili.

3.1.1 AC Checker

AC Checker [3] è uno strumento estremamente potente che riesce a rappresentare visivamente ciò che è possibile leggere con uno screen reader, evidenziando immediatamente i problemi più gravi e gli elementi inaccessibili all'utente. Sfortunatamente questo strumento si limita a fare un'analisi statica sulla pagina HTML, senza eseguire il codice JavaScript, e quindi può esclusivamente analizzare pagine statiche o create dal server. Non è possibile usare questo tool per siti web come quelli che possono utilizzare questa libre-

ria, che fanno uso di tecnologie che generano la pagina sul browser tramite accesso al DOM mediante JavaScript.

3.1.2 WAVE Webaim

WAVE [6] è un'estensione che valida automaticamente le linee guida proposte da WCAG 2.1 e fornisce degli strumenti per poter controllare in modo efficace che il sito analizzato rispetti quanto meno i requisiti tecnici di contrasto e di leggibilità degli elementi presenti. Questo strumento ha evidenziato in particolare che il colore primario della libreria Bootstrap non ha un contrasto adeguato su uno sfondo bianco, per cui è stato necessario renderlo più scuro o, in alternativa, aumentarne la dimensione e renderlo in grassetto. Come si può notare dalla figura 14, una volta corretti i problemi dovuti alle scelte stilistiche di Bootstrap, WAVE non rileva nessuna inadempienza alle linee guida WCAG 2.1.

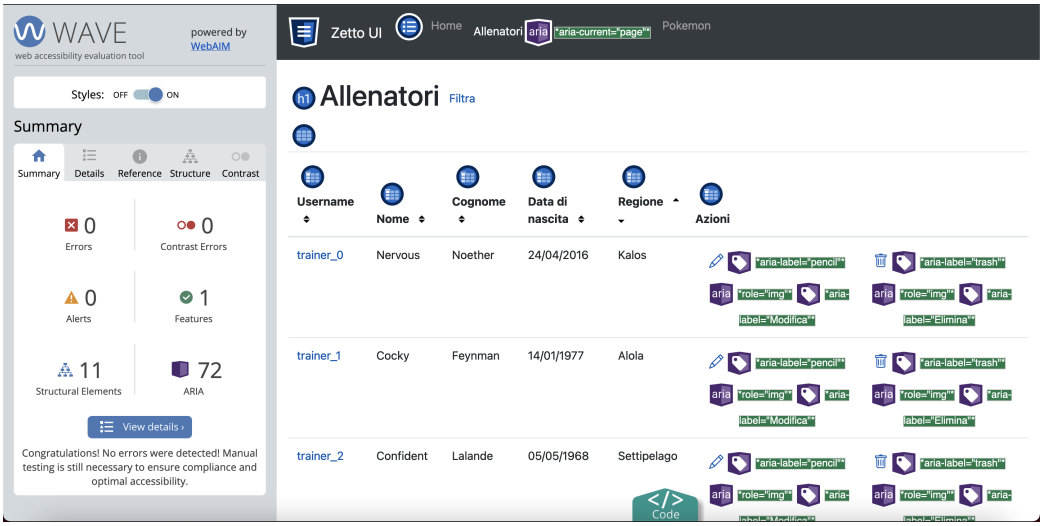


Figura 14: Risultato analisi WAVE

3.1.3 Voiceover

Voiceover [2] è lo screen reader creato da Apple e inserito in ogni dispositivo dotato di schermo prodotto dalla compagnia di Cupertino; è di facile utilizzo, il che lo ha reso per molti anni la scelta più ovvia per le persone che necessitano di supporto nella lettura dei contenuti. Una volta seguito il tutorial che spiega in maniera molto intuitiva come muoversi tra i contenuti basterà seguire gli eventuali suggerimenti e indicazioni forniti durante l'uso per muoversi nelle pagine. Grazie a Voiceover è possibile controllare che una pagina

sia effettivamente navigabile e accessibile e se tutti i contenuti visibili sono percettibili anche da una persona non vedente. Ovviamente la percezione dello sviluppatore senza deficit visivo sarà diversa da quella di una persona abituata a utilizzare questo tipo di strumento e questo tipo di navigazione ma che non ha idea di come sia strutturato il contenuto da un punto di vista del layout.

3.1.4 Lighthouse

Lighthouse [9] è uno strumento prodotto da Google che si occupa di eseguire analisi relative alla qualità delle applicazioni web, principalmente relative alla presenza di metadati e alla velocità di caricamento. In aggiunta ha anche un'ottima sezione per eseguire l'analisi dell'accessibilità delle pagine che illustra anche come effettuare quei controlli manuali che sono ritenuti necessari. I risultati dei test di accessibilità eseguiti sul progetto di demo, come si può vedere dalla figura 15, ritornano un risultato perfetto (100), seppur sottolineino anche loro l'importanza dei test manuali.

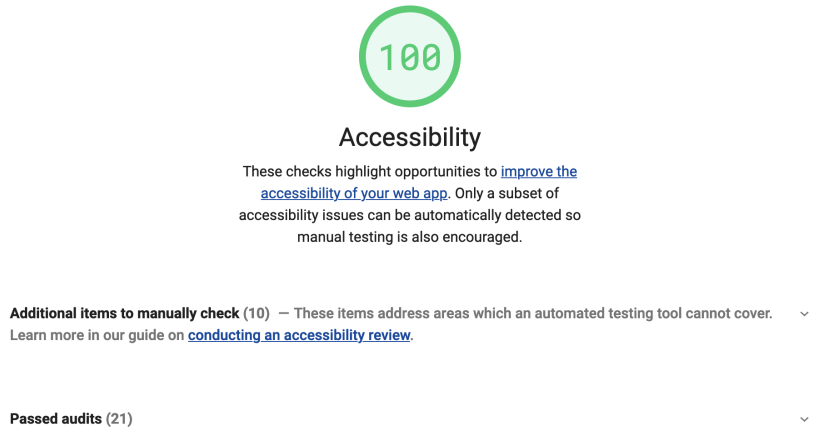


Figura 15: Risultato dell'analisi con Google Lighthouse

3.2 Tecnologie

La libreria da me sviluppata è scritta in TypeScript [13], un linguaggio di programmazione, creato da Microsoft che introduce tipizzazione forte in JavaScript, per VueJs.

VueJs [21] è un framework JavaScript creato da Evan You per la creazione di applicazioni web, che spicca particolarmente nel panorama dei framework per lo sviluppo web tra i due colossi React [5] di Facebook e Angular [8]

di Google per la sua leggerezza e semplicità d'uso, che lo portano ad avere volumi d'uso paragonabili ai due precedentemente citati [19].

Per la gestione dei dati la libreria si poggia su Portofino, un framework open-source scritto in Java per la creazione di API restful, a partire dalla definizione dell'entità presente sul database. Portofino consente di generare un backend automaticamente a partire da un database preesistente. Dalla quinta versione, Portofino ha abbandonato il supporto alla generazione automatica delle pagine JSP, creando la richiesta per una libreria come questa per la generazione, quasi automatica, delle interfacce utente.

3.3 Automazione

Un argomento spesso sottovalutato è la gestione del ciclo di vita del progetto che, essendo aperto a contribuzioni pubbliche su GitHub, risulta ancora più oneroso, in quanto deve consentire ad un nuovo contributore di poter comprendere la struttura e l'organizzazione dell'intero codice. Uno strumento particolarmente utile sono state le GitHub Actions [7], ossia uno strumento di Continuos Integration (CI) che consente, in base agli eventi scatenati dal repository, di eseguire delle azioni quali la compilazione, l'esecuzione dei test e il rilascio del pacchetto su NPM. Questo sistema, offerto senza limiti da GitHub per i progetti open-source, oltre a consentire l'automazione delle procedure ripetitive come gli altri sistemi di CI, consente di riutilizzare azioni già pronte, favorendo il riuso del codice e semplificando la creazione di nuove azioni.

Al fine di semplificare la comprensione dello stato e dei cambiamenti della libreria ho deciso di seguire due convenzioni per la gestione del versionamento della libreria e per il tracciamento dei cambiamenti, queste convenzioni sono rispettivamente il *semantic versioning* [16] e il *conventional commit* [1]

3.3.1 Semantic versioning

Il versionamento semantico è un sistema di versionamento che prevede l'utilizzo di tre numeri divisi da un punto che danno informazione all'utente della compatibilità del software con altre versioni dello stesso software. Questa stringa è strutturata come *MAJOR.MINOR.PATCH* e i tre numeri, quando incrementati, rappresentano rispettivamente:

- MAJOR: indica che è presente almeno un cambiamento che rompe la compatibilità con le versioni precedenti.
- MINOR: indica un rilascio in cui vengono introdotte nuova funzionalità.

- PATCH: indica un fix che non comporta modifiche nel comportamento del software.

3.3.2 Conventional commit

A questo punto possiamo sapere se una versione della libreria è compatibile con un'altra, ma non sappiamo dire cosa "rompe" effettivamente la compatibilità, il che rende complicato il passaggio tra una major release e un'altra. Il sistema di conventional commit è stato sviluppato dal team di Google che si occupa di mantenere Angular per definire uno standard per il formato dei messaggi di commit e generare da essi il file di changelog. I messaggi per essere validi devono rispettare la seguente espressione regolare `^(\w+)(\(\w+\))?: ?(\w+)` in cui i gruppi rappresentano rispettivamente: la tipologia di modifica che apporta quel determinato commit, lo scope della modifica e, infine, il messaggio del commit. Una spiegazione più dettagliata è disponibile sulla specifica tecnica dello standard, consultabile al link `https://github.com/angular/angular/blob/master/CONTRIBUTING.md#-commit-message-format`

4 Conclusioni

L'accessibilità è un requisito divenuto obbligatorio per le applicazioni della Pubblica Amministrazione che, dal 23 settembre 2020, ha l'obbligo di garantire l'accessibilità delle piattaforme utilizzate per erogare i suoi servizi al cittadino. E' al contempo un requisito che sta diventando sempre più fondamentale anche per i privati che vogliano estendere il proprio mercato verso nuove fasce di utenti quali persone con disabilità o anziane.

La realizzazione di questa prova finale è un primo ma fondamentale passo per la realizzazione di interfacce utente inclusive che favoriscono l'interazione per gli utenti con disabilità. Come abbiamo argomentato nei capitoli precedenti, esistono una serie di difficoltà riconosciute che rendono una applicazione non accessibile o poco usabile. Durante la prova finale abbiamo realizzato delle componenti che superano queste difficoltà e che sono facilmente riutilizzabili all'interno dei propri applicativi. Questo fa si che lo sviluppatore possa realizzare velocemente delle interfacce che siano al contempo potenti e complesse, ma che non limitino la loro fruizione da parte di utenti con disabilità.

Congiuntamente all'uso di un framework quale ManyDesigns Portofino che permette di realizzare applicazioni a partire da una base dati, la libreria permette allo sviluppatore di non doversi occupare né della creazione di servizi di backend, né della realizzazione di pagine di frontend sia da un punto di vista funzionale, sia in un'ottica di accessibilità.

Ovviamente sono consapevole, ed è necessario che ogni sviluppatore che si occupa di accessibilità faccia lo stesso, che una libreria di componenti accessibili non rende automaticamente accessibile un'applicazione. La libreria è un primo passo e una facilitazione che rende lo sviluppo più veloce, ma, al contempo, è necessario continuare a verificare che la creazione dei propri applicativi mantenga un'attenzione particolare ai temi dell'accessibilità con la stessa cura che si mantiene per la veste grafica e l'esperienza utente. L'unico modo per garantire la completa accessibilità è il controllo umano, la libreria elimina gli errori più comuni, e quindi permette di partire da una buona base per effettuare i test con persone con disabilità. I costi per le verifiche umane sono spesso proibitivi per la maggioranza dei progetti. AgID ha suggerito la creazione di un canale di feedback diretto e continuo con l'utente che accede all'applicazione in maniera atipica per verificare e migliorare la fruibilità dei contenuti. Trattandosi di una libreria, che va mantenuta nel tempo, queste segnalazioni aiuteranno a farla progredire e a beneficiarne saranno tutte le applicazioni che ne fanno uso.

Al momento della stesura di questo documento la libreria è in uso su tre progetti sviluppati da ManyDesigns s.r.l. e ho ricevuto ottimi feedback dagli sviluppatori che la hanno utilizzata e che hanno riscontrato (1) una netta riduzione dei tempi di sviluppo e (2) un miglioramento della qualità complessiva sia del software prodotto in termini di accessibilità, usabilità e qualità grafica complessiva. Grazie a questa libreria le applicazioni risultanti sono riuscite a mantenere una esperienza utente e un design consistente tra le varie pagine, seppur fosse necessario mostrare dati eterogenei tra loro, in maniera rapida, veloce e consistente.

4.1 Sviluppi futuri

Come per ogni applicazione lo sviluppo potrebbe perfezionarsi ed espandersi all'infinito, per cui è fondamentale determinare un MVP (Minimum Viable Product) e determinare le funzionalità e migliorie da introdurre nel medio-breve termine. In questo caso ho individuato un paio di funzionalità rilevanti che intendo introdurre a seguito di questo percorso per fornire uno strumento utile e *free as in freedom* alla comunità degli sviluppatori.

4.1.1 Accessibilità per le applicazioni mobili

L'accessibilità delle applicazioni web è al giorno d'oggi un argomento su cui si inizia a prestare attenzione, ma quello che ancora non è stato affrontato è l'accessibilità delle applicazioni mobili. Gli smartphone sono un canale che viene sempre più utilizzato per accedere ai servizi e necessitano della stessa attenzione delle applicazioni web.

Le componenti realizzate nella libreria possono essere utilizzate anche per la realizzazione di applicazioni mobili utilizzando framework per la realizzazione di applicazioni ibride[6] quali Ionic[7].

Uno dei prossimi sviluppi sarà di utilizzare la libreria per realizzare un'applicazione mobile su un caso d'uso reale presso ManyDesigns sia per ambiente Ios (iPhone) che per Android.

[6]Con applicazione ibrida intendiamo un'applicazione che utilizza come interfaccia verso l'utente un browser headless.

[7]https://ionicframework.com/

4.1.2 Nuovi modelli

Come accennato nella sezione 2.2 al fine di agevolare l'adozione della libreria ritengo che sia fondamentale fornire il supporto alle librerie grafiche più comuni, iniziando con:

- Vuetify - `https://vuetifyjs.com/`.
- antd - `https://antdv.com`.
- Buefy - `https://buefy.org`.

4.1.3 Upgrade dello stack tecnologico

Recentemente è stata rilasciata la terza versione di VueJS che con il tempo andrà a sostituire la versione corrente. Siccome questa versione con il tempo andrà a sostituire quella attualmente un uso occorre aggiornare la libreria in modo che sia compatibile con la nuova versione.

Una novità introdotta da VueJS 3 è il supporto a JSX, ossia il sistema di definizione del DOM utilizzata da ReactJS[5]. Questa nuova funzione renderebbe relativamente semplice introdurre il supporto anche a ReactJS, supportando un questo modo due dei tre framework per lo sviluppo front-end più utilizzati.

4.1.4 Integrazione in Portofino

In accordo con il core mantainer di Portofino, Alessio Stalla, nella prossima versione del framework verrà sostituito il frontend attuale con una nuova versione basata su questa libreria e sarà inserita direttamente nell'archetipo pubblicato su Maven Central[8].

[8] https://search.maven.org/

Riferimenti bibliografici

[1] Conventional commit. Homepage: https://www.conventionalcommits.org/en/v1.0.0/.

[2] Apple. Apple VoiceOver. Available at: https://www.apple.com/it/accessibility/vision/.

[3] Inclusive Design Research Centre. Web Accessibility Checker. Available at: https://achecker.achecks.ca/checker/index.php.

[4] La Camera dei deputati ed il Senato della Repubblica. Legge 9 gennaio 2004. Available at: https://www.gazzettaufficiale.it/eli/id/2004/01/17/004G0015/sg.

[5] Facebook. React JS. Homepage: https://reactjs.org.

[6] Institute for Disability Research Policy and Practice. WAVE® Web Accessibility Evaluation Tool. Available at: https://wave.webaim.org.

[7] GitHub. GitHub Actions. Homepage: https://github.com/features/actions.

[8] Google. Angular. Homepage: https://angular.io.

[9] Google. Google Lighthouse. Available at: https://developers.google.com/web/tools/lighthouse.

[10] Steve Krug. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. New Riders Publishing, USA, 3rd edition, 2014.

[11] ManyDesigns. Portofino. Homepage: https://portofino.manydesigns.com/.

[12] Manydesigns Mattia Dapino. Portofino-js. Homepage: https://github.com/manydesigns/portofino-js/.

[13] Microsoft. Typescritp. Homepage: https://www.typescriptlang.org.

[14] AgID Agenzia per l'italia digitale. Accessibilità. Available at: https://www.agid.gov.it/it/design-servizi/accessibilita.

[15] AgID Agenzia per l'italia digitale. Linee guida sull'accessibilità degli strumenti informatici. Available at: https://docs.italia.it/AgID/documenti-in-consultazione/lg-accessibilita-docs/it/stabile/index.html.

[16] Tom Preston-Werner. Semantic Versioning. Homepage: https://semver.org.

[17] Anthony T. The Aesthetic-Accessibility Paradox. November 2019.

[18] Material Design Team. Unveiling Material You. May 2021.

[19] Tkrotoff. Front-end frameworks popularity. Gist: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190.

[20] W3C. Web Content Accessibility Guidelines (WCAG) Overview. Last access October 2021. Available at: https://www.w3.org/WAI/standards-guidelines/wcag/.

[21] Evan You. Vue JS. Homepage: https://vuejs.org.

Elenco delle figure

1 Collegamento presente sulla pagina delle risposte di Google per saltare al contenuto 6
2 Esempio di bottoni identici, ma in un caso (sinistra) non accessibili 7
3 Due form con contrasti diversi 8
4 Diversi font divisi tra accessibili e non 8
5 Due testi con diversi allineamenti del testo 9
6 Tabella che illustra il contrasto tra vari colori 9
7 Tabella generata dalla libreria presa dal progetto di demo . . . 11
8 Form di ricerca di un progetto che utilizza questa libreria . . . 11
9 Codice sorgente per ottenere la tabella mostrata in figura 7 . . 12
10 Codice di una pagina che contiene due delle componenti della libreria 13
11 Componente di dettaglio di una entità del progetto di demo . 14
12 Form di creazione generata dalla libreria 14
13 Ciclo di vita della libreria 17
14 Risultato analisi WAVE 19
15 Risultato dell'analisi con Google Lighthouse 20