UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Corso di laurea in INFORMATICA

Implementazione di un minisito lato front-end per la sponsorizzazione di un software di tracciabilità nell'ambito farmaceutico e/o parafarmaceutico

Candidata:
Cristina Corghi

Relatore:
Prof. Riccardo Martoglia

Anno accademico 2021-2022

Indice

Elenco delle Figure

Ringraziamenti

Innanzitutto vorrei ringraziare il Professore Riccardo Martoglia per la disponibilità e l'essenziale aiuto nella stesura di questa tesi.

Ringrazio i miei genitori e la mia famiglia per la pazienza e il supporto che mi hanno sempre dato in questi anni di studio e fatica.

Un ringraziamento va a Davide, il mio tutor, per avermi supportato e sopportato durante il periodo di tirocinio, per avermi insegnato con pazienza, e per avermi sempre aiutato quando non ci saltavo fuori.

Ringrazio tutte le mie amiche, che mi hanno sempre aiutata ad andare avanti anche quando volevo mollare, mi hanno consolata e supportata come nessuno avrebbe potuto fare.

Infine, un ringraziamento speciale va a Luca, per avermi supportata nei momenti più bui e per avermi sempre aiutata nello studio; è grazie a lui che sono arrivata fino a qui ed è a lui che voglio dedicare questa tesi.

Introduzione

L'obiettivo del presente elaborato è quello di descrivere, nella maniera più chiara possibile, la realizzazione di un minisito per la sponsorizzazione di un nuovo software nell'ambito farmaceutico e/o parafarmaceutico.

L'azienda richiedente il sito ha espresso il desiderio di implementare un sito che da una parte sponsorizzasse il nuovo software, e dall'altra che invitasse l'utente a vederlo ad una mostra dedicata. Il sito, quindi, sarebbe dovuto essere moderno, affascinante, che catturasse l'attenzione dell'utente. E per realizzare queste richieste è stato necessario ricorrere a strumenti che enfatizzassero l'interfaccia utente, ma al contempo che l'applicazione fosse veloce ed efficiente.

Il Capitolo 1 prevede l'esposizione dell'obiettivo/contesto della tesi.

Nella parte I sono descritti i vari framework utilizzati per la realizzazione del progetto, in particolare:

- Nel Capitolo 2 si tratta di Vue.js, un framework JavaScript per lo sviluppo front-end.
- Il Capitolo 3 è dedicato a Nuxt 3, che è il framework ibrido Vue, il cui compito principale è quello di sviluppare applicazioni con linguaggio JavaScript, sia lato Client sia lato Server, introducendo il concetto di Server-Side Rendering.
- Nel Capitolo 4 si approfondisce GreenSock Animation Platform (GSAP), che è un animation framework scritto in Javascript per la realizzazione di animazioni.
- E infine, il Capitolo 5 è riservato a Windi CSS, un framework css che permette di creare rapidamente siti web senza mai uscire dal template.

Concludendo, la parte II è dedicata ai capitoli di progettazione e implementazione del sito.

Capitolo 1

Obiettivo della tesi

Il presente capitolo ha lo scopo di descrivere l'obiettivo/contesto della tesi.

L'azienda in cui è stato svolto il periodo di tirocinio si chiama APVD srl di Carpi (MO), e opera nel settore della comunicazione pubblicitaria. Progetti di comunicazione, pianificazione mezzi, realizzazioni grafiche e servizi fotografici, web design, elaborazioni digitali, prestampa, stampa offset tradizionale e digitale sono tutti servizi realizzati internamente.

In modo particolare:

- il reparto Web si occupa della creazione di siti, e-commerce e app;
- la parte Grafica si occupa di virtual design, art direction, corporate identity, brochure e folder, advertising e packaging;
- la parte di Fotografia, che comprende shooting, editing e fotoritocco, fashion, industrial, still life, food e portrait, dispone di uno studio fotografico interno;
- il centro di Stampa Digitale, in cui è possibile stampare in grande e piccolo formato, dispone di un supporto a carte speciali, dati variabili, personalizzazioni, finishing e nobilitazioni;
- infine un altro ruolo importante che ricopre l'azienda è la produzione di Video, sia aziendali che pubblicitari, in particolare si occupano di: sceneggiatura, pre e post produzione, shooting, editing, motion graphic, animazione, show reel e tutorial.

Il reparto Web è quello in cui è stato svolto il periodo di tirocinio, e lo scopo di esso, e su cui poi è stata costruita la presente tesi, è stato quello di sviluppare un sito lato front-end (richiesto da un cliente di APVD), con lo scopo di sponsorizzare un nuovo software nell'ambito farmaceutico e/o parafarmaceutico.

Uno dei problemi principali in questo settore (e in generale in ogni settore che richieda un blister e una vendita di materiale che ha prezzo e caratteristiche differenti per nazione), è la tracciabilità dei prodotti. Può capitare infatti che un medicinale venga venduto ad una nazione e che qualcuno lo prenda per poi esportarlo

e venderlo ad un'altra (dove magari ha un prezzo differente o delle caratteristiche differenti).

L'azienda richiedente il sito ha automatizzato parecchio la tracciabilità, andando ad inserire un QR per ogni cellula (slot del blister) che, una volta richiamato, va ad interrogare un servizio di tracciamento e risponde con il tracciato dei movimenti. Tecnicamente si tratta in parte di un SaaS (Software as a Service, che è un servizio di cloud computing che offre agli utenti finali un'applicazione cloud, munita di piattaforme e dell'infrastruttura IT che la supportano, tramite un browser web [1]), e in parte di un macchinario da inserire nella linea di produzione che crea e stampa i QR.

Il tirocinio è iniziato che il sito era già in via di sviluppo, in particolare tutta la parte back-end era già stata realizzata. I requisiti che sono stati richiesti erano sostanzialmente tutta la parte front-end del sito, quindi in particolare:

- le cinque pagine richieste;
- le animazioni.

Tutto questo sarà approfondito nella Parte II del presente elaborato.

Parte I - Tecnologie utilizzate

Capitolo 2

Vue.js

Vue.js, come cita la documentazione ufficiale, "è un framework accessibile, performante e versatile per la creazione di interfacce utente web. Fornisce un modello di programmazione dichiarativo e basato su componenti che consente di sviluppare in modo efficiente interfacce utente, semplici o complesse." [2]

Vue.js è stato utilizzato nel sito per lo sviluppo front-end. Il suo utilizzo ha permesso di estendere l'HTML, creare modelli e componenti riutilizzabili, il che lo rende un framework leggero e reattivo.

- **Accessibile**: La missione di Vue è stata quella di essere un framework accessibile, che chiunque potesse imparare rapidamente. Si basa su HTML, CSS e JavaScript standard con API intuitive e documentazione di facile comprensione.
- **Versatile**: Vue può essere collegato ad un varietà di strumenti per risultati migliori e inoltre fornisce un sistema di installazione dei plugin. [3]

Le due caratteristiche principali di Vue sono:

- **Rendering dichiarativo**: Il rendering dichiarativo in Vue consente di eseguire il rendering dei dati sul DOM utilizzando una semplice sintassi del modello.
- **Reattività**: È una delle caratteristiche più distintive di Vue. Gli stati dei componenti sono oggetti JavaScript reattivi. Quando vengono modificati, la vista si aggiorna. Ciò rende la gestione dello stato semplice e intuitiva. [4]

2.1 Il Framework Progressivo

Perché Vue.js è chiamato "il framework progressivo"? L'idea del progressivo è che si inizia con qualcosa di piccolo e si costruisce gradualmente qualcosa di più grande. È quello che si può fare con Vue.js: si può iniziare con qualcosa di veramente piccolo, entrare nel proprio progetto, per poi farlo crescere gradualmente. Quindi normalmente, quando si crea un'applicazione, si esegue il **rendering**. Fondamentalmente, quello che si sta facendo nel rendering è avere dei dati da presentare

come front-end; il livello di visualizzazione delle cose è mostrato nella Figura 2.1. [5]

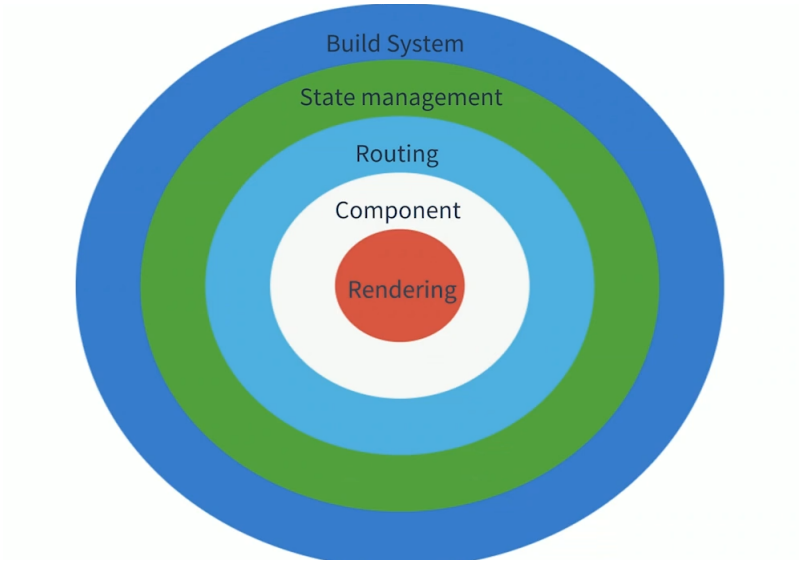

Figura 2.1: Livello di visualizzazione

A seconda del caso d'uso, Vue può essere utilizzato in diversi modi:

- Miglioramento dell'HTML statico senza una fase di compilazione.
- Incorporamento come Web Components in qualsiasi pagina.
- Single-Page Application (SPA).
- Fullstack / Server-Side Rendering (SSR).
- Jamstack / Static Site Generation (SSG).
- Targeting desktop, mobile, WebGL. [6]

2.2 Single-File Components

I componenti globali possono essere sufficienti per molti progetti Vue. Possono funzionare molto bene per applicazioni di piccole e medie dimensioni, in cui JavaScript viene utilizzato solo per migliorare determinate visualizzazioni.

Tuttavia, in progetti più complessi, o quando il frontend è interamente guidato da JavaScript, possono sorgere diversi svantaggi:

- Le definizioni globali impongono nomi univoci per ogni componente. Ciò significa che non possiamo avere due componenti chiamati ProductList, anche

se vengono utilizzati in luoghi diversi dell'applicazione.

- Gli string templates mancano di evidenziazione della sintassi. Inoltre, gli elementi del template, risiedono in un file diverso rispetto al codice del componente. Questo rende più difficile tenerne traccia.
- Non esiste il supporto CSS, il che significa che mentre HTML e JavaScript sono modularizzati in componenti, CSS viene escluso.
- Infine, non esiste un passaggio di compilazione e questo ci limita a utilizzare HTML e JavaScript ES5, piuttosto che preprocessori come Pug e Babel. Quindi, non possiamo sfruttare il JavaScript moderno.

Tutti questi problemi vengono risolti dai Single-File Components, con estensione .vue. Ciò è possibile con strumenti di compilazione come Webpack o Rollup. In Figura 2.2 viene mostrato un esempio di Single-File Component Hello.vue.

```vue
<script>
export default {
  data() {
    return {
      greeting: 'Hello World!'
    }
  }
}
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

Figura 2.2: Single-File Component

Questo file ha tre sezioni:

1. **Il template**: dove risiede il markup del componente.
2. **Lo script**: in cui il file esporta un oggetto, che è l'oggetto opzioni del componente Hello.
3. **Lo stile**: nel quale ci sono tutte le regole CSS, che sono legate allo stile del componente.

In questo modo possiamo leggere e modificare tutti gli aspetti del componente in un unico file.

Da notare che il componente è denominato Hello.vue, che non è un nome di componente proprio, infatti, i nomi dei componenti devono sempre essere composti da più parole. Quindi un nome migliore sarebbe stato HelloWorld o HelloThere. [7]

2.3 API Styles

I componenti Vue possono essere creati in due diversi stili di API: **Options API** e **Composition API**.

Le Options API utilizzano opzioni come "data", "methods" e "mounted". Con la Composition API, abbiamo un unico "setup" hook in cui scriviamo il nostro codice reattivo.

Entrambe le API sono alternative l'una all'altra. Possono essere usate anche insieme, ma sono diverse in termini di aspetto.

Nelle Options API, ci si deve limitare ad un oggetto per configurare un componente con proprietà e metodi; ma in Compositions API, si utilizzano hook diversi per fare le stesse cose. Questa funzione permette di avere il controllo completo sul codice. È possibile inoltre organizzare il proprio codice come si preferisce.

Le Compositions API risolvono due problemi principali che avevano le Options API:

1. Raggruppano parti di codice rilevanti usando gli hooks.
2. Aiutano a riutilizzare il codice in tutta l'applicazione molto facilmente utilizzando i "composables".

Per creare dati reattivi, in Vue si utilizzano gli stati. Queste API utilizzano approcci diversi per creare stati:

- In Options API, è possibile dichiarare un metodo chiamato **data()** nell'oggetto principale. Questo metodo restituisce sempre un oggetto con lo stato di cui si ha bisogno in quel particolare componente. Questo è l'unico posto nel componente in cui si possono aggiungere tutti i tipi di dati di cui si potrebbe aver bisogno.
- In Composition API, è possibile aggiungere dati reattivi usando le funzioni **ref()** e **reactive()**. Quando si chiamano queste funzioni, bisogna solo fornire i valori iniziali. La differenza tra ref() e reactive() è davvero sottile: ref() viene usato per qualsiasi valore primitivo come stringhe, numeri, booleani, mentre reactive() è usato solo per gli oggetti.

Per concludere, molti sviluppatori stanno adottando sempre di più le Composition API per le loro nuove caratteristiche e facilità d'uso, infatti:

- Aiutano a scrivere codice pulito.
- Rendono il codice molto semplice e facilmente leggibile.
- È possibile aggiornare o aggiungere facilmente nuove funzionalità alla propria applicazione.
- È possibile organizzare il proprio componente in diverse sezioni con i relativi codici.
- È possibile creare una funzione componibile per riutilizzare il codice tra diversi componenti. [8]

2.4 Store

Al centro di ogni applicazione Vue c'è lo **store**. Uno "store" è fondamentalmente un contenitore che contiene lo stato dell'applicazione. Un utente che ha effettuato l'accesso è un esempio perfetto di dati che appartengono a uno store.

Ci sono due cose che rendono uno store Vue differente da un oggetto globale:

- Gli store Vue sono reattivi. Quando i componenti Vue recuperano il suo stato, si aggiornano in modo reattivo ed efficiente se lo stato dello store cambia.
- Non è possibile modificare direttamente lo stato dello store. L'unico modo per farlo è eseguire in modo esplicito le mutazioni. [9]

2.4.1 Pinia

Pinia è uno *state management solution*. Prima di essa, Vuex è stato per diverso tempo lo state management solution ufficiale, ma Pinia ora ha preso il suo posto ed è la soluzione consigliata nei nuovi documenti Vue.js.

Che cos'è lo State Management?

Le applicazioni spesso hanno bisogno di accedere agli stessi dati anche su componenti diversi. È qui che entra in gioco uno state management solution come Pinia. Invece di ogni componente che tiene traccia del proprio stato e lo trasmette in giro, Pinia funge da archivio globale, memorizzando lo stato in un unico posto e trasmettendolo ai componenti quando lo richiedono, oltre a fornire meccanismi affinché i componenti lo mutino.

È utile utilizzare Pinia quando il progetto è destinato a crescere, ma non c'è alcun svantaggio nell'utilizzarlo fin da subito.

Di seguito i motivi per i quali è consigliato usare Pinia per lo State Management:

- L'API semplice e ben congegnata di Pinia rende la scrittura degli store familiare quanto la creazione di componenti. Ciò significa che ci sono meno concetti da imparare rispetto a una soluzione come Vuex.
- Con Pinia vengono dedotti tutti i tipi di dati, il che significa che gli stores forniscono un completamento automatico completo e accurato anche in JavaScript.
- Per quei casi in cui il comportamento predefinito di Pinia non è sufficiente, fornisce anche un sistema di plug-in completo con funzionalità come transazioni, sincronizzazione dell'archiviazione locale, ecc.
- Pinia è modulare di default. Ciò significa una migliore organizzazione del codice per gli stores e applicazioni più performanti.
- Infine, Pinia ha un design estremamente leggero.

In conclusione, Pinia è la soluzione di State Management del futuro per Vue.js, e con le sue API ben progettate, l'utilizzo intuitivo e l'ingombro ridotto, è facile comprenderne il motivo. [10]

Capitolo 3

Nuxt 3

Vue.js è inquadrato come framework, tuttavia sarebbe più opportuno inquadrarlo come una libreria, in quanto non fornisce da subito tutte quelle feature necessarie per lo sviluppo di un applicativo completo ma necessita di componenti esterni per gestire routing e altre funzionalità.

Nuxt.js può esser visto come un potenziatore di Vue.js. Il principale compito di Nuxt.js è sviluppare app con codice JavaScript eseguito sia lato Client sia lato Server, introducendo il concetto di SSR (Server-Side Rendering).

Nuxt 3, come cita la documentazione ufficiale, è "il framework ibrido Vue. È open source, e rende lo sviluppo web semplice e potente". Nuxt 3 è stato riprogettato con un core più piccolo e ottimizzato per prestazioni più veloci e una migliore esperienza per gli sviluppatori. L'obiettivo di Nuxt è rendere lo sviluppo web intuitivo e performante. [11]

Nuxt nasce con un file di configurazione chiamato **nuxt.config.js**, il quale permette di sovrascrivere e aggiungere qualsiasi opzione si voglia. È sicuramente il file principale di una qualsiasi applicazione nuxt. Uno degli aspetti più rilevanti che è possibile gestire in questo file riguarda la registrazione di **plugin nuxt**, con possibilità di distinguere cosa è eseguito sul client e cosa sul server.

Questo framework è stato molto utile da utilizzare nel sito, soprattutto per i seguenti aspetti:

- supporta la COMPOSABLE/ directory, che serve per importare automaticamente i composables Vue nella propria applicazione usando gli auto-imports. [12]
- permette di utilizzare Vite, che è un'alternativa no-bundler al webpack creata dall'autore di Vue.js. Utilizzando moduli ES nativi, fornisce un avvio istantaneo del server e un'esperienza di sviluppo HMR velocissima. [13]
- come detto prima, introduce il concetto di Server-Side Rendering (SSR), che è la capacità di un'applicazione di contribuire visualizzando la pagina Web sul server invece che nel browser. Lato server invia una pagina completa-

mente renderizzata al client; per rendere interattiva l'app lato client, Vue deve eseguire la fase di idratazione. Durante questa fase, crea la stessa applicazione Vue che è stata eseguita sul server, abbina ciascun componente ai nodi DOM che deve controllare e collega i listener di eventi DOM. [14]

3.1 Limiti di Vue

Non esiste una best practice per la struttura delle cartelle in Vue. Un'applicazione Vue, di default, fornisce solo "assets" e "components", mentre con l'integrazione di Nuxt si ha la seguente gerarchia:

- **layouts**: Nuxt fornisce un framework di layout personalizzabile che si può utilizzare in tutta l'applicazione, ideale per estrarre modelli di codice o interfacce utente comuni in componenti di layout riutilizzabili.
- **pages**: contiene le viste di livello superiore (in file .vue), utilizzate per generare percorsi.
- **components**: contiene i componenti Vue riutilizzabili.
- **public**: contiene i file statici, che saranno mappati partendo dal server root. Per esempio favicon o i robot.txt.
- **assets**: contiene i file non compilati. Tipicamente immagini, fonts, css.
- **plugins**: contiene i plugin JavaScript, da gestire prima di avviare l'app Vue.
- **middleware**: definisce funzioni JavaScript personalizzate da eseguire prima del rendering di una pagina o di un gruppo di pagine.

Per ottenere questo è necessario lanciare il seguente comando: *npx create-nuxt-app nome-app*. npx è un tool destinato a completare l'esperienza di utilizzo dei pacchetti dal registro npm. Nel nostro caso cerca nei registri npm la presenza di quell'eseguibile, create-nuxt-app, poi avvierà la procedura per creare e configurare un app nuxt.

Un altro problema di Vue è che le sue applicazioni non sono Seo friendly ("SEO è un acronimo per Search Engine Optimization, l'ottimizzazione per i motori di ricerca. Ne fanno parte tutte le strategie attraverso cui viene migliorato il posizionamento di un sito web all'interno dei risultati delle ricerche effettuate sul web"[15]), ciò significa che qualsiasi applicazione sviluppata usando solamente Vue.js, senza alcun tipo di pre-render lato server, potrebbe non essere indicizzata correttamente dai browser. Questo aiuto lo fornisce Nuxt, in quanto è pre-configurato per generare l'applicazione lato server gestendo in modo ottimale e ottimizzando le rotte così da sfruttare al meglio i tag relativi alla SEO.

Un ulteriore problema è che le applicazioni Vue.js sono notoriamente estremamente lente nei caricamenti iniziali e in altre computazioni. Con l'utilizzo di Nuxt è possibile scegliere il render **Universale** o **Statico**.

Per quanto riguarda il rendering Universale, il browser richiede un URL (lato client + lato server) abilitato, il server restituisce al browser una pagina HTML completamente visualizzata. Indipendentemente dal fatto che la pagina sia stata generata in anticipo e memorizzata nella cache o renderizzata al volo, a un certo punto Nuxt ha eseguito il codice JavaScript (Vue.js) in un ambiente server, producendo un documento HTML. Gli utenti ottengono immediatamente il contenuto dell' applicazione, contrariamente al rendering lato client. [16]

Il rendering statico, invece, permette di ottenere tutti i vantaggi di un'app universale senza server. Ciò migliora le prestazioni, nonché la SEO e un migliore supporto offline.

Un ultimo prolema principale che ha Vue è la gestione del router che, al crescere dell'applicazione, può procurare problemi e diventare più difficile da mantenere, spingendo gli sviluppatori a mantenere e gestire diversi file.

Nuxt risolve automaticamente le rotte utilizzando i file all'interno della cartella **pages/**. Dopo aver letto ogni singolo file con estensione .vue dentro la cartella pages, Nuxt genererà automaticamente un file di rotte che potrà essere visualizzato nella cartella .nuxt con il nome di **routes.json**, senza nessun'altra extra configurazione. [17]

3.2 Auto Imports

Nuxt importa automaticamente funzioni di supporto, composable e API Vue da utilizzare nell'applicazione senza importarle esplicitamente. In questo modo esegue il recupero dei dati (data fetching), ottiene l'accesso all'app context e al runtime config, gestisce lo stato e definisce componenti e plug-in. [18]

3.3 Server Engine

Nuxt 3 è alimentato da un nuovo motore server, chiamato **Nitro**. Questo motore ha diversi benefici:

- Supporto multipiattaforma per Node.js, per i browser, ecc.
- Supporto per rotte API.
- Modalità ibrida per siti statici e serverless.

Le **API endpoint** e il **Middleware** del server vengono aggiunti da Nitro che utilizza internamente **h3** (H3 è un framework h(ttp) minimale costruito per prestazioni e portabilità elevate [19]).

Le caratteristiche chiave sono:

- I gestori possono restituire direttamente oggetti/array per una risposta JSON gestita automaticamente.
- I gestori possono restituire le Promises, che saranno attese (sono supportati anche **res.end()** e **next()**).
- Funzioni di supporto per l'analisi del corpo, la gestione dei cookie, i reindirizzamenti, le intestazioni e altro ancora. [20]

3.4 Head Management

Nuxt fornisce buoni valori di default per i meta tag *charset* e *viewport*, ma è possibile sovrascriverli se necessario, nonché personalizzare altri meta tag.

Dentro la funzione *setup*, è possibile chiamare **useHead** con chiavi che corrispondono ai meta tags: *title*, *titleTemplate*, *base*, *script*, *style*, *meta* e *link*, come **htmlAttrs** e **bodyAttrs**. Ci sono anche due scorciatoie, **charset** e **viewport**, che impostano i meta tag precedenti.

In alternativa, si può passare una funzione che restituisce l'oggetto per i metadati reattivi. Un esempio è mostrato nella Figura 3.1.

```
<script setup>
useHead({
  title: 'My App',
  // or, instead:
  // titleTemplate: (title) => `My App - ${title}`,
  viewport: 'width=device-width, initial-scale=1, maximum-scale=1',
  charset: 'utf-8',
  meta: [
    { name: 'description', content: 'My amazing site.' }
  ],
  bodyAttrs: {
    class: 'test'
  }
})
</script>
```

Figura 3.1: Esempio di Head Management

Un'altra opzione interessante è **titleTemplate**, che permette di fornire un modello dinamico per personalizzare il titolo del sito, ad esempio aggiungendo il nome del sito al titolo di ogni pagina. Può essere anche una stringa.

Se si desidera utilizzare una funzione (per il controllo completo), questa non può essere impostata in nuxt.config, ma si consiglia invece di impostarla all'interno di app.vue, dove si applicherà a tutte le pagine del sito. [21]

3.5 Data Fetching

Nuxt fornisce **useFetch**, **useLazyFetch**, **useAsyncData** e **useLazyAsyncData** per gestire il recupero dei dati all'interno dell'applicazione. Queste opzioni lavorano solo con **setup** o **Lifecycle Hooks**.

3.5.1 useFetch()

All'interno delle pagine, componenti e plug-in si può utilizzare useFetch per ottenere un qualsiasi URL. Genera automaticamente una chiave in base all'URL per ragioni di cache e alle opzioni di recupero, oltre a dedurre il tipo di risposta dell'API, come mostrato nell'esempio in Figura 3.2: [22]

```
<script setup>
const { data: count } = await useFetch('/api/count')
</script>

<template>
  Page visits: {{ count }}
</template>
```

Figura 3.2: Esempio di useFetch()

Capitolo 4

GSAP - GreenSock Animation Platform

GreenSock Animation Platform (GSAP) è un animation framework scritto in Javascript, messo a punto in più di 10 anni di sviluppi dai migliori designer. Può gestire praticamente qualsiasi elemento contenuto nel DOM (CSS, SVG, React, canvas, Vue, Angular, oggetti, ecc. . .) e può essere integrato in qualsiasi progetto in quanto svincolato da altre librerie e framework. È estremamente veloce e viene utilizzato da più di 8 milioni di siti e dalle più grandi compagnie al mondo come per esempio Google, Amazon, Coca Cola e EA Sports.

La maggior parte delle animazioni presenti sul sito sono state realizzate con GSAP, come per esempio il menu, i testi, le immagini, i numeri, ecc.

Le caratteristiche di GSAP sono le seguenti:

1. **Velocità e performance**.
2. **Robustezza**: è un'animazione JavaScript creata per i professionisti, anima colori, proprietà CSS, matrici e molto altro. Crea straordinari effetti attivati dallo scorrimento.
3. **Compatibilità**: GSAP va d'accordo con HTML, SVG, React, Vue, Angular, jQuery, Canvas, CSS, nuovi browser, vecchi browser, dispositivi mobili e molto altro ancora. Risolve automaticamente i numerosi problemi di compatibilità.
4. **Anima qualsiasi cosa**: Oltre alla possibilità di animare qualsiasi cosa è possibile creare animazioni nuove.
5. **Leggero ed espandibile**: Modulare, flessibile e ultra efficiente, la sua architettura a plug-in mantiene robusto il motore principale consentendo l'aggiunta di praticamente qualsiasi funzionalità tramite plug-in opzionali. GSAP include un'incredibile quantità di potenza in un pacchetto sorprendentemente piccolo.
6. **Nessuna dipendenza**: GSAP non è basato su strumenti di terze parti come jQuery. Questo approccio riduce al minimo i tempi di caricamento

e massimizza le prestazioni. È un framework monolitico, indipendente da chiunque.

7. **È gratuito**. [23] [24]

4.1 L'oggetto gsap

L'oggetto *gsap* funge da punto di accesso per la maggior parte delle funzionalità di GSAP. È un oggetto generico con vari metodi e proprietà che creano e controllano **Tween** e **Timeline**.

4.1.1 Tween

Una Tween è ciò che fa funzionare tutta l'animazione: è da pensare come un insieme di proprietà ad alte prestazioni. È sufficiente inserire gli obiettivi (gli oggetti che si vuole animare), una durata e tutte le proprietà che si desidera che vengano animate, e quando la testina di riproduzione della Tween si sposta in una nuova posizione, è facile capire quali dovrebbero essere i valori delle proprietà, e a quel punto li applica di conseguenza.

Metodi comuni per creare una Tween:

- **gsap.to()**: è il tipo più comune di animazione; consente di definire i valori di destinazione.
- **gsap.from()**: definisce i valori da cui deve essere animato un oggetto, ovvero i valori iniziali di un'animazione.
- **gsap.fromTo()**: consente di definire i valori iniziali e finali per un'animazione. È una combinazione del metodo from() e to(). [25]

4.1.2 Timeline

Una Timeline è un contenitore per Tween. Anima le Tweens in ordine sequenziale, che non dipende dalla durata della Tween precedente. La Timeline agevola il controllo delle Tweens nel loro insieme e la gestione dei loro tempi.

Una Timeline riguarda unicamente il raggruppamento di cose e il coordinamento del tempo: non imposta mai le proprietà sugli obiettivi (lo gestiscono le Tweens).

Le Timeline possono essere istanziate in questo modo: *gsap.timeline()*.

È possibile concatenare più Tween a una Timeline in due modi diversi, come mostrato nella Figura 4.1 [25]:

```
##Method 1
const tl = gsap.timeline(); // create an instance and assign it a variable
tl.add(); // add tween to timeline
tl.to('element', {});
tl.from('element', {});

##Method 2
gsap.timeline()
    .add() // add tween to timeline
    .to('element', {})
    .from('element', {})
```

Figura 4.1: Metodi per concatenare più Tween a una Timeline

Da notare che l'intera piattaforma GSAP è orientata agli oggetti, in questo modo è possibile creare singole istanze Tween, ma si può semplicemente chiamare .to(), .from( ) o .fromTo() direttamente sull'istanza Timeline per eseguire la medesima operazione in meno passaggi. [26]

Capitolo 5

Windi CSS

Windi CSS è un framework CSS molto diverso dai classici framework ai quali siamo abituati: ha infatti un approccio completamente differente alla creazione di interfacce web. Permette di creare rapidamente siti Web moderni senza mai uscire dal codice HTML.

Windi CSS funziona eseguendo la scansione di tutti i file HTML, componenti JavaScript e qualsiasi altro template, generando gli stili corrispondenti e quindi scrivendoli in un file CSS statico. È veloce, flessibile e affidabile.

I framework tradizionali, tra cui il più famoso è sicuramente Bootstrap, procurano una serie di componenti come pulsanti, card, finestre modali, form che senza dubbio velocizzano il lavoro e la prototipizzazione, ma diventano scomodi e noiosi quando è necessario modificare la loro idea di design. Windi CSS, invece, non fornisce componenti preconfezionati ma classi di utility che permettono di costruire esattamente quello che abbiamo in mente senza abbandonare in continuazione l'HTML. [27]

Nel sito in questione Windi è stato ampiamente utilizzato perché, come viene spiegato più avanti, non ha set di componenti pre-progettati, ma classi di utilità che possono combinarsi tra di loro per ottenere il layout che si desidera. Questo permette di scrivere pochissimo (o niente) CSS, e perciò la realizzazione del sito risulta estremamente ottimizzata ed efficiente.

Le principali caratteristiche di Windi sono:

- **Design Responsive e Mobile-First**: Ciascuna classe utility fornita da Windi CSS prevede una **responsive variants**, che permette di cambiarne il comportamento in base alla dimensione dello schermo del dispositivo. Le varianti sono dei prefissi disponibili su tutte le classi, e ne modificano lo scopo principale. Di default Windi usa un sistema mobile first breakpoint, ciò vuol dire che le utilities senza prefisso (per esempio *text-center*), hanno effetto su tutte le dimensioni dello schermo, mentre le utilities con prefisso (per esempio *md:text-left*), hanno effetto solo dalla dimensione dello schermo specificata in poi. [28]

- **Value Auto-infer**: Poiché Windi CSS genererà solo le utilità CSS che si utilizzano, consente di utilizzare valori arbitrari nelle proprie classi e generare stili corrispondenti in base alla semantica appropriata.
- **Important Prefix**: Si può anteporre a qualsiasi classe di utilità con "!" per impostarle come !important. Questo può essere molto utile quando si desidera ignorare le regole di stile precedenti per una proprietà specifica. [29]
- **Utility-First**: Come è già stato detto, con Windi si da uno stile agli elementi applicando classi preesistenti direttamente nell'HTML. Questo approccio permette di implementare un design personalizzato senza la necessità di scrivere una sola riga di CSS. Ciò porta a diversi benefici:
  - Non c'è più bisogno di inventarsi dei nomi per le classi.
  - Il CSS smetterà di crescere. Infatti grazie a Windi i fogli di stile cesseranno di esistere o comunque avranno poche linee di codice.
  - È più sicuro apportare modifiche. I CSS sono globali e quando si esegue una modifica non si può sapere cosa si stia rompendo; mentre le classi nell'HTML sono locali, quindi si possono cambiare senza preoccuparsi che qualcos'altro si rompa.

Per concludere, la domanda che sorge spontanea è: **perché allora non usare direttamente gli stili in linea?**

Una reazione comune a questo tipo di approccio è chiedersi: "non sono solo stili in linea?" e in un certo senso è così: si stanno applicando gli stili direttamente agli elementi invece di assegnare loro un nome di classe e quindi applicare uno stile a quella classe.

Ma l'uso delle utility classes presenta alcuni importanti vantaggi rispetto agli stili inline:

- **Costruire il design con vincoli**: Usando gli stili in linea, ogni valore è un numero magico. Con le utilities, gli stili si scelgono da un sistema di progettazione predefinito, il che rende molto più semplice creare interfacce utente visivamente coerenti.
- **Responsive design**: Non si possono utilizzare le media queries sugli stili in linea, invece con le utilities di Windi è possibile creare interfacce completamente reattive. [30]

Parte II - Implementazione del minisito

Capitolo 6

Progettazione

6.1 Strategia progettuale

La progettazione di un sito, anche se di piccole dimensioni, necessita di una strategia progettuale ben definita.

Oltre a promuovere il prodotto e dargli importanza con un sito dedicato, l'obiettivo del sito è quello di invitare gli utenti a vedere la presentazione del prodotto ad una fiera dedicata.

Alberatura sito:

- una **homepage**;
- una pagina chiamata **tech solution**, per la descrizione della soluzione tecnica in cui si espone cosa è e come funziona il prodotto;
- una pagina denominata **verify**, su cui far atterrare l'utente che scansiona il codice QR stampato sull'astuccio.
- una pagina **contacts** in cui vengono descritte le aziende e dove è presente una form che serve per la raccolta dei contatti;
- e infine la pagina **discover** in cui è presente il video di presentazione del prodotto e il calendario in cui è possibile prenotarsi alla fiera.

I passi progettuali per la realizzazione del sito sono stati i seguenti:

1. Iniziale documentazione di tutte le tecnologie descritte nella Parte I.
2. Realizzazione della homepage.
3. Header e menu.
4. Pagina tech solution.
5. Pagina verify.
6. Pagina contacts.
7. Pagina discover.

6.2 Homepage

La homepage è una pagina che (come tutte le altre), è realizzata grazie all'aggregazione di vari componenti. I componenti sono file .vue molto utili in quanto è possibile riutilizzarli in qualsiasi parte del programma.

In questo caso l'homepage è composta da 10 componenti:

1. **HorizontalPanels**: è il componente che racchiude tutti gli altri e che implementa lo scroll orizzontale del sito per i dispositivi di medie e grandi dimensioni, mentre lo scroll verticale per i dispositivi di piccole dimensioni come gli smartphone. Lo scroll è stato implementato con la proprietà ScrollTrigger di GSAP, che crea animazioni basate sullo scorrimento con pochissime linee di codice.

2. **PanelIntro**: è il componente in Figura 6.1 in cui:

   - è presente l'immagine del QR code che gli utenti possono scansionare e venire direttamente reindirizzati alla pagina verify, oppure possono cliccare il link "Verify Now".
   - le frecce sono a loro volta dei componenti (Vector), che si muovono a velocità costante all'infinito in orizzontale da desktop e in verticale da mobile.
   - i testi sono a loro volta dei componenti (PanelInfoText) che compaiono allo scorrimento del mouse.
   - la foto del blister è statica, mentre il testo circolare si comporta come le frecce: ruota in senso orario all'infinito. Questo meccanismo è stato implementato con le Props di Vue.js (spiegate più avanti).
   - infine, la scritta grande "Blister Track and Trace" è realizzata con gsap allo scroll. Parte da sinistra e, all'avanzamento dello scroll, la scritta si sposta sempre verso sinistra fino a che la visualizzazione dell'interfaccia non cambia.

Figura 6.1: Componente PanelIntro

3. **PanelStepTitle**: questo componente viene utilizzato 5 volte nella homepage, ed ha la struttura come nella Figura 6.2. Il numero è un'immagine svg con un'animazione gsap chiamata DrawSVGPlugin. Il testo invece è anch'esso un'animazione gsap che compare allo scroll del mouse.

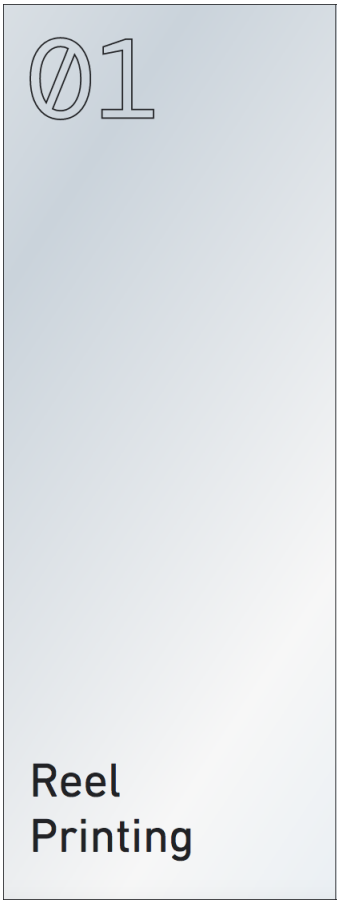

Figura 6.2: Componente PanelStepTitle

4. **PanelG600, PanelQA e PanelReels**: sono componenti come mostrato in Figura 6.3 e Figura 6.4, composti da immagini e testi che compaiono allo scroll.

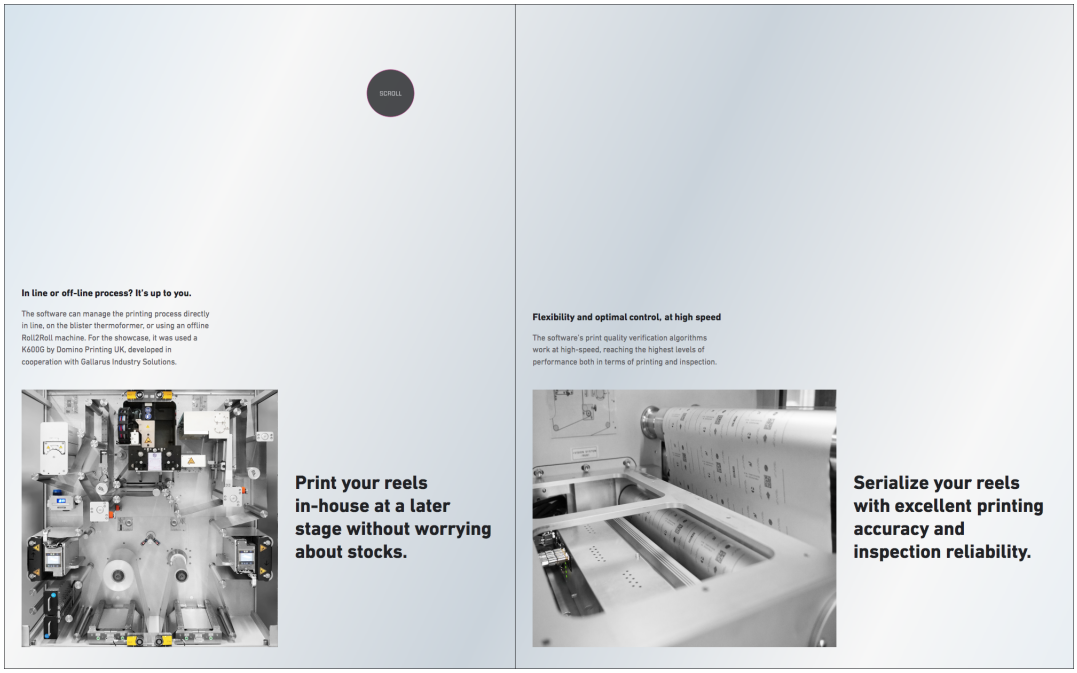

Figura 6.3: Componente PanelG600 e PanelQA

Figura 6.4: Componente PanelReels

5. **PanelWorkflow**: il componente è mostrato in Figura 6.5 e in questo caso l'immagine del macchinario è fissa mentre i testi e le immagini posti in basso dispongono di un'animazione allo scroll.

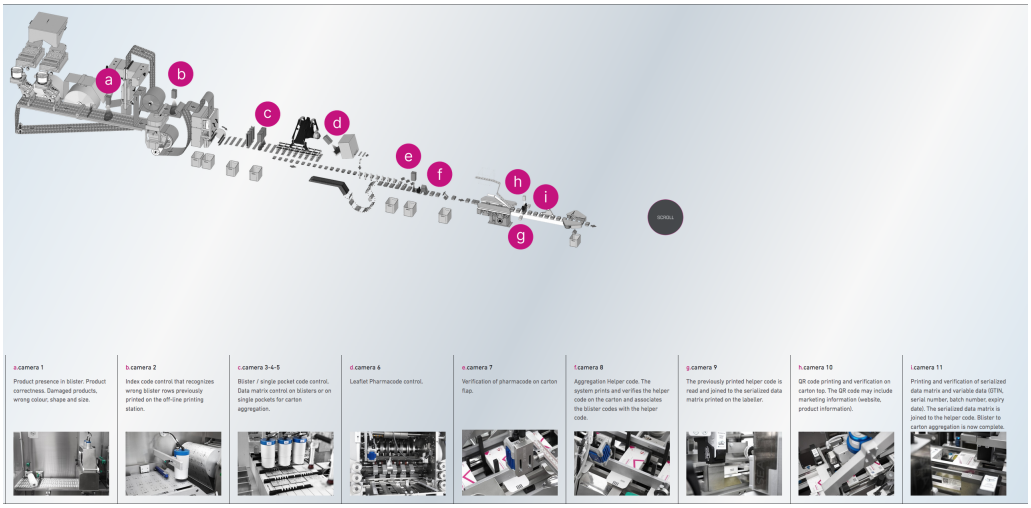

Figura 6.5: Componente PanelWorkflow

6. **PanelAggregation e PanelSection04**: sono composti da immagini e testi che si animano allo scorrimento.

7. **PanelVerify**: come mostrato in Figura 6.6, il componente è composto:

   - dall'immagine del package fissa;
   - dal testo circolare uguale a quello del componente PanelIntro, dove all'interno è presente il link "Try it now" che conduce alla pagina Verify;
   - in basso a destra il link identico a quello descritto sopra;
   - e le frecce che si comportano allo stesso modo di quelle presenti in PanelIntro.

Figura 6.6: Componente PanelVerify

6.3 Header e Menu

L'header è un componente visibile in tutte le pagine del sito presente in *app.vue*, che è il componente principale in cui sono nidificati tutti gli altri componenti, e si comporta nel seguente modo: nella homepage ha il colore del background, come si può vedere nella Figura 6.7, mentre nelle altre pagine assume un altro colore (Figura 6.8).

Figura 6.7: Header homepage

Figura 6.8: Esempio header pagina tech solution

La caratteristica dell'header nella homepage è che quando l'utente clicca sul simbolo del menu, oltre alla comparsa di quest'ultimo, si trasforma come l'header in Figura 6.8, quindi i loghi delle aziende da neri diventano bianchi, il background diventa nero e i bordi delle linee bianche. Inoltre l'hamburger menu ruota di 90° (Figura 6.9).

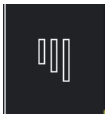

Figura 6.9: Hamburger menu

Questa peculiarità è stata realizzata con *v-bind*, una proprietà di Vue.js che permette l'associazione dei dati e la manipolazione dell'elenco di classi e degli stili inline di un elemento. E poiché le classi e gli stili sono entrambi attributi, si può utilizzare v-bind per assegnare loro un valore stringa in modo dinamico. [31]

In questo caso v-bind è stata utilizzata per controllare se l'header era bianco o nero e come comportarsi di conseguenza.

Nelle altre pagine invece, dove l'header era già di colore nero, l'unica animazione presente era la rotazione dell'hamburger menu e l'effettiva comparsa del menu.

Per quanto riguarda il menu, invece, è stato realizzato con le *Props* di Vue.js. Le props rappresentano le proprietà esterne che permettono di configurare un componente. Devono essere definite a priori tramite la funzione *definePropps()*. Inoltre, se necessario, è possibile definire il loro tipo (String, Boolean, Number...), l'eventuale obbligatorietà (*required*) o un valore di default. [32]

Il numero è uguale a quello descritto nella sottosezione 6.2, ed ogni pannello è un link alla pagina corrispondente. Vedere la Figura 6.10 per maggiore chiarezza.

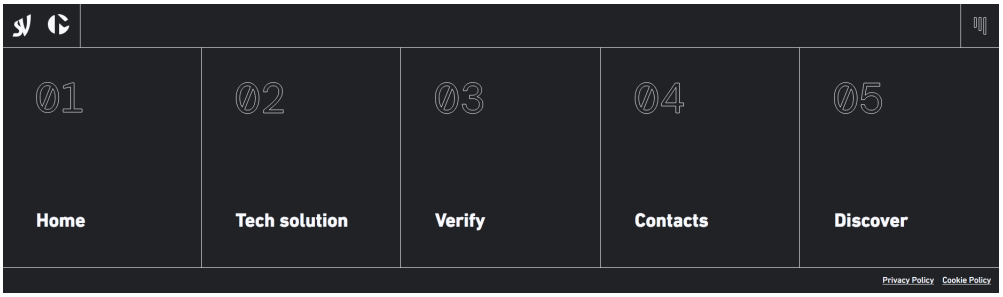

Figura 6.10: Menu

6.4 Tech Solution

La pagina **tech solution** ha lo scopo di descrivere la soluzione tecnica, cioè cosa è e come funziona il prodotto. Quattro sono i componenti realizzati per questa pagina:

1. **PageTitle**: come si può notare nella Figura 6.11, PageTitle è composto dal titolo (realizzato con le Props) e dalle frecce svg descritte nella Sezione 6.2.

Figura 6.11: Titolo della pagina Tech Solution

2. **TechTextImg, TechTextImgDxSx, TechOnlyText**: sono componenti formati da testi e immagini oppure solo testi. In particolare è interessante il componente TechTextImgDxSx in quanto, a secondo del valore passato, sposta l'immagine a destra/sinistra nel layout, come si può vedere in Figura 6.12. Questo grazie ad una prop chiamata *imglast* di tipo booleano che, con valore false sposta l'immagine a sinistra e il testo a destra, con valore true l'esatto opposto.

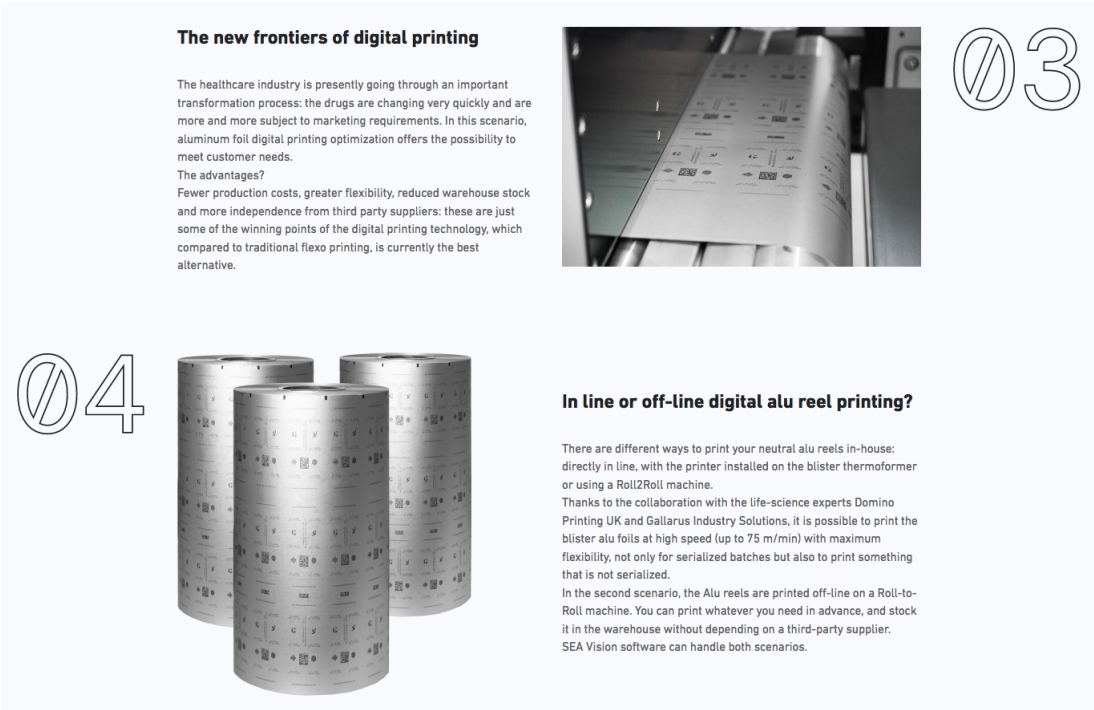

Figura 6.12: Componente TechTextImgDxSx

6.5 Verify

La pagina **verify** serve per far atterrare l'utente che scansiona il codice QR stampato sull'astuccio oppure, in assenza della scansione, per poter digitare a mano i due codici richiesti nella form.

Questa pagina è formata da quattro componenti:

1. **InfoPoint**: componente formato da un bottone *Info*, che apre un Modale (spiegato nella sezione 7.4). Il modale che compare nello schermo è visibile in Figura 6.13.

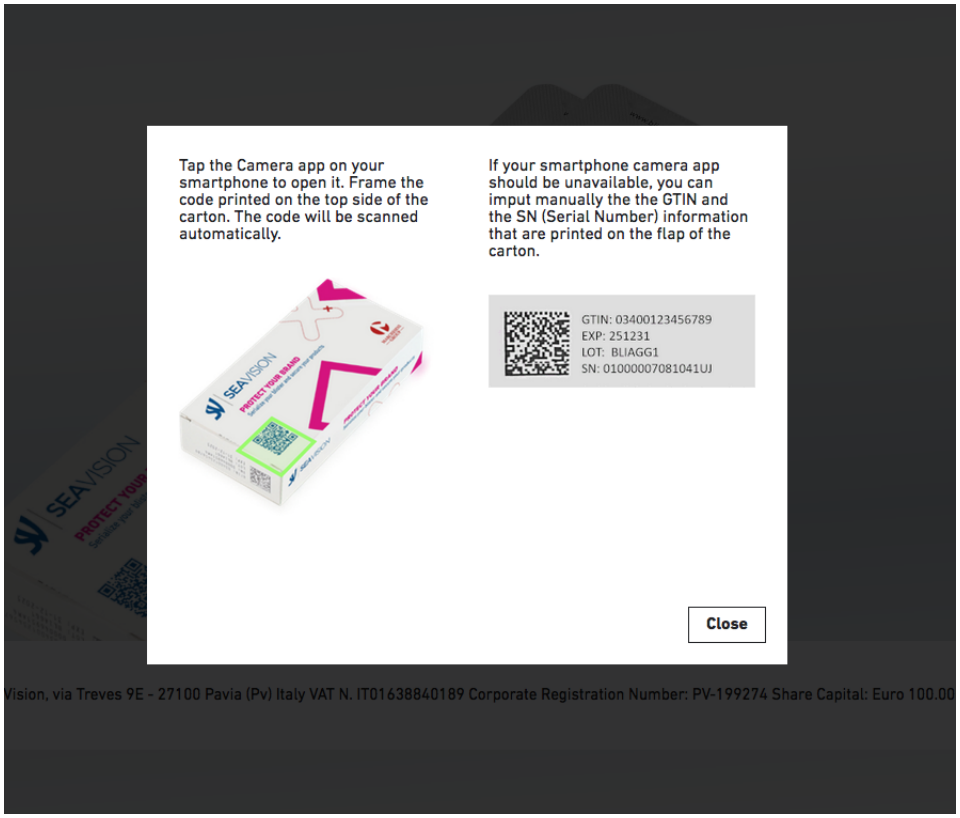

Figura 6.13: Modale

2. **SerialForm**: componente composto da una form con due campi: Gtin e Serial, realizzata con **FormKit**, che è un framework per la creazione di form Vue che semplifica la struttura dei moduli, la generazione, la convalida, i temi, l'invio, la gestione degli errori e tanto altro. [33]

3. **Serials**: restituisce i risultati ottenuti dai codici inseriti, ovvero i dati del package e di ogni blister.

4. **BlisterAnimation**: componente che gestisce l'animazione dei blister (Figura 6.14) che escono dal package, interamente realizzata con proprietà gsap. In particolare si è fatto uso di una timeline che gestisse i tempi delle animazioni e delle tween gsap.to() e gsap.fromTo() descritte nel capitolo 4.

Figura 6.14: Animazione dei blister

6.6 Contacts

Nella pagina **Contacts**, l'unico componente presente è PageTitle descritto nella Sezione 6.4. Il resto della pagina è formato dai due loghi delle aziende richiedenti il sito con una descrizione. Cliccando sui loghi l'utente viene reindirizzato verso le homepage delle aziende.

Infine, è presente una form il cui cudice è stato fornito direttamente dal cliente dell'azienda ospitante.

6.7 Discover

Anche la pagina **Discover**, come Contacts, dispone di un solo componente, PageTitle.

Il resto è formato da:

- due testi;
- un calendario in cui è possibile prenotarsi alla fiera, realizzato con Calendly, che è un servizio che semplifica la programmazione degli appuntamenti;
- e da un video esplicativo del prodotto realizzato dal cliente.

Capitolo 7

Implementazione

Il presente capitolo ha l'obiettivo di commentare le principali funzioni implementate con il relativo codice applicativo.

In particolare è interessante:

- come viene gestita l'animazione delle frecce (componente Vector);
- il componente Header e Menu;
- il componente BlisterAnimation della pagina verify;
- la gestione dei campi gtin e serial;
- il Modale utilizzato per l'info point nella pagina verify;
- il componente Numbers.

7.1 Vector

L'animazione delle frecce viene gestita attraverso l'uso delle *Props*. Come si può vedere in Figura 7.1, sono state definite cinque Props:

1. **arrows**: di tipo Number, è il numero delle frecce che viene passato al componente;
2. **horizontal**: di tipo Boolean, che ha valore True solo quando la dimensione dello schermo è uguale o superiore alle medie dimensioni (in questo caso le frecce si muovono in orizzontale), False altrimenti (movimento delle frecce verticale);
3. **dark**: anch'esso di tipo booleano, che ha valore True quando si vuole che il contorno delle frecce sia nero, False invece che sia bianco;
4. **big**: di tipo booleano, con valore True le frecce sono di dimensione più grande;
5. **speed**: di tipo Number, permette di controllare la velocità con cui si muovono le frecce.

const **numberscount** è un array di numeri, il doppio di quelli che gli vengono passati, in questo modo è possibile animare le frecce.

const **varspeed** è una variabile css che va a finire in uno style per settare la velocità di animazione, dato che si tratta di un'animazione css.

const **styletranslate** gestisce la posizione delle frecce, per esempio: se il numero delle frecce passate è 3, in realtà come detto prima, ne vengono create 6, e le 3 in più sono in position *absolute*, in questo modo quelle originali dettano la larghezza del container. Quelle in position absolute è necessario riposizionarle con un trasform css, basato sul fatto che sia una lista di frecce orizzontali o verticali; sostanzialmente il trasform le sposta in percentuali progressive dettate dal loro indice i.

Con la proprietà *linear infinite* definita nel css, il movimento delle frecce continua all'infinito.

```
1  <script setup>
2  const props = defineProps({
3      arrows: {
4          type: Number
5      },
6      horizontal: {
7          type: Boolean,
8          default: true
9      },
10     dark: {
11         type: Boolean,
12         default: false
13     },
14     big: {
15         type: Boolean,
16         default: true
17     },
18     speed: {
19         type: Number,
20         default: 5
21     }
22 })
23
24 const numberscount = computed(() => [...Array(props.arrows * 2).keys()])
25 const varspeed = computed(() => `--speed: ${props.speed}s`)
26 const styletranslate = (i) => {
27     if (i > props.arrows) {
28         const t = props.horizontal ? 'translateX' : 'translateY'
29         const p = props.horizontal ? 'left: 0' : 'top: 0'
30         return `${p}; transform: ${t}(calc(100% * ${i - 1}))`
31     }
32 }
33
34 </script>
```

Figura 7.1: Script componente Vector

7.2 Header e Menu

Per realizzare l'header si è fatto uso del componente *Transition* di Vue.js, che è un componente built-in, ciò significa che è disponibile nel template di qualsiasi componente senza doverlo registrare. Può essere utilizzato per applicare animazioni di entrata e uscita su elementi o componenti passati tramite il suo slot predefinito.

L'elemento *slot* è uno slot di uscita che indica dove deve essere visualizzato il contenuto dello slot fornito dal genitore. L'entrata o l'uscita possono essere attivati in uno dei seguenti modi:

- Rendering condizionale tramite v-if.
- Visualizzazione condizionale tramite v-show.
- Commutazione di componenti dinamici tramite l'elemento speciale *component*. [34]

Nel caso dell'header, Transition è stato utilizzato per cambiare il colore dei loghi delle aziende.

È presente inoltre il componente *Hamburger*, a cui è associato un evento @click e a cui viene passata una props booleana *open* per verificare se il menu è stato aperto. (Figura 7.2)

```
1  <template>
2  <div>
3      <div ref="header">
4          <div class="relative z-10">
5              <div class="relative flex z-10 border-b min-h-[56px] md:min-h-[80px]" :class='{"border-gray text-gray bg-black-900" : iswhite, "transition-colors duration-300 delay-[0.6s] border-
   black-900": !iswhite}'>
6
7                  <div class="relative border-r border-solid p-5 flex-shrink-0" :class='iswhite ? "border-gray" : "transition-colors delay-[0.6s] duration-300 border-black-900"'>
8                      <NuxtLink to="/" class="absolute top-0 left-0 w-full h-full z-10" />
9                      <div class="flex gap-3 md:gap-8 " :class="{'to-black': !iswhite }">
10                         <div class="relative">
11                             <Transition name="fade">
12                                 <img class="w-4 md:w-auto" :key="'white'" v-if="iswhite" src="~/assets/img/SV_logo_white.png">
13                                 <img class="w-4 md:w-auto" :key="'black'" v-else src="~/assets/img/SV_logo_black.png">
14                             </Transition>
15                         </div>
16                         <div class="relative">
17                             <Transition name="fade">
18                             <img class="w-5 md:w-auto" :key="'white'" v-if="iswhite" src="~/assets/img/Marchesini_logo_white.png">
19                             <img class="w-5 md:w-auto" :key="'black'" v-else src="~/assets/img/Marchesini_logo_black.png">
20                             </Transition>
21                         </div>
22                     </div>
23                 </div>
24
25                 <div class="flex border-r border-solid w-full p-5 md:text-24 pointer-events-none select-none" >
26                     <ClientOnly>
27                         <div class="flex flex-col justify-center">
28                             <div>{{ title }}</div>
29                         </div>
30                     </ClientOnly>
31                 </div>
32
33                 <div class="grid place-content-center flex-grow-0 p-5" :class='iswhite ? "border-gray" : "transition-colors delay-[0.6s] duration-300 border-black-900"'>
34                     <Hamburger :open="open" @click="togglemenu()" :iswhite="iswhite" />
35                     </div>
36             </div>
37
38             <div class="absolute left-0 w-full top-full z-0">
39                 <div :class='{"!max-h-0 pointer-events-none" : !open}' class="bg-black-900 relative overflow-hidden transition-all duration-[0.8s]">
40                     <Menu />
41                     </div>
42             </div>
43         </div>
44     </div>
45 </div>
46 </template>
```

Figura 7.2: Template del componente Header

Per quanto riguarda il Menu, invece, anche ad esso è associato un evento. Se l'hamburger menu non è stato cliccato, allora il menu è nascosto, altrimenti viene mostrato.

Ogni pannello del menu è un link che reindirizza alla pagina corrispondente. I link sono stati implementati tramite *NuxtLink*, che è un componente di Nuxt.js che serve per poter navigare nelle pagine dell'applicazione.

Come si può notare dalla Figura 6.10, i link sono cinque e, per evitare ridondanza di codice, si è deciso di utilizzare un array di oggeti di nome *links* visualizzabile in Figura 7.3. Questo permette, nel template, di utilizzare un ciclo for per tutti i link e mostrarli uno dopo l'altro con il relativo numero. (Figura 7.4)

```
1  const links = [
2      {
3          name: 'Home',
4          link: '/',
5          number: 1
6      },
7      {
8          name: 'Tech solution',
9          link: '/tech_solution',
10         number: 2
11     },
12     {
13         name: 'Verify',
14         link: '/verify',
15         number: 3
16     },
17     {
18         name: 'Contacts',
19         link: '/contacts',
20         number: 4
21     },
22     {
23         name: 'Discover',
24         link: '/achema',
25         number: 5
26     }
27 ]
```

Figura 7.3: Array di oggetti del componente Menu

```
1  <template>
2      <div class="grid grid-cols-2 menu-max-h overflow-y-scroll md:overflow-auto lg:grid-cols-5 z-0">
3          <template v-for="link in links" :key="link.link">
4              <NuxtLink class="maxlg:border-b border-gray" :to="link.link">
5                  <div class="flex flex-col justify-between items-start p-6 md:p-8 xl:p-16 gap-4 md:gap-5 border-r border-solid border-gray h-full">
6                      <Numbers :number="link.number" />
7                      <div class="text-24 md:text-36 font-bold text-gray pt-10 md:pt-20 md:pt-40">
8                              {{ link.name }}
9                      </div>
10                 </div>
11             </NuxtLink>
12         </template>
13         <div class="py-3 border-t border-b border-gray flex flex-col justify-center md:flex-row md:justify-end col-span-full gap-4 px-4">
14             <div class="text-gray">
15             <NuxtLink class="underline" to="/privacy_policy">Privacy Policy</NuxtLink>
16             </div>
17             <div class="text-gray">
18                 <NuxtLink class="underline" to="/cookie_policy">Cookie Policy</NuxtLink>
19             </div>
20         </div>
21     </div>
22 </template>
```

Figura 7.4: Template del componente Menu

7.3 BlisterAnimation

BlisterAnimation è il componente che gestisce l'animazione dei blister nella pagina verify. Questa animazione viene eseguita solo in due casi:

- quando l'utente, dalla homepage, scansione il QR code;
- quando l'utente digita manualmente i codici gtin e serial ed essi sono corretti.

Per questa animazione sono stati creati due componenti ref(), *blister* e *pack* (ref() prende un valore interno e restituisce un oggetto ref reattivo e mutabile, che ha

una singola proprietà .value che punta al valore interno) che sono stati definiti nel template (Figura 7.5).

In questo modo è possibile animarli con proprietà gsap:

- prima di tutto è stata definita una timeline *tl*, che si ricorda che è un potente strumento di sequenziamento che funge da contenitore per le Tween e altre Timeline, semplificandone il controllo nel suo insieme e la gestione precisa dei tempi, con *paused: true*. paused ottiene o imposta lo stato di pausa dell'animazione, ed indica se quest'ultima è attualmente in pausa o meno.
- a questo punto si è fatto uso delle Tween *to()* e *fromTo()*: con la Tween to() il package è stato spostato dal centro dello schermo all'angolo in basso a sinistra, mentre con la Tween fromTo() è stata decisa la posizione di partenza del blister (esattamente quella finale del package) e la sua posizione finale (in alto a destra), per dare l'effetto che i blister uscissero dalla scatola. I blister, infatti, anche se nella stessa posizione iniziale del package, non sono comunque visibili, in quanto il package ha uno z-index più elevato. La proprietà z-index nei CSS specifica la rilevanza di visualizzazione di un elemento rispetto ad un altro.

Arrivati a questo punto, in base al numero di blister (che può essere maggiore o uguale a 2), vengono effettuate delle trasform su tutti i blister escluso il primo. Come un mazzo di carte, i blister si spostano leggermente verso l'angolo in alto a destra, come si può vedere dalla Figura 6.14.

```
1  <template>
2      <div class="max-h-full relative grid place-content-center">
3          <div class="absolute w-full h-full grid place-content-center" :numberserials="numberserials" v-for="number in numbers_of_blister" :key="number">
4              <img class="blister block m-auto" ref="blister" src="~/assets/img/Blister_Verify.png">
5          </div>
6          <div class="z-10">
7              <img ref="pack" src="~/assets/img/Package_5.png">
8          </div>
9      </div>
10 </template>
```

Figura 7.5: Template del componente BlisterAnimation

7.4 Gtin e Serial

Si possono distinguere due casi nella gestione dei campi gtin e serial:

1. il QR code viene scansionato dall'utente nella homepage;
2. i campi vengono inseriti manualmente dall'utente.

Nel primo caso l'utente, una volta scansionato il QR code, viene reindirizzato alla pagina verify, e in particolare alla funzione *getUrlParams*. getUrlParams salva nella variabile *route* il percorso corrente grazie al composable useRoute() di Nuxt

3. Dopodiché crea un array di due campi (uno gtin e l'alto serial), in cui salva i codici scansionati secondo una query string definita come in Figura 7.6.

Il campo gtin è quel valore dopo lo 01 di minimo 10 cifre, mentre il campo serial è il valore dopo le cifre 21.

```
/^01([0-9a-zA-Z]{10,})21([0-9a-zA-Z]{10,})/
```

Figura 7.6: Query string

La funzione si conclude restituendo gtin e serial. Successivamente i campi vengono controllati dall'API dell'azienda, cioè che i valori inseriti abbiano il numero di cifre corretto e che non siano campi vuoti. Se vengono superati questi controlli allora andranno spediti all'API dell'azienda richiedente il sito.

Nel secondo caso, invece, quando l'utente inserisce i campi manualmente, viene richiamata la funzione *submit* che effettua gli stessi controlli elencati precedentemente e resetta i campi.

7.5 Gestione dell'info point

L'info point nella pagina verify è stato gestito grazie all'uso dei Modali. I Modali in Vue.js sono una convenzione sull'esperienza utente per indirizzare l'attenzione su un contenuto che deve leggere o con cui interagire. Questi tendono a prendere la forma di piccoli blocchi di contenuto direttamente nel campo visivo dell'utente con una sorta di sfondo che oscura o nasconde il resto del contenuto della pagina. [35]

Come viene mostrato in Figura 7.7, il "click" del bottone *Info* richiama una funzione di nome *showModal* che imposta il valore di *isShow* a True e il Modale viene mostrato.

Quando poi il bottone *Close* viene premuto, la funzione *closeModal* imposta *isShow* a False e il Modale viene chiuso.

7.6 Numbers

I numeri presenti nel progetto sono un'immagine svg con un'animazione gsap chiamata **DrawSVGPlugin**, che consente di rivelare (o nascondere) progressivamente il tratto di un svg, ed è possibile animare verso l'esterno dal centro del tratto. [36]

Inoltre nel componente Numbers sono state definite delle Props:

- **number**: il numero che deve essere mostrato;

```
<template>
  <p>
    <button @click="showModal" class="bg-transparent text-purple font-semibold py-2 px-4 text-[11px] rounded-full border hover:border-transparent">Info</button>
  </p>
  <div class="w-10 h-10">
    <Modal v-model="isShow" :close="closeModal" class="z-100">
      <div class="bg-white text-xl">
        <div>
          <slot name="body" />

        </div>
        <div class="m-5 flex justify-end">
            <button @click="closeModal" class="bg-transparent font-semibold py-2 px-4 border hover:border-transparent mt-8">
                <slot name="button" />
            </button>
        </div>
      </div>
    </Modal>
  </div>
</template>
```

Figura 7.7: Modale

- **sizeBig**: booleano, se di valore True viene mostrato di dimensione più grande, False altrimenti;
- **color**: stringa, il numero può avere contorno nero o bianco;
- **duration**: numero, di default 2 secondi, cioè il tempo speso per disegnare completamente il numero;
- **restart**: booleano, controlla se far ripartire l'animazione dopo che è uscita dalla viewport.

Dopodiché grazie ad una Timeline e alla Tween fromTo(), il tratto del numero viene disegnato (vedere Figura 7.8).

```
const tl = gsap.timeline({paused: true})
tl.fromTo(q('svg path'), {drawSVG:"0%"}, {duration: props.duration, drawSVG:"100%", stagger: 0.1})
```

Figura 7.8: Timeline del componente Numbers

Conclusioni

Sulla base del lavoro svolto nella presente tesi, di seguito una sintesi riguardo al percorso seguito e i risultati raggiunti.

Una parte importante del lavoro ha riguardato lo studio dei framework descritti in precedenza, in particolare i concetti base e le caratteristiche di ognuno di essi, al fine di comprenderne a pieno il funzionamento. Infatti, prima della realizzazione del sito vero e proprio, sono stati eseguiti molti test per arrivare ad una comprensione più approfondita del problema presentato.

Per quanto riguarda Vue.js, si può concludere dicendo che è uno dei framework Javascript più famosi al mondo, competendo con aziende del calibro di Google e Facebook, i rispettivi creatori di Angular e React. Vue.js, inoltre, ha preso le caratteristiche migliori di Angular e React e le ha fuse, ma a differenza di Angular che è più ostico da imparare, Vue è più semplice ed è spesso consigliato a chi inizia la carriera nel mondo front-end.

Nuxt.js, invece, è risultato un framework molto potente con molte caratteristiche utili, che ha permesso lo sviluppo dell'applicazione estremamente semplice. In particolare il rendering lato server, lo sviluppo più rapido grazie al router autogenerico, la divisione automatica del codice con pagine pre-renderizzate: tutto questo è impossibile o estremamente complesso da ottenere con Vue.js.

Per quanto riguarda GSAP si può concludere che ha semplificato la creazione e la manipolazione delle animazioni della Timeline, anche se si ha poca o nessuna comprensione di JavaScript.

Mentre grazie a Windi CSS, le pagine e i componenti hanno poche righe di codice di css, e questo ha permesso che i tempi di compilazione si riducessero notevolmente.

Uno sviluppo futuro di questo sito potrebbe essere quello di estenderlo e creare un'altra pagina in cui vengono riportati immagini e video della fiera, quanto e come il nuovo software ha riscosso successo e, nel caso, i miglioramenti futuri su cui si dovrà lavorare.

In conclusione, lo studio di questi framework e la loro integrazione nel sito, ha permesso la realizzazione di un sito web moderno già in funzione ed utilizzabile, capace di catturare l'attenzione dell'utente.

Bibliografia

[1] URL: https://www.redhat.com/it/topics/cloud-computing/what-is-saas.
[2] URL: https://vuejs.org/.
[3] URL: https://techdayhq.com/community/articles/the-top-reasons-why-vue-js-is-so-popular.
[4] URL: https://vuejs.org/guide/extras/reactivity-in-depth.html.
[5] URL: https://medium.com/@rubeena.ajeed/introduction-to-vue-js-a16614f20f77.
[6] URL: https://vuejs.org/guide/introduction.html#the-progressive-framework.
[7] URL: https://vuejs.org/guide/scaling-up/sfc.html#why-sfc.
[8] URL: https://www.webmound.com/composition-api-vs-options-api-in-vue-3/.
[9] URL: https://vuex.vuejs.org/guide/.
[10] URL: https://pinia.vuejs.org/.
[11] URL: https://v3.nuxtjs.org/.
[12] URL: https://v3.nuxtjs.org/guide/directory-structure/composables/.
[13] URL: https://vite.nuxtjs.org/.
[14] URL: https://vuejs.org/guide/scaling-up/ssr.html#client-hydration.
[15] URL: https://www.kuma.cloud/seo-friendly/.
[16] URL: https://v3.nuxtjs.org/guide/concepts/rendering/.
[17] URL: https://giuseppesperanza.medium.com/6-motivi-per-cui-amerai-nuxt-se-gi%C3%A0-utilizzi-vue-js-5f749abe182d.
[18] URL: https://v3.nuxtjs.org/guide/concepts/auto-imports.
[19] URL: https://github.com/unjs/h3.
[20] URL: https://v3.nuxtjs.org/guide/concepts/server-engine.
[21] URL: https://v3.nuxtjs.org/guide/features/head-management#meta-components.
[22] URL: https://v3.nuxtjs.org/guide/features/data-fetching.
[23] URL: https://greensock.com/gsap/.
[24] URL: https://glue-labs.com/articoli/gsap-animation-framework-le-web-application.
[25] URL: https://www.smashingmagazine.com/2020/09/animating-react-components-greensock/.
[26] URL: https://greensock.com/docs/v3/GSAP.
[27] URL: https://windicss.org/.
[28] URL: https://windicss.org/features/responsive-design.html.

[29] URL: https://windicss.org/features/.
[30] URL: https://tailwindcss.com/docs/utility-first.
[31] URL: https://vuejs.org/guide/essentials/class-and-style.html.
[32] URL: https://www.html.it/pag/67708/components-data-template-e-props/.
[33] URL: https://formkit.com/.
[34] URL: https://vuejs.org/guide/built-ins/transition.html#the-transition-component.
[35] URL: https://www.digitalocean.com/community/tutorials/vuejs-vue-modal-component.
[36] URL: https://greensock.com/drawsvg/.