Corso di Laurea Magistrale in
Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Progettazione ed implementazione del front-end web di un catalogo digitale interattivo

Relatori
Prof. Giovanni Malnati
Prof. Daniele Apiletti

Candidati
Daniele Bruscella
Arianna Zuardi

Anno Accademico 2020-2021

Sommario

Al giorno d'oggi, in una società il cui fulcro comunicativo è Internet, per un'azienda essere presente online con il proprio marchio e i propri prodotti non è più una scelta di marketing ma un "obbligo" da ottemperare per essere competitivi sul mercato. Nell'ambito della produzione di applicativi software, emerge sempre più l'esigenza di migrare i prodotti sul web in modo da renderli facilmente accessibili in un contesto multipiattaforma semplificando il processo di distribuzione, installazione e manutenzione.

In questo scenario, partendo da un esistente applicativo desktop, è stata progettata ed implementata l'interfaccia utente di un'applicazione web finalizzata alla consultazione e alla gestione di un catalogo contenente la raccolta di dati tecnici relativi a componenti impiantistici e, più in generale, al settore dell'edilizia. L'applicazione mira a garantire una semplice ma efficace esperienza di navigazione: l'utente ha la possibilità di esplorare il catalogo per macro-aree, categorie e gruppi di prodotti, applicando filtri sui risultati, oppure può semplicemente utilizzare il motore di ricerca integrato. Inoltre, per ogni prodotto è possibile visualizzare la scheda delle specifiche tecniche, metterlo a confronto con altri della stessa categoria, tenerne traccia nella cronologia di navigazione e inserirlo nella propria raccolta di preferiti.

Sul piano tecnologico, le scelte implementative sono in linea con i trend attuali: è stata utilizzata la libreria *React* per la gestione dello stato e per la realizzazione dell'applicazione web sotto forma di *Single Page Application*; per simulare la navigazione multi-pagina e, più in generale, per la progettazione dell'architettura di routing è stata utilizzata la libreria *React Router*. L'interazione con il server è stata gestita tramite le funzionalità della libreria *Axios* e al fine di rendere l'interfaccia utente responsive, usabile e coerente con le linee guida del Material Design di Google ci si è avvalsi dei componenti grafici di *Material-UI*.

Durante lo sviluppo si è prestata particolare attenzione alla scalabilità in modo da minimizzare lo sforzo per l'ampliamento dei contenuti del catalogo e si è cercato di massimizzare l'usabilità affinché l'esperienza di navigazione sia il più possibile conforme alle aspettative dell'utente.

Abstract

Nowadays, in a society where the communication is mainly centred through the Internet, for a company being online with its own brand and products is no longer a marketing choice but an "obligation" to be met in order to be competitive on trade. In the context of the production of software applications, there is an increasing need to migrate products to the web to make them easily accessible in a multi-platform context simplifying the distribution, installation and maintenance process.

In this scenario, starting from an existing desktop application, has been designed and implemented the user interface of a web application aimed at consulting and managing a catalogue containing the collection of technical data relating to plant engineering and, more generally, to the construction sector. The application aims to ensure a simple but effective browsing experience: the user has the possibility to explore the catalogue through macroareas, categories and groups of products, applying filters on the results, or simply to use the integrated search engine. Furthermore, each product can be viewed through the technical specifications sheet, compared with others, kept in the browsing history and inserted in a collection of favourites.

From a technological point of view, the implementation choices are in line with current trends: the *React* library was used for state management and for the creation of the web application as a *Single Page Application*; the *React Router* library was used to simulate multi-page navigation and, more generally, to design the routing architecture. The interaction with the server has been managed through the functions of the *Axios* library and in order to make the user interface responsive, usable and consistent with the guidelines of Google's Material Design, the graphic components of *Material-UI* have been used.

During the development phase, particular attention was paid to scalability in order to minimize the effort for enlarging the contents of the catalogue and usability was maximized in order to make the browsing experience compliant with the user's expectations.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

In un mercato la cui produzione industriale è sempre più diversificata e variegata è di fondamentale importanza l'attività di raccolta dati e la realizzazione di un archivio generalizzato per rendere disponibile in maniera centralizzata l'offerta di prodotti e l'esposizione delle caratteristiche tecniche.

Nel contesto dell'edilizia, la raccolta dei dati tecnici al fine del loro inserimento in un unico grande catalogo digitale è un'iniziativa che trova larga approvazione sia da parte dei professionisti del settore sia da parte dei produttori: dal punto di vista del professionista, un catalogo unico generale può agevolare la ricerca dei prodotti da utilizzare nelle opere progettuali e di conseguenza aumentare anche la facilità di reperimento; dal punto di vista del produttore, invece, può accrescere la visibilità del marchio con conseguente aumento indiretto delle vendite dovuto all'alto tasso di consultazione dei prodotti. Inoltre, la coesistenza all'interno di uno stesso archivio di prodotti dalle simili finalità ma con marchi differenti può stimolare una crescita positiva dell'offerta, con il possibile aumento qualitativo delle specifiche ad un costo onesto sul mercato.

In questo scenario l'iniziativa di realizzare un archivio unico centralizzato è stata concretizzata all'inizio del nuovo millennio nel contesto aziendale di *Edilclima S.r.l.* tramite lo sviluppo di un software applicativo installabile sul PC che rende navigabili e fruibili all'utente i dati tecnici relativi a componenti impiantistici, pannelli solari, materiali per l'isolamento termico-acustico e all'edilizia in generale. La raccolta dei dati è avvenuta tramite la collaborazione con un complesso di produttori che hanno aderito al progetto mettendo a disposizione i rispettivi cataloghi cartacei o digitali dei prodotti, affinché

le informazioni venissero strutturate e inserite in un database.

Tutti gli applicativi software, però, sono accomunati dall'impellente necessità di innovazione. Nello specifico, negli ultimi anni il termine *cloud* inizia ad essere una costante di qualunque prodotto software e stiamo assistendo ad una migrazione massiva di moltissimi applicativi verso il web. Questo accade principalmente per due ragioni: la prima, di carattere globale, perché Internet è universalmente riconosciuto come il fulcro della comunicazione odierna e quindi per le aziende è di fondamentale importanza avere una rete di distribuzione dei prodotti e di interazione con i clienti che si dirama sul web; la seconda, di carattere più tecnico-pratico, è dovuta alla necessità di semplificazione dei processi di distribuzione, installazione e manutenzione di un software e di renderne immediato l'utilizzo in un contesto multipiattaforma.

Alla luce di questa realtà è emersa la necessità di trasformare l'esistente applicativo desktop in un'applicazione web per rendere il catalogo:

- **accessibile, disponibile e gestibile** in ogni momento ed online tramite Internet;
- **consultabile** tramite PC, smartphone e tablet;
- **navigabile** attraverso una nuova interfaccia user-friendly.

Questo lavoro di tesi è incentrato sulla realizzazione dell'applicativo web partendo dall'analisi della soluzione desktop *Archivio Produttori* e della relativa struttura dei dati, proseguendo con la realizzazione delle interfacce grafiche delle pagine per consentire la navigazione e la modifica del catalogo ed infine implementando funzionalità aggiuntive quali il filtraggio, la ricerca e il confronto tra prodotti al fine di migliorare l'esperienza di navigazione dell'utente finale.

L'interfaccia utente è stata sviluppata utilizzando la libreria *React* per la realizzazione di componenti dinamici e per la gestione dello stato, la libreria *React Router* per la definizione del routing e la libreria *Material-UI* per l'introduzione di componenti grafici responsivi che adottano come linea guida il Material Design di Google. L'implementazione dell'applicazione è avvenuta in modo incrementale con particolare attenzione alla scalabilità: partendo in fase iniziale da un sottoinsieme di prodotti dell'archivio, il catalogo è stato ampliato gradualmente a più categorie di prodotti estendendone progressivamente il numero.

Gli sviluppi futuri saranno incentrati principalmente sul proseguimento dell'ampliamento del catalogo per consentirne la navigazione completa e sull'estensione delle già presenti, seppur in forma primordiale, funzionalità di personalizzazione che consentono ad ogni utente di aggiungere nuovi prodotti e di modificare quelli già esistenti.

I contenuti di questo elaborato saranno organizzati nel seguente modo:

- nei primi capitoli saranno illustrate le fondamenta delle applicazioni web, dall'architettura alle tecnologie di sviluppo, analizzando principalmente la libreria *React*, fino all'esperienza utente nella navigazione;
- i seguenti saranno incentrati sulle fasi di sviluppo della soluzione: dall'analisi dell'applicativo preesistente all'architettura e progettazione della soluzione web fino a trattare tutte le funzionalità implementate, con particolare attenzione su quanto di nuovo è stato introdotto;
- infine, l'ultimo capitolo è dedicato alle conclusioni e agli sviluppi futuri programmati.

1.1 Descrizione del contesto

Edilclima S.r.l. è una software-house dedita allo sviluppo di programmi di calcolo per la progettazione impiantistica e per la verifica dell'osservanza dei vincoli di legge. Tali strumenti di calcolo hanno l'obiettivo di supportare nella fase decisionale i progettisti, i certificatori e più in generale tutti i professionisti del settore energetico, impiantistico e antincendio, tenendo in considerazione le esigenze formali e burocratiche anche dal punto di vista ambientale.

Figura 1.1: Logo *Edilclima S.r.l.*

Tra i vari prodotti offerti dall'azienda vi è *Archivio Produttori*: un applicativo nato con lo scopo di gestire (leggere, modificare, aggiornare) gli archivi dei componenti e dei materiali utilizzati dai programmi di calcolo di Edilclima. Ogni singolo progetto creato all'interno di un software di calcolo di Edilclima

può includere diversi prodotti di impiantistica. Tali prodotti vengono selezionati e importati dall'archivio il quale contiene la raccolta dei componenti di tutti i produttori partner che hanno aderito all'iniziativa.

Emerge subito quanto sia importante la scalabilità dell'archivio, affinché possa prestarsi con il minimo sforzo all'integrazione di nuove categorie di prodotti provenienti da nuovi produttori aderenti all'iniziativa.

Inoltre, durante la creazione di un progetto, l'archivio supporta l'utente nella fase di inserimento di prodotti già esistenti sul mercato in maniera rapida e automatizzata che eventualmente sono stati già utilizzati o modificati in precedenza. In questo scenario di utilizzo si evince anche il secondo aspetto su cui ci si è particolarmente concentrati durante lo sviluppo della soluzione: l'usabilità. Infatti, si è cercato di agevolare il più possibile la navigazione dell'archivio in modo da rendere l'esperienza utente il più possibile piacevole e conforme alle aspettative.

1.2 Obiettivi

L'obiettivo principale, in base a quanto esposto, è la realizzazione di un'applicazione web sufficientemente scalabile e usabile per la consultazione e la navigazione dell'archivio.

Tra gli obiettivi paralleli rientrano anche i seguenti aspetti:

- apportare innovazione e miglioramenti alle funzionalità già esistenti dell'archivio, sia dal punto di vista grafico, sia dal punto di vista tecnologico;
- prestare particolare attenzione alle esigenze degli utilizzatori che da anni utilizzano il programma, affinché possano accogliere la migrazione alla soluzione web con positività;
- adottare soluzioni in grado di far evolvere l'applicativo da un pattern d'uso desktop ad uno web.

Capitolo 2

Applicazioni web

Un'applicazione web (dall'inglese *web application* o abbreviato *webapp*) è un'applicazione che viene eseguita tipicamente all'interno di un browser web e che, tramite i classici protocolli di rete, può comunicare con un server al fine di offrire un determinato servizio all'utente finale.

A differenza dei comuni software applicativi e delle app native, le applicazioni web si distinguono per l'indipendenza dalla piattaforma e dal sistema operativo su cui vengono eseguite; infatti, poiché vengono eseguite all'interno di un browser non interagiscono direttamente con il sistema operativo sottostante e di conseguenza non necessitano di installazione, né tantomeno di essere manualmente aggiornate dall'utente in quanto la versione in esecuzione è sempre la stessa per tutti i client.

Anche se il contesto di esecuzione è lo stesso, le applicazioni web sono differenti dai comuni siti web; un sito nasce come una raccolta di pagine collegate tra loro con la finalità ultima di esporre informazioni utili sotto forma di testo o contenuti multimediali; un applicazione web, invece, interagisce continuamente con l'utente per elaborare informazioni e produrre risultati. Si ottiene così un applicativo che è personalizzabile ad hoc in base alle proprie esigenze ma che allo stesso tempo sfrutta la versatilità e la semplicità di utilizzo del web.

Tutti questi principali vantaggi legati alle applicazioni web ne hanno agevolato un'ampia diffusione a livello globale; inoltre, la crescita è stata ulteriormente velocizzata grazie al fatto che le tecnologie di sviluppo per il web sono principalmente open-source e gratuite e di conseguenza esse migliorano grazie al continuo contributo della community degli sviluppatori.

Al giorno d'oggi sono tantissime le applicazioni web utilizzate, con diverse finalità, dalle web mail, ai social network, ai web forum, agli e-commerce. Alcuni esempi concreti sono: *Evernote* e *Trello* per la gestione di note e appunti personali, *Spotify* e *Netflix* per la riproduzione contenuti multimediali in streaming e la *Google Suite* per la gestione e condivisione di documenti.

In questo capitolo verranno analizzati gli aspetti fondamentali legati alle applicazioni web a partire dall'architettura su cui si basano fino alle tecnologie utilizzate per la realizzazione.

Figura 2.1: Applicazioni web

2.1 Storia

A partire dagli anni '80 le applicazioni web prevedevano la suddivisione netta del carico di lavoro tra un elaboratore che aveva il ruolo di *client*, ossia di cliente fruitore del programma e un *server*, colui che offriva il servizio e rispondeva alle richieste. Quindi, la macchina di ogni utente (che fungeva da client) necessitava di un programma installato in locale per comunicare con il server. Periodicamente tale applicativo doveva essere aggiornato manualmente. Inoltre, per ogni sistema operativo era necessario realizzare e distribuire un applicativo differente, e per gli sviluppatori non poche erano le difficoltà da affrontare per il porting del software.

Gli anni '90 sono caratterizzati dalla nascita e diffusione del World Wide Web, dall'apparizione dei primi browser, dalla nascita del protocollo HTTP, del linguaggio HTML e del CSS. È diventato così possibile creare pagine statiche che un server metteva a disposizione dei client.

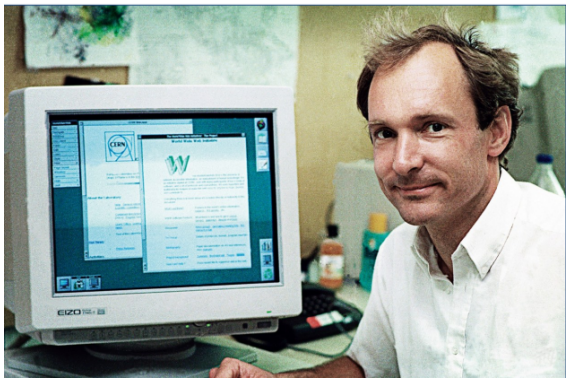

Figura 2.2: Tim Berners-Lee, inventore del World Wide Web e ideatore del primo server HTTP

Agli albori del nuovo millennio si diffonde l'idea del cosiddetto *web dinamico*, alla base della quale vi è l'utilizzo di linguaggi di scripting come JavaScript, ideato da *Netscape* nel 1995, con lo scopo di rendere dinamiche e interattive le pagine web che sino ad allora erano meramente statiche. È l'avvento del *Web 2.0*: gli utenti non sono più visitatori passivi ma soggetti che si relazionano attivamente con le pagine.

Negli anni a seguire sono stati fatti ulteriori passi in avanti con la nascita di *Flash*, un plugin che rendeva possibile l'integrazione di animazioni client-side nelle pagine web e con l'introduzione di *AJAX (Asynchronus JavaScript and XML)*, una tecnica che ha reso possibile l'esecuzione di richieste al server in maniera asincrona ed il conseguente aggiornamento dell'intera interfaccia o di alcune sue parti senza dover ricaricare l'intera pagina. In termini di interattività e dinamicità, questa segna un'enorme svolta nella creazione di siti web dinamici.

Durante il primo decennio degli anni 2000 nasce *HTML5*, l'ultima evoluzione dello standard HTML, che segna una vera e propria innovazione nella progettazione delle pagine web; mantenendo piena retrocompatibilità con le precedenti versioni, introduce nuovi elementi che migliorano la navigazione in termini di multimedialità e interattività. In tal modo, si riesce ad integrare all'interno di una pagina web: animazioni, audio, video e contenuti multimediali senza dover ricorrere a plugin di terze parti sviluppati in linguaggi di scripting. Un'altra grande novità legata a questo standard è la possibilità di utilizzare nuovi tag creati ad hoc per associare un significato semantico alla struttura delle pagine, come ad esempio `<header>`, `<nav>` o `<footer>`, in modo da facilitare la scansione del codice da parte dei motori di ricerca e

migliorare così l'indicizzazione delle pagine.
Dunque *HTML5*, unitamente alla progettazione e alla diffusione di layout responsivi, introduce tutti gli elementi necessari per rendere possibile la diffusione delle applicazioni web.

In questi ultimi anni, in seguito al continuo aggiornamento dei browser, le applicazioni web sono sempre più ricche di funzionalità. Dal 2016 nella conferenza annuale di Google sono state introdotte le *PWA (Progressive Web App)*; applicazioni web che, quando sono in esecuzione su dispositivi mobile, si comportano nello stesso modo delle applicazioni native sia in termini di efficienza che di potenzialità.

2.2 Architettura web

Nell'ambito dell'architettura del software, le applicazioni web rientrano nella categoria dei sistemi distribuiti. I sistemi distribuiti sono composti da più processi che cooperano tra loro perseguendo un unico obiettivo. I processi, nella maggior parte dei casi, risiedono su macchine diverse e comunicano attraverso la rete scambiandosi messaggi in ottemperanza a ben definiti protocolli.

Figura 2.3: Sistema distribuito

Tutti i sistemi distribuiti godono dei seguenti vantaggi:

- sono **scalabili**: il carico di elaborazione viene suddiviso tra più utenti e tra più macchine; quindi, si possono realizzare compiti più complessi che un singolo utente o elaboratore non riuscirebbe a trattare;

- garantiscono una alta **disponibilità**: essendo il calcolo distribuito tra più macchine se anche una sola necessita di manutenzione, comunque, tutto può continuare a funzionare e il servizio può essere erogato;
- è possibile **condividere le risorse** con un approccio cloud-oriented, guadagnando, così, in termini di capacità della singola macchina.

Di fronte alla possibilità di godere dei grandi vantaggi che le applicazioni distribuite offrono, ci si deve anche confrontare con alcune problematiche:

- la complessità di un sistema distribuito è molto elevata: sia dal punto di vista implementativo perché bisogna gestire la concorrenza, sia dal punto di vista di testing, sia dalla messa in opera del sistema;
- bisogna gestire i guasti improvvisi e parziali di un singolo processo, che potrebbero influenzare i processi che interagivano con esso;
- bisogna tenere in considerazione tutti i fattori critici connessi alla rete:
  - il limite di banda e l'inaffidabilità della rete perché non è assolutamente garantito che tutti i messaggi vengano consegnati;
  - il ritardo nella comunicazione tra processi;
  - l'eterogeneità della rete, composta da dispositivi differenti con diverse topologie;
  - la sicurezza, per evitare che l'applicazione sia esposta ai numerosi rischi della rete.

Per la gestione di questi problemi, al livello applicativo della pila protocollare del TCP/IP, è necessario progettare e implementare il software curandone la struttura, la scelta degli elementi che lo comporranno e le relazioni tra gli stessi.

L'interazione e lo scambio di messaggi tra i processi distribuiti avvengono secondo uno specifico modello che può essere client-server (il client invia le richieste al server il quale le serve e invia la risposta) o peer-to-peer (l'interazione avviene attraverso uno scambio simmetrico di messaggi tra pari).

Le applicazioni web sono un tipo di sistema distribuito, basato sulle tecnologie e sugli strumenti del web che adottano come modello di interazione il client-server.

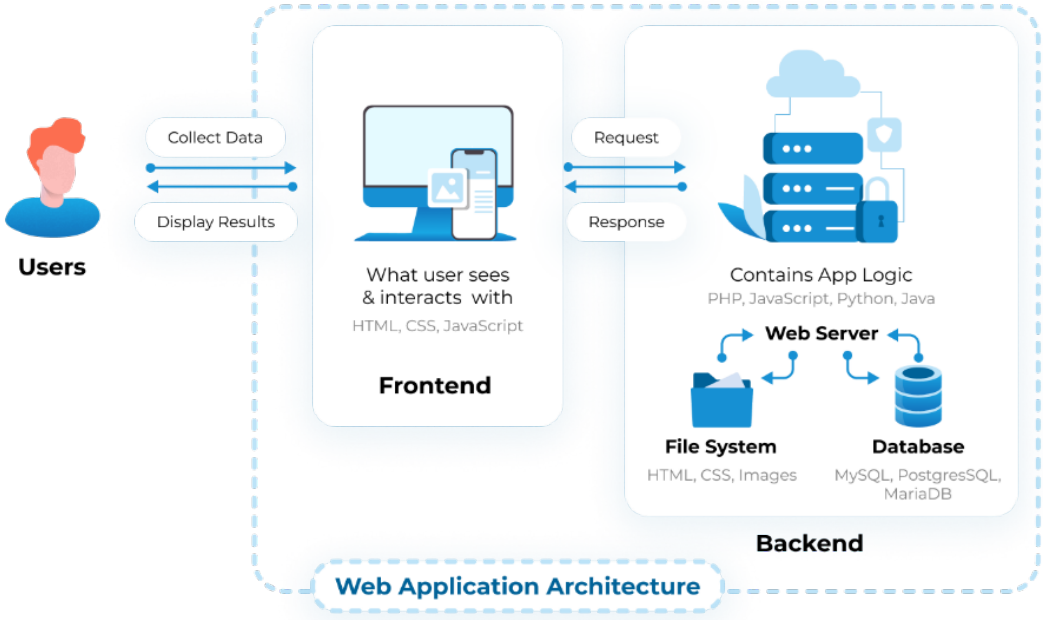

Figura 2.4: Architettura di un'applicazione web

L'architettura può prevedere due o più tier fisici, di cui sicuramente due di essi saranno:

- **front-end** (client side): eseguito nel contesto di un browser web e in grado di rappresentare l'interfaccia utente grafica (GUI) e di dare accesso remoto al sistema tramite il protocollo HTTP;
- **back-end** (server side): costruito a sua volta con un modello stratificato, che comprende tre tier logici che interagiscono tra loro:

  Presentazione: è il livello che è responsabile di inoltrare le richieste e di gestire le risposte verso le *API (Application Programming Interface)* tramite i verbi `GET`, `POST`, `PUT`, `DELETE` dello standard HTTP;

  Elaborazione: implementa la logica applicativa e manipola i *DTO (Data Transfer Objects)* che contengono le rappresentazioni dei concetti propri del dominio applicativo;

  Gestione dei dati: gestisce la persistenza dell'applicazione e garantisce la consistenza dei dati utilizzando uno o più DBMS.

Il modello a strati permette alle applicazioni di mantenere una buona modularità e di separare logicamente i concetti attraverso i moduli.

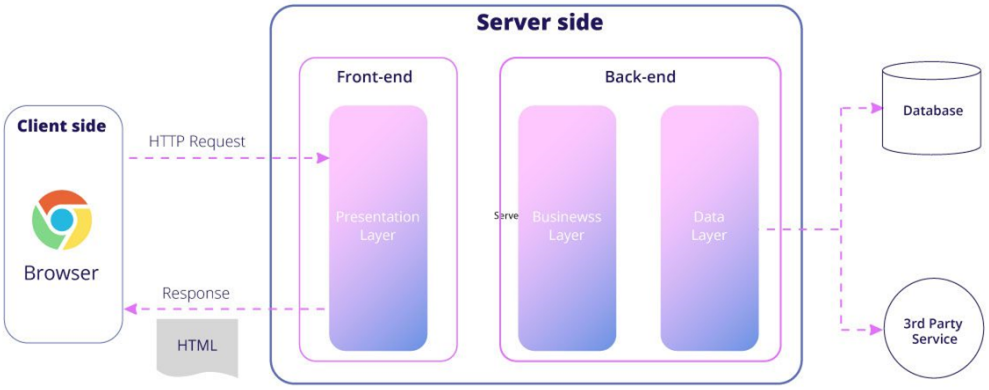

Figura 2.5: Struttura a livelli di un'applicazione web

2.3 Pattern architetturali

La progettazione della struttura un applicazione web può avvenire secondo diversi pattern architetturali.

2.3.1 Multi-Page Application

È il pattern secondo il quale il client per ogni pagina da mostrare esegue una richiesta al server il quale, dopo aver elaborato la richiesta, restituisce il contenuto della pagina con le relative risorse. Quindi la navigazione avviene tra diverse pagine HTML ed ogni pagina esegue opzionalmente del codice JavaScript al fine di personalizzare e rendere dinamico il contenuto. Più nel dettaglio, il browser richiede la pagina HTML mentre il codice JavaScript richiede i dati (tipicamente in formato JSON) e la logica dell'applicazione viene gestita sia lato server, sia lato client.

Secondo questo pattern si esegue un rendering server-side, in quanto è il server che invia al client la pagina corretta da visualizzare e quindi, bisogna attendere sempre il caricamento della nuova pagina.

Un esempio di applicazione di questo pattern si ha nei Content Management System[1].

[1] *CMS (Content Management System)*: software, installato su un server web, per la gestione dei contenuti di siti web

2.3.2 Single-Page Application

È un pattern che rappresenta l'evoluzione di quello tradizionale, secondo il quale sul server risiede **una singola pagina** dell'applicazione che è esattamente la stessa per ogni diverso URL. La pagina, tipicamente un file HTML privo di contenuti, tramite l'esecuzione del codice JavaScript, in risposta alle azioni che l'utente esegue durante la navigazione, riesce a modificare il contenuto visualizzato. Quindi, il contenuto della struttura del DOM[2] si modifica dinamicamente e di conseguenza si riduce il tempo di caricamento delle pagine che avviene quasi istantaneamente perché non si deve attendere che la nuova risorsa HTML arrivi dal server.

In questo caso il front-end gestisce interamente la visualizzazione del contenuto e l'interazione con il server avviene unicamente per lo scambio dei dati da mostrare. Si riduce così il carico di lavoro del server, mentre aumentano molto le responsabilità del browser e del front-end nella gestione dell'applicativo.

I vantaggi principali legati a questo pattern sono i seguenti:

- miglioramento dell'esperienza utente che diventa simile a quella delle applicazioni desktop;
- velocità di esecuzione dell'applicativo e riduzione dei tempi di attesa durante la navigazione (se si sfrutta la cache si può anche consentire la navigazione offline);
- possibilità di gestire più facilmente la responsività dell'applicazione;
- tutta l'applicazione viene implementata in un unico punto e, tramite gli strumenti di sviluppo integrati nei browser, è anche più semplice fare il debug ispezionando il DOM e monitorando i dati che vengono scambiati con il server attraverso la rete;
- si può riutilizzare il codice del back-end per sviluppare app mobile native.

Esistono però anche degli svantaggi ad esso correlati:

[2] *Document Object Model*: (spesso abbreviato come DOM), letteralmente modello a oggetti del documento, è una rappresentazione ad albero della pagina HTML, a partire dal tag radice, passando per ogni nodo figlio. È lo standard ufficiale del W3C per la rappresentazione di documenti strutturati.

- il primo punto di accesso all'applicazione potrebbe essere lento in condizioni di scarsa connettività;

- se il dispositivo su cui si esegue l'applicazione non fosse sufficientemente prestante si potrebbe incorrere in un'esperienza povera in termini di velocità;

- i motori di ricerca e i *SEO (Search Engine Optimization)*, poiché non hanno pagine HTML da ispezionare hanno più difficoltà ad indicizzare l'applicativo;

- poiché il cambiamento del contenuto pagina avviene dinamicamente nel contesto della stessa pagina la cronologia del browser perde di efficacia;

- bisogna curare con molta attenzione la logica sul client perché essendo essa esposta agli utenti finali può incorrere in problemi di sicurezza, tra i più noti il Cross-Site Scripting[3].

2.3.3 Altri pattern

Esistono altri pattern architetturali che si stanno diffondendo nell'ambito dello sviluppo web e sono i seguenti:

Isomorphic Application: è un pattern che mette insieme SPA con MPA, ottenendo un comportamento ibrido secondo il quale alcune pagine vengono costruite dinamicamente lato client mentre altre vengono restituite dal server.

Progressive Web Application: è un pattern pensato principalmente nell'ottica di dispositivi mobile, secondo il quale l'applicativo web, sviluppato con tecnologie web, tende ad integrarsi con il sistema operativo e ad essere molto simile ad un'applicazione nativa. Una PWA può essere "installata" tramite un bottone all'interno del browser; compare nella lista delle app native installate e consente all'utente di ricevere notifiche push.

[3] *XSS (Cross-Site Scripting)*: vulnerabilità che consente ad un attaccante remoto di "iniettare" script dannosi nelle singole pagine con lo scopo di rubare informazioni riservate o installare malware sui browser degli utenti.

2.4 Strumenti e tecnologie di sviluppo

Lo sviluppo di applicazioni web avviene rispettando il ciclo di vita del software come per qualunque altra applicazione: dalla fase di ideazione, alla progettazione, fino allo sviluppo. Per lo sviluppo del front-end sono necessari i seguenti strumenti: un IDE[4], un server applicativo installato in locale, eventualmente un database in locale e il sistema di versionamento del codice.

Relativamente al livello di presentazione le tecnologie basilari utilizzate sono: HTML5, JavaScript e il linguaggio CSS (Cascading Style Sheet) per personalizzare la grafica e il layout delle pagine HTML. Spesso, a corredo di queste tecnologie si utilizzano dei framework web che semplificano lo sviluppo per i compiti comuni dei software nell'ottica del riutilizzo del codice. Tra i più comuni vi sono: *React.js, Angular, Vue.js, Ember.js e Svelte.*

Relativamente al livello di elaborazione lato back-end, i linguaggi principalmente adottati per lo sviluppo sono: *PHP, Python, Java e Kotlin.* Anche in questo caso esistono framework a supporto molto noti tra cui: *Spring, Laravel, Django, Phoenix, Express.js.*

Figura 2.6: Framework web front-end e back-end più utilizzati

[4] *IDE (Integrated Development Environment)*: software che offre tutta una serie di strumenti e funzionalità che supportano il programmatore nello sviluppo e debugging del codice sorgente di un programma.

Capitolo 3

React

React è una libreria JavaScript nata per lo sviluppo di interfacce grafiche interattive e dinamiche con il fine di supportare una navigazione ricca e articolata. È stata sviluppata e lanciata nel 2013 da *Facebook* dall'ingegnere Jordan Walke e, poiché distribuita in forma open source, negli anni a seguire è molto cresciuta ed ha trovato largo consenso nelle community di sviluppatori. Ad oggi le applicazioni web che fanno uso di questa tecnologia sono moltissime tra le quali troviamo *Facebook, WhatsApp, Instagram, PayPal, Uber, Netflix e AirBnb*. L'obiettivo primario è quello di suddividere la User Interface (UI) in un insieme di componenti riutilizzabili, cercando di semplificarne la struttura e la relativa gestione dello stato.

Il grande successo di React e l'affermazione crescente della libreria rispetto ad altre preesistenti sul mercato, come ad esempio *Angular.js o Ember.js*, è legato a diversi fattori:

- **Efficienza**: ha una notevole di velocità di esecuzione nonché ottime prestazioni;
- **Flessibilità**: si integra facilmente rispetto ad altri framework in un progetto già esistente, può essere usato sia lato client sia lato server e si possono realizzare oltre che applicazioni web anche app native con *React Native*;
- **Semplicità**: nasce con la premessa di implementare solo il livello *View* invece dell'intero stack MVC il che semplifica il suo utilizzo;

- **Riusabilità**: la principale caratteristica architetturale di suddividere la vista in componenti aumenta la riusabilità del codice ed agevola la gestione di progetti molto ampi;
- **Documentazione**: poiché è mantenuto da Facebook c'è molta documentazione sul web che aumenta costantemente anche per l'altissimo numero di sviluppatori presenti nella community (i contributori del progetto su GitHub, ad oggi [nov, 2021], sono più di 1500, un numero davvero molto alto) il che velocizza anche la fase di apprendimento della libreria;
- **Retrocompatibilità**: rispetto ad altri framework la migrazione da una versione ad un'altra avviene più facilmente.

Nel seguente grafico è rappresentata la percentuale di utilizzo di React rispetto al totale delle applicazioni web esistenti. È possibile notare la costante predominanza sugli altri framework negli ultimi cinque anni:

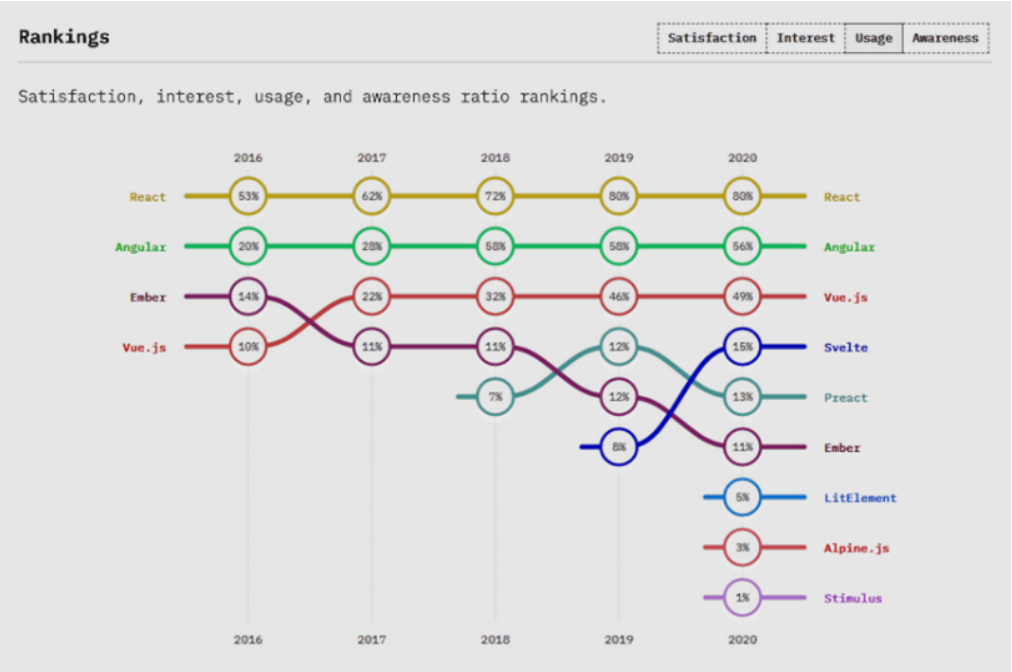

Figura 3.1: Percentuale di utilizzo dei maggiori framework front-end negli anni 2016-2020. [20]

Secondo il blog *LambdaTest*, che ogni anno produce infografiche e articoli sulle tecnologie di sviluppo più utilizzate, alla fine del 2019 il numero delle

applicazioni web sviluppate con React era pari a 475K [6]. Presumibilmente negli ultimi due anni potrebbe essere cresciuto ancora di molto.

Per quanto riguarda il grado di soddisfazione degli sviluppatori, nel seguente grafico viene rappresentato, sui toni del rosso, il valore in percentuale, di quanti lo riutilizzerebbero dopo la prima esperienza o di quanti sono interessati ad approcciarsi al framework. Anche in questo caso, la percentuale di apprezzamento totale è pari all'81.6% e questo valore lascia ben sperare sul futuro di React.

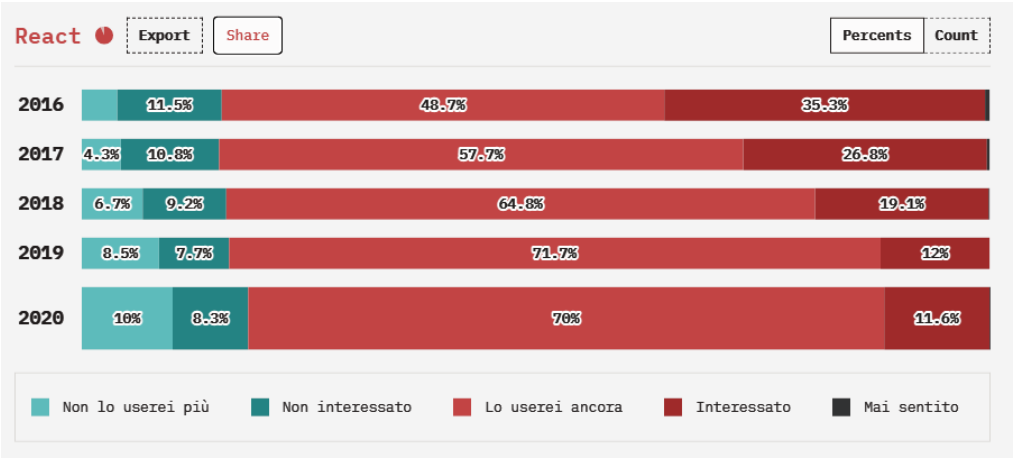

Figura 3.2: Percentuale di soddisfazione-interesse del framework React negli anni 2016-2021. [20]

3.1 Principi Generali

React è una libreria **dichiarativa** in quanto si può creare l'interfaccia e il sistema di eventi senza mai dover interagire direttamente con il DOM. Nella progettazione infatti, si modella lo stato del componente e non le operazioni necessarie per costruirlo. Al contrario, l'approccio imperativo (che si avrebbe ad esempio usando *jQuery*) prevede che si dica al browser esattamente cosa fare e come modificare la struttura del DOM. Inoltre, come riportato nella documentazione online della libreria: "La natura dichiarativa dell'UI rende il codice più prevedibile e facile da debuggare" [15].

React è una libreria basata su **componenti** che possono essere creati in isolamento e che sono **componibili** tra loro per creare UI complesse. La linea guida ideale consiste nella creazione di componenti piccoli ed indipendenti,

possibilmente riusabili che poi vengono composti per aumentare la complessità dell'intera interfaccia. Ad ogni cambiamento di stato i componenti sono in grado di aggiornare e ridisegnare solo le relative parti di UI che dipendono da esso e senza ricaricare l'intera pagina. Questo semplifica molto la fase di creazione di un'interfaccia grafica interattiva.

React sfrutta il concetto dell'**immutabilità**: un po' come accade per le costanti o per le stringhe così in React lo stato è immutabile, ciò vuol dire che non cambia mai direttamente, ma solo tramite apposite funzioni. In questo modo tutte le mutazioni possono essere centralizzate e si ha un'idea di prevedibilità perché si sa esattamente quando avviene il cambiamento. Questo migliora anche le prestazioni perché mutare un oggetto già esistente è sempre più dispendioso rispetto alla possibilità di ricrearlo dal principio.

3.2 Creazione e struttura di un progetto

La creazione di un progetto React avviene tramite il comando `npx create-react-app <nome-progetto>` invocato da terminale oppure automaticamente da un ambiente di sviluppo integrato nel quale viene prodotto il codice sorgente. Questo comando scatena l'inizializzazione di una catena di strumenti necessari per:

- la traduzione del codice ad opera di *Babel* e l'impacchettamento ad opera di *Webpack* affinché il sorgente, che molto probabilmente utilizza concetti più moderni di JavaScript ES6 (classi e moduli), sia interpretabile da tutti i browser;
- avviare un server locale (tipicamente un'istanza di *Node.js*) per la fase di debug.

Il progetto creato è strutturato in diverse parti:

- il file `package.json` contiene le informazioni sulla configurazione del progetto;
- il file `package-lock.json` contiene un elenco dettagliato delle librerie, con la relativa versione e il codice hash.[1];

[1]Un codice hash è una stringa di lunghezza fissa calcolata con una funzione crittografica a partire da un generico dato. A causa delle sue particolari proprietà di univocità si

- la cartella `node_modules`, ricreabile con il comando `npm install`, che contiene i sorgenti scaricati delle librerie importate;
- la cartella `public` che racchiude l'unico file HTML del progetto (unico in quanto il pattern architetturale è il Single Page Application), il file `manifest.json` e le risorse pubbliche che i browser e i SEO possono ispezionare;
- la cartella `src` contiene i sorgenti JavaScript del progetto (tra cui `index.js` che è il punto di partenza dell'applicazione), i fogli di stile ed eventuali altri file.

Durante lo sviluppo del progetto si possono eseguire i seguenti script di base:

- `npm start`: avvia l'esecuzione in debug sull'istanza del server, eseguendo il ricaricamento automatico dell'applicazione per ogni modifica effettuata all'interno del codice (ciò velocizza notevolmente lo sviluppo e la possibilità di individuare eventuali errori);
- `npm test`: lancia il test runner in modalità interattiva;
- `npm run build`: pubblica il progetto in versione definitiva su un server di terze parti;
- `npm run eject`: rimuove le dipendenze di build e modifica le scelte di configurazione, perdendo così la possibilità di aggiornare tutte le librerie automaticamente ma guadagnando flessibilità nella configurazione di Babel e Webpack. Solitamente si chiama questo comando quando si vuole integrare il progetto React in un altro più grande che utilizza differenti tecnologie.

L'aggiornamento automatico dell'applicazione è possibile perché React per effettuare il rendering mantiene una copia virtuale del DOM, detto **Virtual DOM**, che permette al browser di usare meno risorse quando è necessario apportare modifiche ad una pagina. In dettaglio, ogni volta che lo stato di un componente cambia, ovvero si "sporca", React aggiorna il DOM virtuale dei

---

utilizza per identificare univocamente un messaggio oppure per verificare l'integrità dei dati ricevuti da canali potenzialmente non sicuri. Questa verifica viene fatta confrontando il valore di hash dei dati ricevuti con quello dei dati originari per stabilire se durante il trasferimento essi hanno subito delle modifiche.

componenti marcati come "sporchi", poi con un algoritmo di diffing controlla le differenze che tali cambiamenti comportano raggruppandole tutte insieme ed infine esegue un unico aggiornamento al DOM reale. In questo modo il browser esegue il ridisegno e il ricalcolo del layout una sola volta risparmiando risorse.

3.3 Componenti

Un componente è un elemento modulare e isolato che è parte dell'interfaccia e che può essere combinato con altri per creare componenti più complessi nell'ottica del riuso del codice e della specializzazione del componente stesso. L'interfaccia ottenuta risulterà la composizione tra componenti. L'isolamento tra un componente e l'altro agevola la fase di sviluppo, in quanto, le modifiche su un componente non coinvolgono direttamente gli altri; inoltre, la modularità permette di estendere le funzionalità dell'applicativo molto facilmente.

I componenti fino a qualche anno fa venivano realizzati tramite classi JavaScript; al giorno d'oggi invece alle classi si preferiscono le funzioni. Nel loro corpo viene descritta la resa grafica e il comportamento da adottare per ogni cambiamento di stato. Ogni componente deve avere un nome univoco nel contesto del progetto; può ricevere in ingresso dati o funzioni (entrambi immutabili) tramite le props ed eventualmente mantenere dati propri attraverso un suo stato interno.

Le funzioni utilizzate per creare i componenti sono dette "pure" perché producono un risultato che dipende esclusivamente dalle props ricevute in ingresso senza lasciare effetti collaterali sull'ambiente circostante; in altre parole, l'output prodotto è sempre lo stesso ogni qualvolta i dati di input sono gli stessi.

La rappresentazione del componente viene trasformata nell'albero DOM e di conseguenza visualizzata nel browser. Per la renderizzazione del componente padre di tutta l'applicazione viene invocata la seguente richiesta all'interno del file `index.js`:

```
ReactDOM.render(<MainComponent/>, document.getElementById("root"));
```

"React riconosce il fatto che la logica di renderizzazione è per sua stessa natura accoppiata con le altre logiche che governano la UI: la gestione degli

eventi, il cambiamento dello stato nel tempo, la preparazione dei dati per la visualizzazione. Invece di separare artificialmente le tecnologie inserendo il codice di markup e la logica in file separati, React separa le responsabilità utilizzando unità debolmente accoppiate chiamate "componenti" che contengono entrambi." [7]

Tipicamente, si adotta la convenzione di utilizzare un file JavaScript per ogni componente all'interno del quale si può importare (con il comando `import`) tutto quanto è necessario ed esportare (tramite il comando `export`) ciò che si vuole rendere disponibile agli altri componenti fuori dal file. Se una funzione o un componente, nel contesto di un file, non vengono esportati essi saranno solo utilizzati all'interno dello stesso file in cui sono stati creati. L'intero componente creato nel file può essere esportato tramite `export default <NomeComponente>`. Si noti che se nello stesso file possono essere definiti più componenti ma solo quello esportato di default sarà disponibile a chi importa il file. È anche possibile realizzare un export nominale: vengono assegnati nomi arbitrari agli elementi da esportare (es. `export {NomeDiFantasia}`).

Insieme alla creazione di un file per un componente, possono essere creati anche un file per i relativi test da effettuare su di esso ed eventualmente anche un file per lo stile CSS che gli si vuole attribuire.

3.3.1 JSX (Javascript Syntax eXtension)

Quando si crea un componente React, solitamente tramite funzioni JavaScript piuttosto che classi, ciò che viene restituito per definire la UI del componente è un albero JSX. Formalmente JSX è definibile come "un'estensione della sintassi del linguaggio JavaScript che fornisce un modo per strutturare il rendering dei componenti usando una sintassi familiare a molti sviluppatori."[7]. Quindi, l'utilizzo di JSX, dà al programmatore l'impressione di utilizzare una sintassi dichiarativa, molto simile al linguaggio HTML.

Poiché i browser sono in grado di interpretare solo il codice JavaScript puro, è necessario che avvenga un processo di traspilazione per far sì che il codice venga eseguito. Tale processo si occupa di sostituire il codice JSX in chiamate a funzioni JavaScript comprensibili dai browser. In base a quanto indicato nella documentazione ufficiale di React: JSX *"fornisce zucchero sintattico per la scrittura di funzioni della forma:"*[15]

```
React.createElement(component, props, ...children);
```

Il modulo transpiler, per i progetti basati su JavaScript, è *Babel*, già impostato come opzione predefinita alla creazione di un progetto. Ad esempio dato il componente:

```
import React from "react";

const PersonalController = () => {
  return (
    <CustomButton color="red" shadowSize={2}>
      {"Cliccami!"}
    </CustomButton>
  );
}

export default PersonalController;
```

il componente `<CustomButton>` nella fase di transpilazione viene tradotto nel seguente modo:

```
React.createElement(
  CustomButton,
  {color: "red", shadowSize: 2},
  "Cliccami!"
);
```

Come si può notare, nonostante non sia obbligatorio utilizzare la sintassi JSX ci sono notevoli vantaggi nell'adottarla perché è un aiuto *"visuale"* per la costruzione del codice dell'UI. Inoltre, un altro importante aspetto da considerare, è che *"React DOM effettua automaticamente l'escape di qualsiasi valore inserito in JSX prima di renderizzarlo. In questo modo, garantisce che non sia possibile iniettare nulla che non sia esplicitamente scritto nell'applicazione. Ogni cosa è convertita in stringa prima di essere renderizzata. Questo aiuta a prevenire gli attacchi XSS (cross-site-scripting)"*[15].

Se, durante la definizione di un componente, si vuole mescolare codice JavaScript insieme al JSX è necessario racchiuderlo tra parentesi graffe. Nel seguente esempio, vediamo come è possibile ciclare una lista tramite la funzione `map` di JavaScript, che è annidato in JSX e che contiene anche al suo interno altro JSX. Si noti che ad ogni componente viene assegnata una `key` che è indispensabile e che deve essere univoca perché è utilizzata da React per

individuare lo specifico componente da ridisegnare nel caso in cui avvenga un cambiamento di stato.

```
const Users = () => {
  const users = [
    {id: 1, name: "Arianna"},
    {id: 2, name: "Daniele"},
  ];

  return (
    <div className="Example">
      {users.map(user =>
        <MyButton
          key={user.id}
          onClick={() => console.log("Clicked!")}
        >
          {user.nome}
        </MyButton>
      )}
    </div>
  );
}
```

La funzione `render()` può restituire un solo nodo; quindi, se l'albero ne contiene più di uno, è necessario che questi siano posti tutti all'interno di un nodo padre. A questo proposito si può utilizzare un nodo atto a contenere frammenti di DOM: `<React.Fragment>`. Questo assume il ruolo di contenitore di nodi e la sua peculiarità è che non produce nessun tag nell'output. Nonostante JSX sia molto simile al linguaggio HTML essi rappresentano concetti differenti sia dal punto di vista semantico, in quanto il JSX è formalmente JavaScript, sia dal punto di vista sintattico, perché:

- un tag deve essere sempre chiuso (si può usare anche il tag auto-chiudente);
- JSX utilizza la notazione cammello per il nome degli attributi (es. `onclick` diventa `onClick`);
- poiché `class` e `for` sono parole riservate del linguaggio JavaScript, questi diventano rispettivamente `className` e `htmlFor`;
- gli attributi, detti props nel contesto del JSX, non devono necessariamente essere stringa ma possono essere espressioni JavaScript di qualsiasi tipo.

Per convenzione un elemento HTML nativo è identificato dal suo rispettivo tag la cui lettera iniziale è minuscola. Un elemento JSX invece, è identificato dalla lettera iniziale maiuscola: il tag viene interpretato come nome di un componente React. Se si vuole definire uno stile CSS per un componente JSX è necessario racchiudere le regole in un dizionario ed associarlo al componente tramite la proprietà `style`. Tale dizionario deve avere come chiavi le proprietà del CSS, definite anch'esse tramite notazione cammello, alle quali associare i rispettivi valori. In questo modo, ogni componente è in grado di incapsulare completamente il proprio stile. Di seguito è riportato un esempio.

```
const buttonStyle = {
  backgroundColor: "blue",
  margin: "5px"
};

const CustomButton = () => {
  return (
    <div>
      <button style={buttonStyle}>Send</button>
    </div>
  );
}
```

3.3.2 Ciclo di vita di un componente

Mentre l'applicazione si esegue, ogni istanza di un componente React, contrassegnata da un id univoco, attraversa tre fasi del ciclo di vita:

1. **Montaggio**: corrisponde alla prima creazione di un componente, quindi viene eseguita per la prima volta la relativa funzione JavaScript e il risultato restituito viene immesso nell'albero DOM.
2. **Aggiornamento**: è la fase di ridisegno del componente, che avviene quando le props o lo stato interno subiscono un cambiamento. In questo caso viene rieseguita la funzione del componente, il risultato restituito viene confrontato con il DOM esistente e si aggiornano solo le parti dell'interfaccia interessate dal cambiamento.
3. **Smontaggio**: corrisponde alla rimozione dell'elemento dal DOM. Questo può accadere se durante la fase di aggiornamento alcuni componenti non necessitano più di essere mostrati.

La fase di aggiornamento può ripetersi più volte durante la vita di un componente a differenza delle fasi di montaggio e smontaggio che avvengono rispettivamente una sola volta.

Figura 3.3: Ciclo di vita di un componente

3.3.3 Props

React utilizza il concetto di *flusso unidirezionale dei dati*: la condivisione dello stato e il trasferimento dei dati avviene solo in un'unica direzione e mai al contrario. Si procede con un approccio di tipo top-down, quindi dal componente padre, contenitore, verso il componente figlio che è suo diretto discendente. Sulla base di questo concetto, le props vengono utilizzate per condividere informazioni tra componenti all'interno di una gerarchia. Il contenuto delle props è immutabile, cioè non può mai essere modificato all'interno del componente che lo riceve. Ogni props ha un nome univoco nel contesto di un componente e può assumere come valore:

- una stringa, racchiusa tra apici, come avviene anche per un attributo HTML;
- un dato JavaScript racchiuso tra graffe, del tipo `array, boolean, number, object`. Nel caso si voglia passare un oggetto definito inline si avrà un doppio livello di parentesi.
- qualunque espressione JavaScript, racchiusa tra parentesi graffe.

La possibilità di passare una funzione come props è un aspetto molto interessante, perché questo consente di centralizzare la logica applicativa di un

componente al livello superiore. Infatti, un componente figlio potrebbe occuparsi solo della presentazione e demandare la logica di gestione al padre invocando invocando la funzione che gli è stata passata.

```
const CardContainer = () => {
  const name = "Bob";

  return (
    <UserCard
      name={name}
      imgPath="/imgs/bob.jpg"
      style={{padding: "3px"}}
    />
  );
}

const UserCard = (props) => {
  return (
    <div style={props.style}>
      <img src={props.imgPath} alt={props.name}/>
      <h1>{props.name}</h1>
    </div>
  );
}
```

Per accedere al contenuto delle props, all'interno del componente di livello inferiore, si utilizza il parametro unico **props** che rappresenta un oggetto JavaScript le cui chiavi coincidono con gli attributi JSX definiti nel componente superiore. Come si può notare dal precedente frammento di codice, il componente figlio `UserCard` riceve come parametro di ingresso l'oggetto `props` cosi strutturato `{name: "Bob", imgPath: "/imgs/bob.jpg", style: {padding: "3px"}}` di cui si avvale per inizializzare l'elemento da renderizzare. Esiste anche un modo alternativo e più compatto per accedere al contenuto delle props: tramite **assegnamento di destrutturazione** di un oggetto. Questo consiste nell'inserire tra parentesi graffe le variabili i cui nomi coincidono con le chiavi dell'oggetto props da destrutturare; i valori di tali variabili corrisponderanno ai relativi valori dell'oggetto props. Ad esempio, riconsiderando il frammento di codice precedente, utilizzando l'assegnamento di destrutturazione il componente `UserCard` sarebbe stato:

```
const UserCard = (props) => {
  const {name, imgPath, style} = props;

  return (
    <div style={style}>
      <img src={imgPath} alt={name}/>
      <h1>{name}</h1>
    </div>
  );
}
```

L'assegnamento di destrutturazione può anche essere anticipato nella lista dei parametri formali da passare alla funzione del componente.

```
const UserCard = ({name, imgPath, style}) => {...}
```

Il componente superiore può inglobare in esso un altro componente anche senza passargli tutte le props che gli sono necessarie. Per evitare problemi con props non assegnate, all'interno del componente figlio è possibile assegnare un valore di default alle props.
Esiste, inoltre, anche un altro operatore del JavaScript noto come **operatore di diffusione**, che torna utile al componente padre quando il numero di props da passare al figlio è alto. Anziché passare manualmente una props per volta, tramite l'operatore di diffusione il passaggio avviene automaticamente con la distribuzione delle proprietà dell'oggetto al componente che si sta invocando:

```
const CardContainer = () => {
  const user = {
    name: "Bob",
    imgPath: "/imgs/bob.jpg"
  }

  return (
    <UserCard name={user.name} imgPath={user.imgPath}/>
    <UserCard {...user}/> //operatore di diffusione
  );
}
```

Una props speciale è `children`, utilizzata per iniettare automaticamente nel componente inferiore il JSX passato nel componente padre come children, cioè ciò che compare tra il tag di apertura e di chiusura. Per chiarire il concetto si osservi il seguente esempio:

```
const Chapter = (props) => {
  return (
    <div style={{padding: "8px"}}>
      <h1>{props.title}</h1>
      {props.children}
    </div>
  );
}

const Section = (props) => {
  return (
    <div style={{paddingBottom: "5px"}}>
      <h2>{props.title}</h2>
      <p>content....</p>
    </div>
  );
}

const Thesis = () => {
  return (
    <Chapter title="React">
      <Section title="Props"/>
      <Section title="Stato di un componente"/>
    </Chapter>
  );
}
```

I due componenti `Section` sono passati come figli all'interno del componente `Chapter` tramite la props `children`. Questo meccanismo permette di disaccoppiare il componente `Chapter` dal suo contenuto rendendolo così più riusabile.

3.4 Stato di un componente

"Mentre le props permettono ad un componente di ricevere proprietà dal suo genitore, di essere istruito a stampare alcuni dati per esempio, lo stato permette a un componente di prendere vita da solo, ed essere indipendente dall'ambiente circostante."[3]

"Fino a un po' di tempo fa, i componenti di classe erano l'unico modo per definire un componente che avesse un proprio stato, e che potesse accedere ai metodi del ciclo di vita in modo da poter eseguire operazioni la prima volta che il componente veniva creato, quando veniva aggiornato o rimosso."[3]

L'introduzione di **React Hooks**, a partire dal 2019, ha apportato un'importante novità, permettendo la gestione dello stato, della sua evoluzione e l'esecuzione di effetti collaterali all'interno di componenti funzionali, rendendoli così molto più potenti.

Gli hooks sono funzioni il cui nome inizia per convenzione con *use*. Per ogni componente funzionale ne possono essere invocati molteplici e tutti nello stesso ordine ovvero non possono essere usati all'interno di un ramo condizionale del codice. Essi sono legati alla singola istanza di un componente, garantendo così l'isolamento tra uno stato ed un altro.

Tra gli hooks presenti all'interno della libreria React, che saranno analizzati in dettaglio nel seguito di questo paragrafo in quanto i più utilizzati, vi sono:

- `useState`;
- `useReducer`;
- `useContext`;
- `useEffect`;

ma libreria ne mette a disposizione tanti altri tra cui `useMemo, useCallback, useRef`. Inoltre, per gestire particolari comportamenti o esigenze è possibile definire hooks personalizzati, che danno al programmatore la possibilità di condividere stato e logica applicativa tra i componenti.

3.4.1 useState

È utilizzato per mantenere e gestire lo stato interno del componente che si modifica sulla base di eventi scatenati su alcune parti del componente stesso. `useState` accetta in ingresso il valore di inizializzazione dell'elemento e restituisce un array di due elementi: la variabile di stato che si aggiorna ad ogni variazione dello stato e la funzione che si utilizza per alterare tale stato. Per accedere agli elementi dell'array si può utilizzare l'assegnamento di destrutturazione come nel seguente esempio in cui si vuole gestire lo stato di un contatore:

```
const [count, setCount] = useState(0);
```

All'interno di un componente questo hook è richiamabile quante volte si desidera per creare tutte le variabili di stato di cui si necessita.

3.4.2 useReducer

Spesso, nel contesto di applicazioni articolate, lo stato da mantenere potrebbe diventare complesso e problematico: questo accade ad esempio in situazioni in cui l'aggiornamento di una variabile condiziona lo stato di un'altra. Per gestire uno stato articolato si utilizza il meccanismo azioni/riduzioni:

- ogni azione che può scatenare una modifica sullo stato è descritta in un oggetto la cui chiave `type` ne indica la tipologia;
- il riduttore è una funzione che:
  - accetta in ingresso lo stato attuale e il tipo dell'azione da eseguire;
  - restituisce in uscita un nuovo stato modificato sulla base dell'azione indicata.

Il riduttore contiene al suo interno un costrutto del tipo `switch (action.type) {...}` che in base al tipo di azione specificata, esegue gli step necessari per aggiornare lo stato.

Lo hook utilizzato per realizzare il meccanismo azioni/riduzioni prende il nome di `useReducer` che:

- accetta due parametri in ingresso: la funzione di riduzione e lo stato iniziale;
- restituisce una lista formata da due elementi: lo stato corrente e il `dispatch` che è in grado di inviare l'azione corretta alla funzione di riduzione.

Il `dispatch`, invocato in un componente, richiama la funzione di riduzione, aggiorna lo stato e il componente che lo ha invocato.

3.4.3 useContext

Se si vuole condividere lo stato tra più componenti è necessario che esso sia posizionato il più in alto possibile: infatti, ogni stato è condivisibile in direzione top-down con i componenti sottostanti tramite l'utilizzo delle props.

Quando, però, è necessario condividere lo stato con un componente che si trova diversi livelli più in basso la situazione potrebbe complicarsi, perché tutti i figli nel mezzo devono fungere da mezzo di passaggio pur non essendo direttamente interessati al dato che transita al loro interno. Un altro problema emerge quando è necessario condividere lo stato con un componente al di fuori della catena gerarchica.

Per gestire queste situazioni esiste lo hook `useContext` tramite il quale è possibile passare lo stato e permetterne l'aggiornamento senza dover ricorrere all'utilizzo delle props.

Il funzionamento è il seguente:

- si crea il contesto: `const MyContext = React.createContext();`
- si crea un componente *wrapper* che contiene tutti gli elementi che potranno accedere al contesto:

```
<MyContext.Provider value={myContext}>
  {children}
</MyContext.Provider>
```

- per accedere al contesto, all'interno di un componente children si usa il Consumer:

```
const Child = () => (
  <MyContext.Consumer>
    {myContext => (<div style={{color: myContext.color}}/>)}
  </MyContext.Consumer>
);
```

Per semplificare l'accesso al contesto dichiarato, al Consumer si preferisce lo hook `useContext`. Questo hook accetta come parametro il contesto da consumare e restituisce il valore corrente contenuto nel contesto.

```
const {color} = useContext(MyContext);
```

L'utilizzo congiunto degli hooks `useReducer` e `useContext` permettono di creare uno o più stati complessi e generali per la gestione di un intero applicativo.

3.4.4 useEffect

Questo hook è un'implementazione del design pattern *Observer* e si utilizza per eseguire appunto effetti collaterali ad ogni invocazione di un componente. Esso accetta due parametri: una funzione e un array che contiene una lista di variabili di stato da osservare. Il comportamento è il seguente: la funzione passata come parametro viene rieseguita dopo ogni render del componente e ogni volta che una variabile osservata cambia valore. Se si vuole far in modo che la funzione venga eseguita solo dopo il primo render è sufficiente passare un array vuoto in questo modo si ha la certezza che la funzione non sarà più eseguita.

```
useEffect((function), [var1, var2, ..., varN]);
```

3.5 Routing

Come visto in precedenza, le SPA sono in grado di eseguire l'intera applicazione nel contesto della stessa pagina. Questo però comporta un problema potenzialmente grave a livello di navigabilità: gli utenti, abituati alla navigazione in ottica multi-pagina, avvertono la costante necessità di essere guidati per sapere sempre quale pagina stanno visitando e a che livello di profondità si trovano rispetto alla radice del sito.

Nel contesto della MPA il meccanismo di navigazione predefinito dei browser fungeva da guida e da punto di riferimento fisso. Con React, invece, gli URL devono essere gestite manualmente. Tecnicamente si tratta di implementare il cosiddetto **meccanismo di routing** che si realizza solitamente ricorrendo alle funzioni della libreria *React Router*, la quale sfrutta internamente l'API della cronologia offerta dai browser per aggiornare dinamicamente l'URL ad ogni cambiamento di pagina e per simulare il corretto comportamento del bottone "indietro" del browser che altrimenti non sarebbe stato possibile realizzare. Per realizzare il tale meccanismo:

- si utilizza l'elemento `Router` alla radice della gerarchia visuale;
- all'interno di un componente `Switch` si racchiudono, nell'ordine dal più specifico al più generico, tutti gli elementi che appartengono ad un URL differente.

I figli di `Switch` sono componenti `Route` aventi le seguenti props:

- `path`: stringa da concatenare all'URL;
- `exact`: booleano che indica se il path termina esattamente con la stringa passata in path;
- `component`: il componente da renderizzare per la specifica route.

All'interno del componente renderizzato possono essere utilizzati degli hooks che la libreria mette a disposizione per supportare la gestione dell'instradamento ed in particolare:

- `useHistory()`: per avere accesso alla cronologia di navigazione;
- `useLocation()`: per avere accesso all'URL corrente;
- `useParams()`: restituisce un oggetto di chiavi/valori con i parametri dell'URL;
- `useRouteMatch(expression)`: confronta l'URL corrente con l'espressione fornita e restituisce l'oggetto corrispondente.

3.6 Interazione con il server

Il server, oltre ad ospitare l'intera interfaccia utente ed inviarla ai browser che ne fanno richiesta, mantiene al suo interno tutti i dati necessari per il corretto funzionamento dell'applicazione.

Il client effettua le richieste ogni qualvolta sono necessari nuovi dati da mostrare. Tali richieste vengono indirizzate agli specifici URL sui quali sono state pubblicate le API (Application Programming Interface) che espongono tutte le funzionalità necessarie. Le richieste vengono eseguite tramite i verbi del protocollo HTTP: `GET, POST, PUT, DELETE e PATCH`.

Per eseguire le richieste in maniera asincrona si può utilizzare la libreria *Axios Hooks* che mette a dispozione alcuni hooks che semplificano l'interazione con il server. Lo hook principale è `useAxios(config, options)` che accetta in ingresso due parametri:

- `config`: una stringa URL o un oggetto Axios.Config (che permette di specificare tutti i parametri di configurazione, tra cui URL, metodo HTTP da usare e parametri da allegare alla richiesta);

- `options`: un oggetto opzionale i cui campi indicano se si vuole fare uso della cache e se la richiesta deve essere eseguita in modo manuale, ovvero quando il programmatore decide di eseguirla;

e restituisce in uscita due valori:

- un oggetto con i campi `{data, loading, error, response}` che corrispondono agli stati in cui la richiesta si può trovare:

  `data`: si riempie con il corpo della risposta del server quando la richiesta termina con successo, altrimenti diventa `undefined`;

  `loading`: un booleano che vale `true` finché la richiesta è in esecuzione;

  `error`: contiene i dati dell'eventuale errore;

  `response`: un oggetto contenente la risposta completa inviata dal server;

- una funzione che permette di azionare la richiesta manualmente.

Capitolo 4

User eXperience Design

Il punto di forza delle applicazioni web è l'interattività: si cerca di rendere l'utente un protagonista attivo durante la navigazione. Affinché questa ambizione possa concretizzarsi, è necessario che la fase di progettazione dell'interfaccia web sia il più possibile orientata alle esigenze degli utenti e che l'interazione possa risultare efficace, efficiente, piacevole, facile da apprendere e da usare.

Infatti, il successo di un applicativo è fortemente legato all'empatia che l'utente prova nei suoi confronti mentre lo utilizza. I fattori che possono influenzare questo sentimento sono legati alle seguenti caratteristiche del software: **funzionalità, usabilità, versatilità e robustezza**. Inoltre, bisogna considerare anche il fattore della **gradevolezza** che, pur non essendo di natura tecnica, è fondamentale per garantire la sensazione di piacevolezza durante l'utilizzo del software; questo fattore è uno dei principali elementi che condiziona l'apprezzamento del prodotto nella sua complessità.

Questi concetti sono stati formalizzati nell'espressione *User eXperience* che definisce l'insieme delle sensazioni che un utente prova durante l'interazione con un prodotto. Per garantire un'esperienza utente gradevole è buona norma, in fase di progettazione, immedesimarsi nell'utente finale, considerando i possibili diversi ruoli che egli possa impersonare e immaginando quali possano essere le azioni compiute abitualmente, cercando di renderle il più possibile intuitive e semplici e offrendo suggerimenti testuali o visuali su come possa avvenire l'interazione. È molto importante che tali suggerimenti siano in linea con le funzionalità implementate per evitare sentimenti negativi legati alla delusione e all'insoddisfazione.

Inoltre, considerando la totalità dell'applicazione, per garantire una resa grafica piacevole e chiara è necessario anche mantenere **coerenza** dal punto di vista grafico, stilistico, progettuale e soprattutto dal punto di vista semantico: i layout, le icone, le immagini, i font, le indicazioni testuali e il significato che si attribuisce ad ognuno di loro deve mantenere coerenza per non confondere e demoralizzare l'utente.

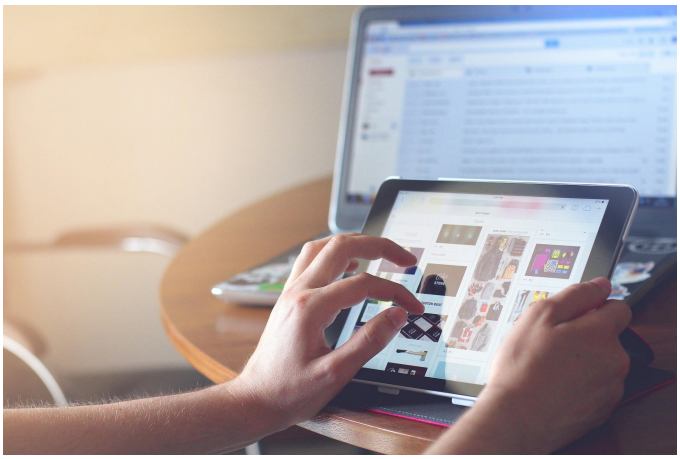

Figura 4.1: User eXperience

Sulla base di queste considerazioni, la realizzazione dell'interfaccia è un compito abbastanza delicato e complesso e richiede la collaborazione di diverse figure professionali. Il risultato deve sempre privilegiare l'utente finale, nell'ottica della *User Centered Design*. Per realizzare l'interfaccia è necessario procedere per passi, reiterati più volte, ognuno dei quali viene descritto nel seguito.

Analisi del contesto

È necessario analizzare il contesto da diversi punti di vista:

- **utente**: individuare chi sono gli utenti, i loro desideri e necessità, che livello di esperienza hanno, quali sono diversità linguistiche, culturali e cognitive che li differenzino;

- **applicativo**: dell'applicativo: individuare quali sono le funzionalità da offrire e i dati da mostrare;

- **interazione**: individuare le azioni elementari e stabilire come visualizzare i dati nella maniera più chiara possibile. Si possono anche preconfezionare casi d'uso per individuare particolari azioni che un utente può compiere per realizzare un compito.

Ideazione

È la fase in cui vengono raccolte le idee, definito il prototipo dell'interfaccia e i componenti che la comporranno sulla base del contesto analizzato, delle azioni e delle finalità del prodotto.
Per problemi comuni e casi d'uso simili tra diversi applicativi, devono essere individuati pattern progettuali già noti ai quali gli utenti sono già abituati e per i quali esistono relativi pattern di sviluppo già ampiamente utilizzati.

Realizzazione della GUI

Dopo aver ideato l'interfaccia, si procede con la realizzazione della GUI utilizzando l'approccio *divide et impera*, cioè suddividendo il problema generale in piccole diverse interazioni. Durante la realizzazione bisogna:

- prevedere eventuali configurazioni alternative responsive per diversi dispositivi fisici con diverse caratteristiche fisiche. Tutte le configurazioni devono essere consistenti ossia devono mostrare gli stessi contenuti e dati, offrire le stesse funzionalità e avere un aspetto grafico coerente alle altre;
- prevedere eventualmente, un'interazione continua, cioè che possa ad esempio, iniziare su un dispositivo e continuare in un altro mantenendo sincronia durante il passaggio.
- rispettare gli obiettivi di usabilità (efficacia, efficienza, sicurezza, utilità, apprendibilità, memorabilità) e dare sempre all'utente un feedback sull'esito delle operazioni svolte o ancora in corso.

Eseguire Test

In seguito alla realizzazione seguono sempre diverse fasi di test necessarie per capire l'efficacia del prodotto e il livello di soddisfazione dell'utente.

Figura 4.2: Processo di realizzazione dell'interfaccia grafica[7]

4.1 Interaction Design e Information Architecture

Esistono due diverse tipologie di progettazione che si differenziano sulla base degli scopi per cui si utilizza l'interfaccia:

- **Interaction Design**: progettazione delle funzionalità che permettono di realizzare un compito specifico;
- **Information Architecture**: pratica che permette di organizzare, strutturare ed etichettare i contenuti al fine di renderli facilmente fruibili all'utente finale.

Durante la progettazione dell'applicativo *Archivio Produttori* è stata posta particolare attenzione all'architettura dell'informazione. Nello specifico nella definizione della struttura organizzativa dei contenuti e dei dati informativi, in modo che siano esposti in maniera ben organizzata, che siano facilmente navigabili e ben etichettati. A tal fine è stata individuata l'alberatura completa dell'applicativo, i menù di navigazione e le relative etichette, il sistema di ricerca interno e di confronto tra prodotti e l'organizzazione delle pagine in cui vengono mostrati ingenti quantità di dati.

La struttura informativa ben organizzata facilita l'utente nel trovare in breve tempo tutte le informazioni di cui ha bisogno e crea sin da subito una buona

impressione in termini di affidabilità e credibilità dell'applicativo. Facendo un paragone con uno spaccato di vita quotidiana, avere le informazioni mal organizzate è paragonabile alla sensazione che si avrebbe entrando in un supermercato in cui accanto al reparto della macelleria troviamo quello dei detergenti: certamente questo non lo renderebbe né credibile né apprezzabile.

Quindi, nonostante l'architettura dell'informazione sia un aspetto invisibile di un applicativo web, deve essere curata meticolosamente perché rappresenta lo scheletro che tiene in piedi tutta l'applicazione: quanto più è solido tanto più l'esperienza utente migliora e la prospettiva di successo aumenta.

4.2 Material Design

Per realizzare un prodotto web che possa garantire una buona esperienza utente è necessario mantenere uno stile grafico coerente, piacevole e accattivante per gli utilizzatori. In linea con questi obiettivi, un'importante innovazione dal punto di vista del design è stata introdotta nel 2014 da Google con l'annuncio del *Material Design*.

Nato per uniformare la resa grafica dei prodotti Google, è diventato uno vero e proprio standard diffuso e largamente utilizzato nella progettazione di applicazioni web e mobile. Questo standard si descrive attraverso tre principali tematiche: principi e linee guida, componenti e personalizzazione del tema.

4.2.1 Principi e linee guida

L'idea alla base di questo linguaggio di design è incentrata sulla metafora del *"material"*, un materiale: in pratica ogni elemento grafico viene paragonato ad un materiale reale, ovvero ad un oggetto che sia in grado di adattarsi alle diverse situazioni in cui esso è contestualizzato. Matìas Duarte, il vicepresidente del reparto design di Google, per meglio esplicare il concetto disse: *"Proprio come la carta, il nostro materiale digitale si può espandere o restringere riformandosi in modo intelligente. I materiali hanno superfici fisiche e bordi. Cose come ombre e cuciture forniscono il significato di quello che tocchi."*

Quindi l'interfaccia è paragonabile ad un insieme di fogli di carta con le relative proprietà fisiche: spessore, ombre, movimento. Gli elementi vengono strutturati gerarchicamente uno sull'altro, mantenendo in primo piano i

controlli che interagiscono principalmente con l'utente, come ad esempio i bottoni. La struttura dell'interfaccia deve essere incentrata sull'esperienza utente in modo che sia il più immersiva possibile. La struttura gerarchica dei componenti viene resa tramite l'utilizzo di profondità e tridimensionalità, privilegiando sempre uno stile semplice e pulito.

Figura 4.3: Principi del *Material Design*

Le animazioni devono essere utilizzate per dare feedback all'utente, in modo da mantenere alta la sua attenzione e garantire continuità durante la navigazione. Tutte le interfacce *"material"* devono essere responsive in modo da adattarsi dinamicamente a tutti i dispositivi da cui si accede.

4.2.2 Componenti

Oltre a stabilire le linee guida e i principi a cui ci si dovrebbe attenere per garantire una buona esperienza utente, sono stati introdotti numerosi componenti grafici utilizzabili nello sviluppo web e mobile per la realizzazione dell'interfaccia che includono al loro interno la logica per renderizzare l'attivazione del componente, lo stato di errore, il passaggio del puntatore del mouse, la pressione, la selezione e così via.

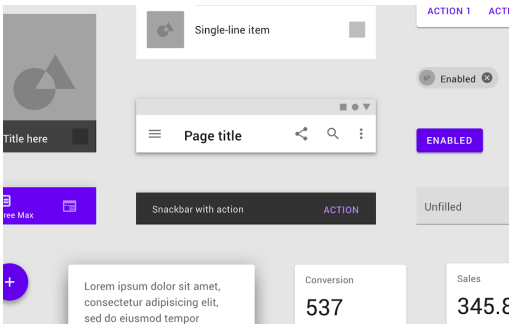

Figura 4.4: Componenti del *Material Design*

4.2.3 Personalizzazione del tema

Material Design dedica un'intera sezione della documentazione alla personalizzazione del tema: si può scegliere la palette di colori, le tipologie di caratteri e le forme dei componenti affinché si possa realizzare una personalizzazione coerente alle esigenze dell'applicativo. Queste scelte, ovviamente, devono sempre essere in linea con la coerenza grafica e stilistica imposte dallo standard.

Per l'utilizzo dei colori, viene associato un nome semantico in base al contesto di applicazione; vengono così definiti il colore primario e quello secondario che molto spesso corrispondono ai colori del marchio dell'applicativo, i quali sono accompagnati da altri come: `error`, `success`, `warning` e `background` che attribuiscono un colore allo stato che rappresentano.

Partendo dalla scelta di un colore primario e di uno secondario, attraverso il *Material Color Tool*, è possibile generare una palette di colori in linea con le linee guida dello standard da poter utilizzare all'interno dell'applicativo. Inoltre, è anche possibile, creare le combinazioni per le modalità dark e light.

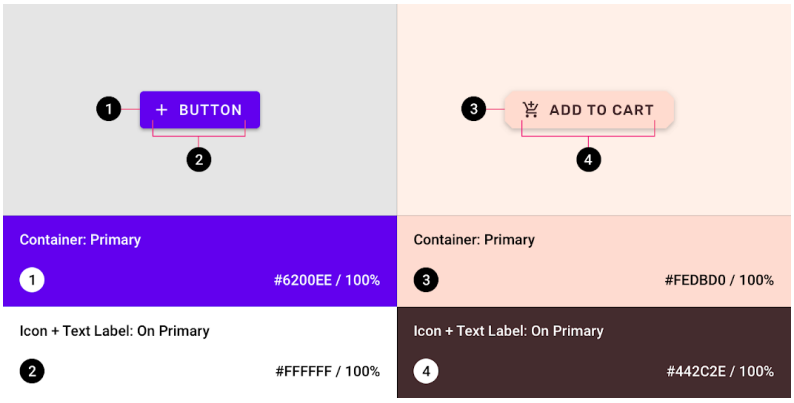

Figura 4.5: Personalizzazione del tema di *Material Design*

4.3 MUI

MUI (**M**aterial to builds **UI**s) è una libreria open-source di componenti React per la realizzazione di interfacce grafiche. È stata sviluppata da Google nel 2014 con il nome *Material-UI* con l'obiettivo di supportare gli sviluppatori React nella creazione di interfacce grafiche secondo gli standard del *Material*

Design. Ad oggi è una delle librerie più utilizzate sia per l'ampio numero di componenti che mette a disposizione sia per la versatilità ad essere utilizzata in contesti web ed in altri più in generale. È supportata da una vasta community che ne favorisce una progressiva crescita. A settembre 2021 è stata rilasciata la nuova versione 5 che ha sancito un cambiamento di nome da Material-UI a MUI. Tra le più interessanti novità introdotte nella nuova versione vi è l'integrazione della libreria *Emotion* che ha permesso di semplificare di parecchio le modalità di attribuzione dello stile grafico ai componenti, prestando particolare attenzione alle prestazioni.

Tutti i componenti che MUI mette a disposizione dello sviluppatore sono stati progettati e ideati mantenendo fede alle linee guida del Material Design e sono riusabili, accessibili e componibili. Un altro punto di forza della libreria è la possibilità di gestire la resa grafica complessiva dei componenti attraverso l'utilizzo dei temi.

I componenti della libreria sono organizzati sotto le seguenti categorie:

- **Layout**: componenti per realizzare la struttura della pagina, tra i quali `Grid`, un componente ottimizzato per la realizzazione responsiva dell'interfaccia in quanto consente di preservare le proporzioni tra i vari elementi che la compongono. In questa sezione rientrano anche `Box` e `Container`;

- **Navigation**: componenti che supportano l'utente durante la navigazione, ad esempio `Breadcrumbs`, `Drawer`, `Menu`;

- **Inputs**: componenti che permettono l'immissione di dati, ad esempio: `Button`, `TextField`, `Switch` e `Slider`;

- **Data Display**: componenti per rappresentare informazioni, ad esempio: `Chip`, `Table`, `Typography`, `Tooltip`, `Datagrid`. Rientrano in questa categoria anche le icone, che possono essere componenti già integrati nella libreria e standardizzati secondo il Material Design oppure `SvgIcon`, per utilizzare icone personalizzate in formato svg, o `Icon` per generiche icone font;

- **Feedback**: componenti che forniscono indicazioni in risposta alle azioni compiute dall'utente, ad esempio, `Alert` e `Snackbar` sono utilizzati per mostrare messaggi oppure `Skeleton` che renderizza la sagoma di un componente mentre si è in attesa del caricamento dei dati necessari o `Progress` che offre un'indicazione temporale della durata di un processo;

- **Surfaces**: componenti che rappresentano delle superfici per altri componenti, tra cui `Paper` e `Accordion`;
- **Utils**: una raccolta di componenti di utilità utilizzabili per migliorare l'interfaccia.

La libreria è in continua espansione e il numero di componenti cresce di versione in versione. Infatti, sotto la sezione *Lab* della libreria sono raccolti tutti i componenti non ancora considerati stabili e che vengono lo stesso resi disponibili per permetterne il testing.

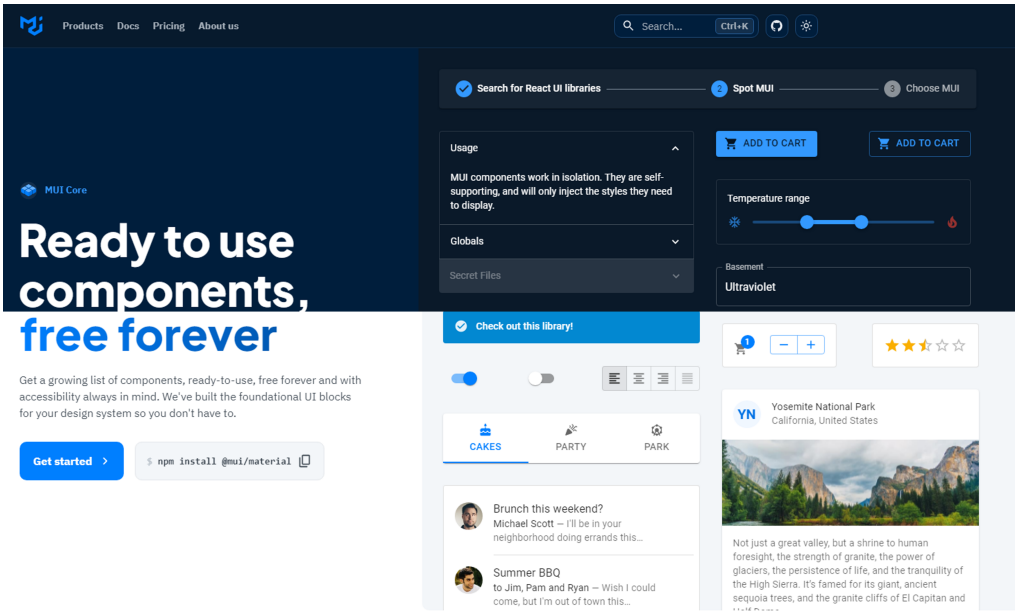

Figura 4.6: MUI: componenti in modalità dark e light

Capitolo 5

Progettazione

In questo capitolo vengono illustrate le fasi di progettazione di *Archivio Produttori* necessarie per definire le specifiche architetturali e funzionali dell'applicativo web. Tale processo è avvenuto nel contesto di un team di lavoro composto da figure professionali con competenze eterogenee, all'insegna della collaborazione e della contaminazione di idee.

5.1 Analisi applicazione desktop

Archivio Produttori, in seguito definito archivio, è un applicazione desktop che espone una raccolta di prodotti e di materiali di edilizia, termotecnica e impiantistica appartenenti a diversi produttori. Esso viene distribuito a corredo di numerosi software di calcolo e di progettazione proprietari di *Edilclima S.r.l.* in modo tale da poter importare dall'archivio un elemento necessario nel progetto avendone a disposizione le caratteristiche tecniche e fisiche. In aggiunta, all'interno dell'archivio sono state integrate particolari funzionalità per svolgere calcoli automatizzati e ricavare parametri utili nella configurazione dell'elemento.

Ad esempio, tra i più importanti software proprietari di *Edilclima S.r.l.* vi è *EC700*, utilizzato per calcolare le prestazioni energetiche degli edifici in conformità alle specifiche tecniche UNI/TS 11300. All'interno di un progetto creato nell'ambito di tale software, quindi, si possono prelevare dall'archivio gli elementi utilizzati nella configurazione dell'edificio e proseguire successivamente con il calcolo della prestazione.

Un'altra funzionalità parallela dell'archivio, popolato direttamente dai produttori aderenti all'iniziativa, è quello di dare visibilità ai prodotti che include, poiché saranno sempre alla mano dei progettisti che lo utilizzano: questo potrebbe causare per il produttore anche un aumento di vendite e di popolarità.

L'interfaccia grafica del software è riportata nella seguente figura:

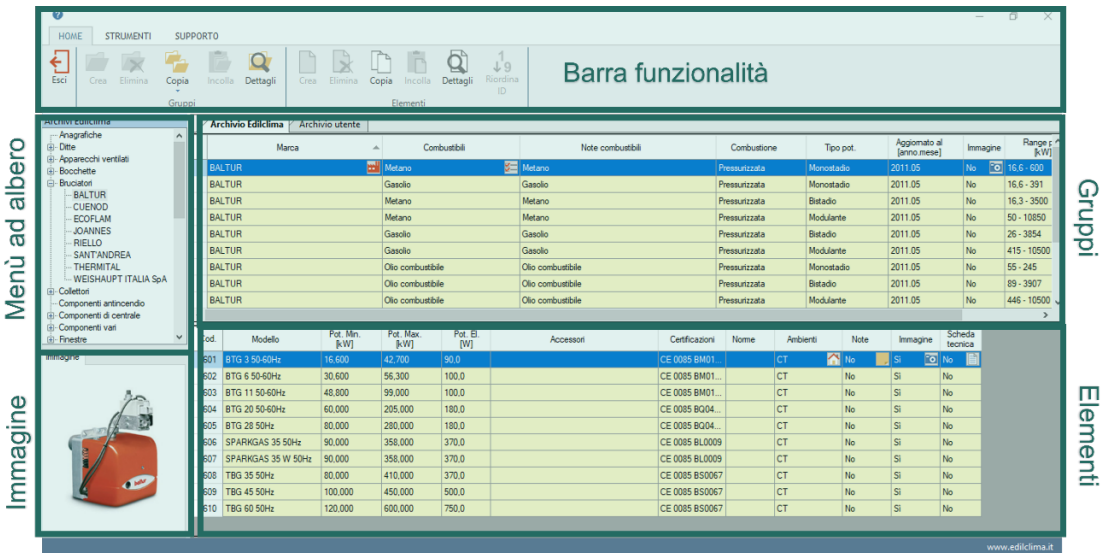

Figura 5.1: Interfaccia desktop di *Archivio Produttori*

Esistono due tipologie di archivio:

Archivio Edilclima: contenente la raccolta di tutti gli elementi resi disponibili da *Edilclima S.r.l.* che è accessibile in sola consultazione;

Archivio Utente: archivio compilato dall'utente che ha la possibilità di importare e modificare al suo interno prodotti presenti in Archivio Edilclima e di inserirne nuovi in base alle proprie esigenze.

In fase iniziale *Archivio Utente* non contiene alcun prodotto, pertanto, di seguito si farà riferimento ad *Archivio Edilclima*. Esso è strutturato su quattro livelli:

- **Ditta**: costituita dall'azienda produttrice;
- **Elemento**: costituito da un singolo componente appartenente ad un gruppo;

- **Gruppo**: una raccolta di più elementi appartenenti alla stessa categoria e alla stessa ditta;
- **Categoria**: una raccolta di più gruppi;

Per navigare l'archivio si utilizza il menù ad albero mostrato nella parte sinistra dell'interfaccia, che contiene la voce Ditte e l'elenco delle categorie di tutti gli elementi (Bocchette, Bruciatori, Collettori, Componenti Antincendio, etc.). Selezionando una categoria viene mostrata nella parte superiore della sezione centrale l'elenco dei gruppi. Allo stesso modo, selezionando un gruppo, nella parte inferiore viene mostrato il relativo elenco di elementi ad esso appartenenti. Per ogni elemento selezionato viene mostrata la relativa immagine nel riquadro "Immagine" posto in basso a sinistra. Per ogni ditta, gruppo o elemento vi è la possibilità di visualizzare una scheda in cui sono riportate le relative specifiche e tramite la quale è possibile accedere ad eventuali funzionalità aggiuntive.

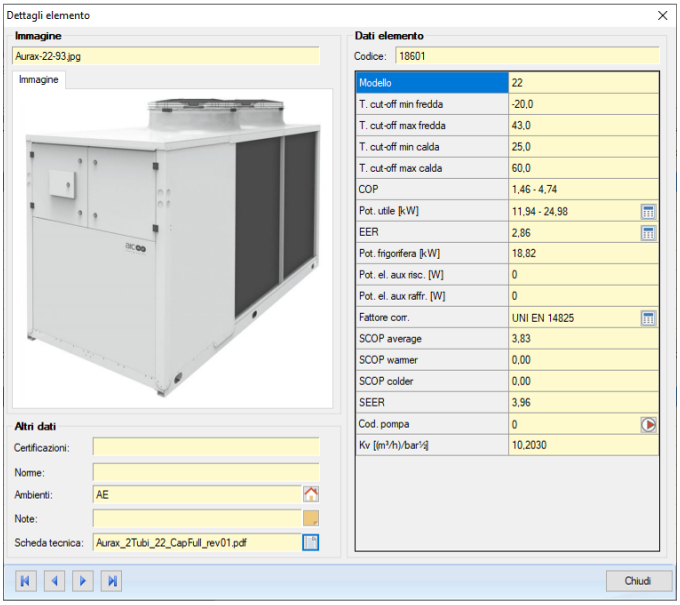

Figura 5.2: Scheda *Dettagli elemento* di *Archivio Produttori*

5.2 Progettazione applicazione web

Dopo aver analizzato l'applicativo esistente, si è proceduto nella progettazione di una soluzione web-based sotto forma di Single Page Application che

consentisse l'interazione e la fruizione del catalogo per gli utenti, la possibilità di aggiornarlo per i produttori e di manutenerlo per *Edilclima S.r.l.*

Le motivazioni per cui è stata scelta una soluzione web-based sono coerenti con tutti i principali vantaggi che la rete offre: migrando l'archivio sul "cloud" si semplificano i processi di aggiornamento, manutenzione e di distribuzione. Quindi non sarà più necessario dover installare fisicamente l'applicativo su ogni macchina poiché esso diventerà accessibile semplicemente tramite un browser ed una connessione internet. In caso di aggiornamenti, questi saranno subito disponibili per tutti senza dover preoccuparsi di notificare ai clienti il nuovo rilascio. Infine, dato che al giorno d'oggi quasi tutti i dispositivi sono dotati di un browser, non vi saranno più problemi legati alla eterogeneità delle piattaforme.

5.2.1 Architettura dell'informazione

In questa fase si è cercato di organizzare e strutturare i contenuti in modo efficace per:

- ridurre la complessità: si è cercato di rendere le informazioni chiare e fruibili;
- supportare la scalabilità: la struttura organizzativa è stata progettata in modo che risultasse facilmente adattabile man mano che l'archivio si estende;
- creare familiarità con il prodotto e rendere piacevole la navigazione sia per utenti con elevata esperienza sia per nuovi utilizzatori.

Partendo dall'analisi di tutte le categorie dell'archivio, si è scelto di mantenere la struttura organizzativa gerarchica dei dati aggiungendo un nuovo livello, denominato **"Area"**, che raccoglie le categorie dalle finalità simili. La nuova struttura è rappresentata in Figura 5.3.

Questa scelta è stata adottata per compattare le categorie, per semplificare la progettazione dell'interfaccia e aumentare la facilità di navigabilità dei contenuti.

Considerando l'alto numero di categorie presenti in archivio e la complessità di casistiche particolari relative ad ognuna di esse, in fase iniziale sono quindi

Figura 5.3: Struttura gerarchica web dei dati di *Archivio Produttori*

state prese in considerazione solo cinque tra le trentadue categorie presenti: Bruciatori, Generatori, Pompe, Pompe di Calore e Sistemi Ibridi.

5.2.2 Analisi del contesto utente e progettazione funzionalità

La progettazione dell'interfaccia è avvenuta seguendo i canoni della *User Centered Design*: quindi, in prima analisi, l'attenzione è stata posta all'esperienza utente. Ogni utente utilizzatore del sistema può assumere diversi ruoli:

- utente occasionale, che consulta tutto il catalogo in sola visualizzazione;
- utente abituale, che consulta tutto il catalogo e può creare e modificare un proprio personale archivio utente;
- utente produttore, che consulta il catalogo contenente i propri prodotti, eventualmente inserisce i nuovi o modifica quelli esistenti; questa operazione è sottoposta ad un workflow di validazione finale da parte di Edilclima;
- utente amministratore, tipicamente personale qualificato dell'azienda, a cui sono garantiti i massimi privilegi: può visualizzare il catalogo completo, inserire, eliminare e modificare prodotti di qualunque produttore, ed inoltre gestire il workflow di inserimento o modifica prodotti ad opera del produttore convalidandone i dati.

È stato preso in analisi anche il livello esperienza utente, constatando che:

- l'utente medio dell'applicativo ha un elevata conoscenza di dominio: essendo tipicamente un esperto del settore ha un alto grado di interpretabilità e di comprensione dei dati;

- dal punto di vista della navigabilità web l'esperienza utente andrà costruita nel tempo durante l'utilizzo del prodotto, cercando di agevolare il passaggio da un pattern d'uso per un applicativo desktop verso uno totalmente differente e innovativo nel contesto web.

Successivamente sono state individuate le seguenti esigenze e le necessità comuni a tutti gli utenti: esplorare il catalogo prodotti in maniera efficiente e confortevole e visualizzare in maniera chiara e dettagliata le specifiche tecniche. Sull'analisi di questi aspetti sono state progettate le funzionalità base di esplorazione, consultazione e modifica dell'archivio. Inoltre, nell'ottica di garantire un'esperienza di navigazione gradevole oltre alle funzionalità base già presenti nell'applicativo esistente, sono state programmate ulteriori funzionalità aggiuntive:

- **Ricerca**: integrazione di un motore di ricerca testuale per trovare elementi o gruppi presenti in archivio;

- **Filtri**: inserimento di un pannello di filtri da poter applicare all'elenco dei gruppi e degli elementi;

- **Confronto**: funzionalità di confronto tra elementi appartenenti alla stessa categoria;

- **Preferiti**: gestione di un elenco di prodotti preferiti personalizzato per ogni utente;

- **Cronologia/Visti di recente**: salvataggio e gestione della cronologia degli ultimi prodotti visualizzati.

5.2.3 Sitemap

L'organizzazione dell'architettura delle informazioni e l'analisi delle funzionalità è stata riepilogata nella sitemap mostrata in figura, dalla quale si individuano le diverse pagine dell'applicativo web, la maggior parte delle quali navigabili solo dopo aver effettuato la procedura di autenticazione.

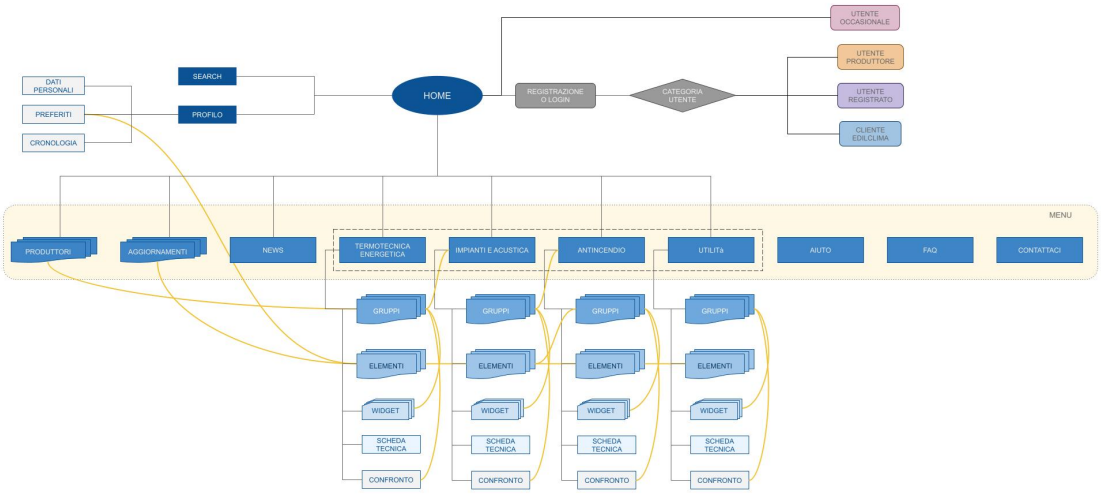

Figura 5.4: Sitemap di *Archivio Produttori*

Nella zona delimitata dal tratteggio sono indicate le quattro aree individuate Termotecnica Energetica, Impianti e Acustica, Antincendio e Utilità, al di sotto delle quali si dirama la struttura gerarchica dei dati. Ad ogni area, con cardinalità molti a molti, appartengono una o più categorie.
Una categoria di fatto è un insieme di gruppi. Un gruppo contiene un insieme di elementi. Ad ogni elemento possono essere associati uno o più widget all'interno dei quali sono rappresentate informazioni aggiuntive ed eventualmente consentono di effettuare calcoli avanzati su determinati campi.
In giallo vengono rappresentate le correlazioni tra le informazioni rappresentate e le funzionalità.

Per ogni elemento del catalogo ci sono molteplici punti di accesso: ad esempio, esso può essere raggiunto tramite la navigazione tradizionale attraversando area, categoria e gruppo oppure dalla pagina dei risultati di ricerca, dalla pagina del confronto o da quella dei prodotti preferiti.

5.2.4 Prototipazione

La fase di progettazione UI/UX dell'applicazione web ha previsto la realizzazione di alcuni mockup, ad opera di Flavia Moschetto, che forniscono indicazioni a scopo illustrativo ed espositivo sulla struttura grafica delle pagine web da implementare. Questi hanno rappresentato il punto di partenza per la successiva fase di implementazione.

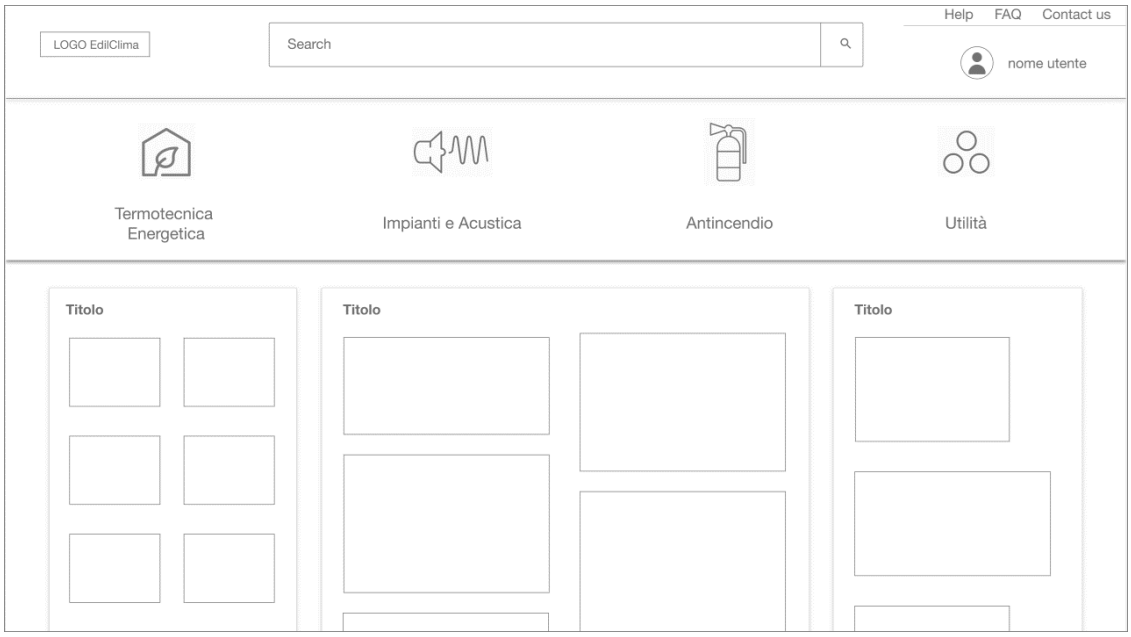

Figura 5.5: Mockup Homepage

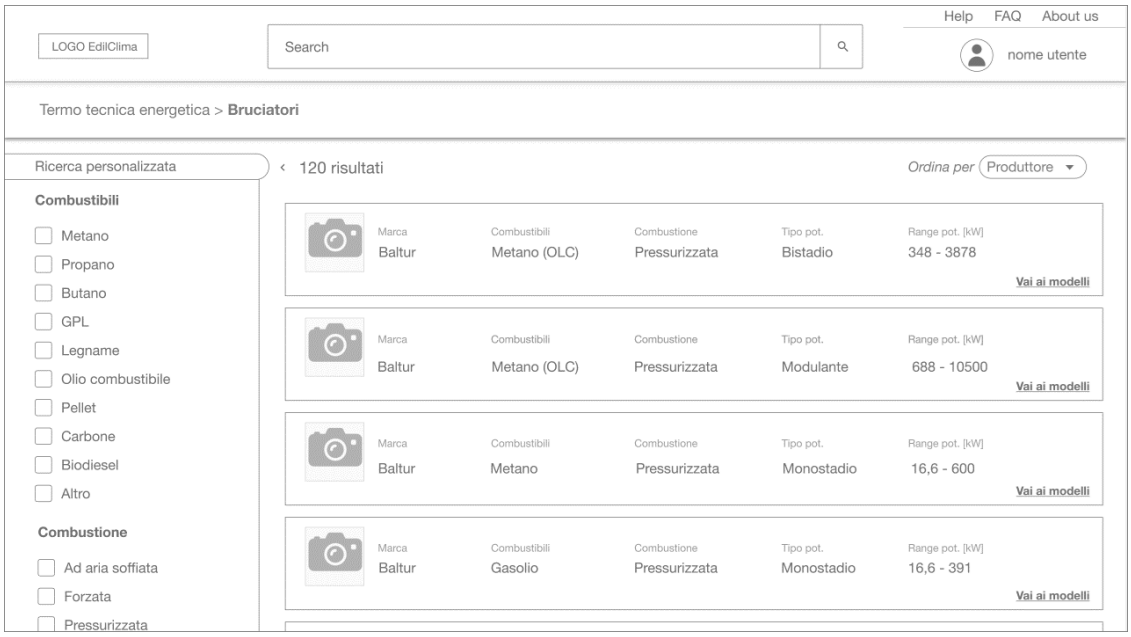

Figura 5.6: Mockup pagina Categoria

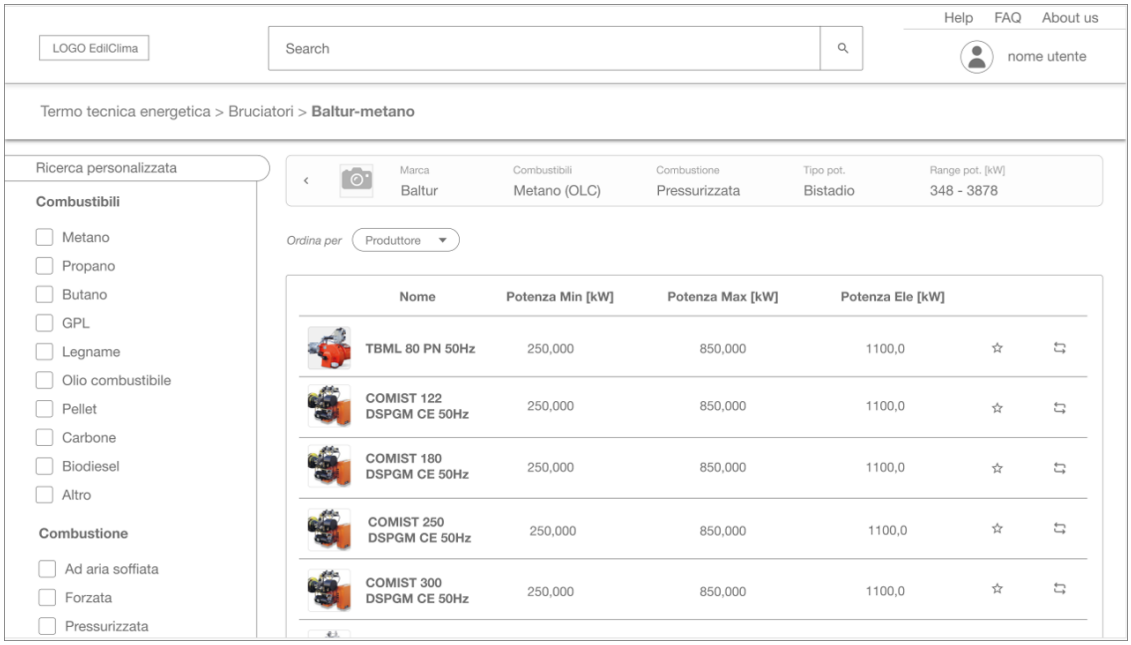

Figura 5.7: Mockup pagina Gruppo

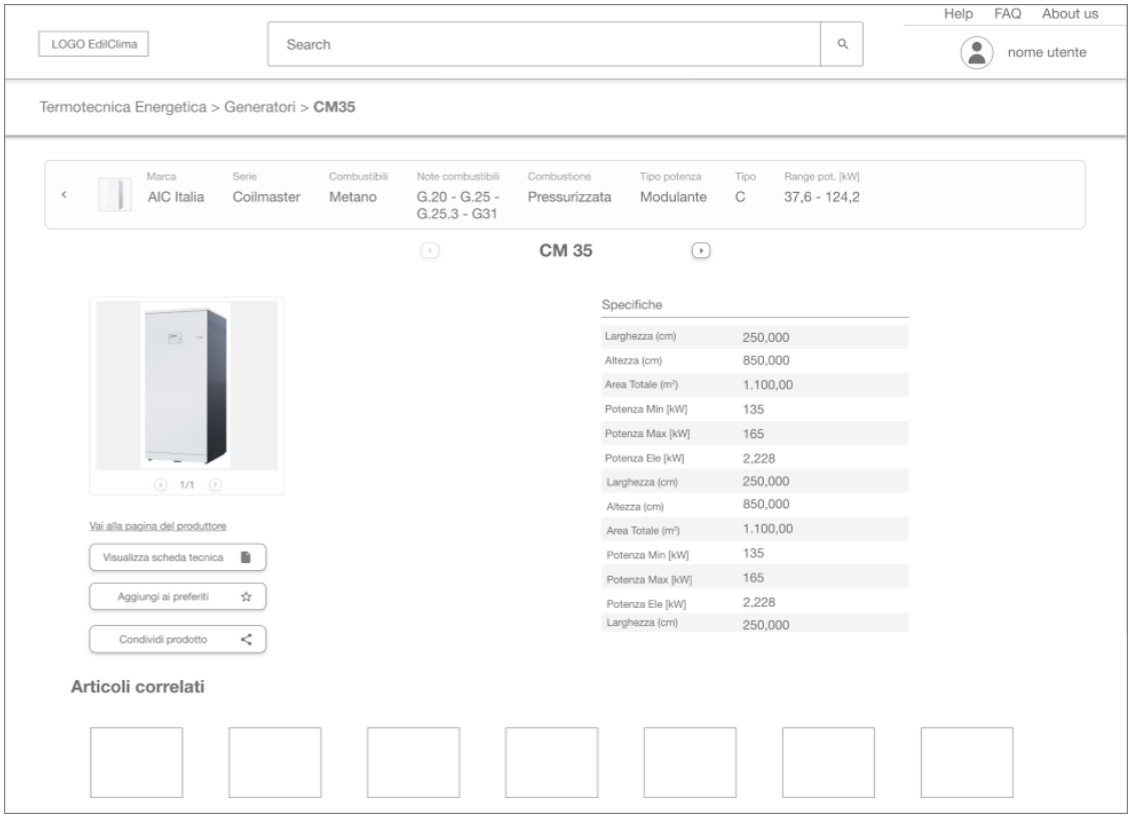

Figura 5.8: Mockup pagina Dettagli elemento

Capitolo 6

Implementazione

Anche la fase implementativa si è svolta nel contesto di un team, all'interno del quale è stata adottata la *metodologia agile*[1], suddividendo il lavoro in milestone e procedendo iterativamente per sprint della durata di circa 2 settimane ognuno. La conclusione di ogni sprint è stata caratterizzata sempre dal testing e dal confronto diretto con il committente del prodotto, in modo da ottenere feedback e riscontri immediati sulle fasi di sviluppo.

In questo capitolo verrà analizzato lo sviluppo front-end della soluzione, avvenuto tramite l'utilizzo della libreria React e con l'ausilio della libreria di componenti grafici MUI.

6.1 Caratteristiche generali

Il progetto è stato implementato avvalendosi delle potenti funzionalità offerte dall'ambiente di sviluppo integrato *JetBrains IntelliJ IDEA*. In fase preliminare sono stati installati *Node.js*, un ambiente di runtime JavaScript lato server e *npm*, un gestore di pacchetti che consente di installare e aggiornare automaticamente nuovi pacchetti nel progetto prelevandoli da un vasto ecosistema di pacchetti di terze parti.

[1]La metodologia agile propone un approccio meno strutturato e focalizzato sull'obiettivo di consegnare al cliente, in tempi brevi e frequentemente, software funzionante e di qualità. Fra le pratiche promosse dai metodi agili ci sono la formazione di team di sviluppo piccoli, lo sviluppo iterativo e incrementale e il coinvolgimento diretto e continuo del cliente nel processo di sviluppo. (fonte: Wikipedia)

Il progetto è stato creato con l'istruzione `npx create-react-app`, un esecutore di pacchetti incluso in npm, che si occupa costruire la catena di costruzione lato front-end di una Single Page Application in React e che configura *Babel* e *Webpack* utilizzati rispettivamente per la transpilazione e l'impacchettamento del codice.
Per il versionamento del codice è stato utilizzato *Git* e il repository è ospitato su *Bitbucket* e su *GitHub*.
L'interazione con il server è avvenuta principalmente utilizzando la libreria *Axios*, attraverso richieste fatte agli endpoint delle API REST, documentate attraverso la libreria open source *Swagger*. La gestione dello stato dell'applicazione si avvale dell'utilizzo dei *Context*. In questo progetto vi sono tre contesti principali:

- `UserContext`: gestisce lo stato dell'utente;
- `GlobalContext`: gestisce lo stato di navigazione dell'archivio;
- `SearchContext`: gestisce lo stato inerente ai risultati di ricerca;

L'applicazione è stata internazionalizzata tramite la gestione di tre differenti linguaggi: italiano, inglese e slovacco. Per la traduzione delle stringhe è stata utilizzata la libreria *i18next*. Ad ogni lingua è stato associato un file JSON, contenente il dizionario delle traduzioni. Tale libreria è stata utilizzata per definire la formattazione dei campi numerici e il numero delle cifre decimali da mostrare che può variare sulla base del significato semantico del valore.

Inoltre a partire dal logo di *Edilclima S.r.l.* è stata estrapolata la `favicon.ico` del sito ed una palette di colori utilizzati nel tema.

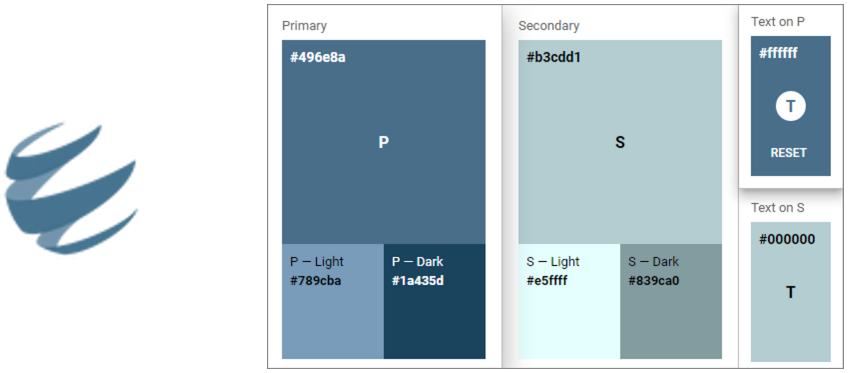

Figura 6.1: `favicon.ico` e colori del tema di *Archivio Produttori*

6.2 Autenticazione

L'accesso all'applicazione avviene tramite la procedura di autenticazione. Attualmente, per il testing e il debug, sono stati creati nel database quattro utenti, uno per ogni ruolo possibile.

L'interfaccia di Login, rappresentata nella seguente figura, permette l'inserimento dei campi username e password.
Utilizzando l'attributo `type="password"` del componente `TextField` la password viene cifrata e nascosta durante l'inserimento. Attraverso il bottone, identificato dall'icona di un occhio, è possibile mostrare la password in chiaro. I dati inseriti nel form di autenticazione sono validati ed in caso di errori vengono mostrati degli opportuni messaggi all'utente.

La realizzazione del form di login e, più in generale, di tutti i form presenti all'interno dell'applicazione ha previsto l'utilizzo della libreria *Formik* la quale, tenendo traccia dei valori, degli errori e dei campi visitati, permette la gestione dello stato e dei campi del form in maniera semplice e immediata.

Per la validazione dei singoli campi dei form ci si è avvalsi della libreria *YUP*, un generatore di schemi JavaScript per l'analisi e la convalida dei valori. Gli schemi YUP sono altamente espressivi e consentono di modellare complesse convalide.

Lo standard di autorizzazione utilizzato è *OAuth* che permette di effettuare chiamate in modo sicuro e autorizzato verso le API, attraverso l'utilizzo di un token di accesso.

È stato utilizzato il *JSON Web Token (JWT)*, definito nello standard aperto RFC-7519, il cui formato è una stringa codificata in base-64 composta da tre parti separate tra loro da un punto: la prima parte è un header nel quale viene specificato il tipo di token e l'algoritmo di firma utilizzato; la seconda parte un payload, contenente dati arbitrari in formato JSON; infine, la terza parte contiene la firma digitale di header e payload. Poiché i dati sono firmati non è possibile modificarli, altrimenti si invaliderebbe la firma.

La procedura di autenticazione avviene attraverso una chiamata all'authorization server nella quale, sulla base di una corretta validazione del form, vengono inviati come parametri username e password, che se validi consentono al client di ricevere il token di autenticazione JWT.

Il JWT token viene memorizzato all'interno di un cookie che verrà poi utilizzato nell'header di tutte le richieste successive per accedere alle risorse

protette dimostrando che si dispone dell'autorizzazione necessaria. Il cookie ha una validità temporale di 18 ore, trascorse le quali è necessario rieseguire il login.

Al completamento della procedura di autenticazione nell'applicazione viene inizializzato il contesto dell'utente che contiene tutte le informazioni ad esso associate tra cui nome, email, ruolo utente e lingua utilizzata.

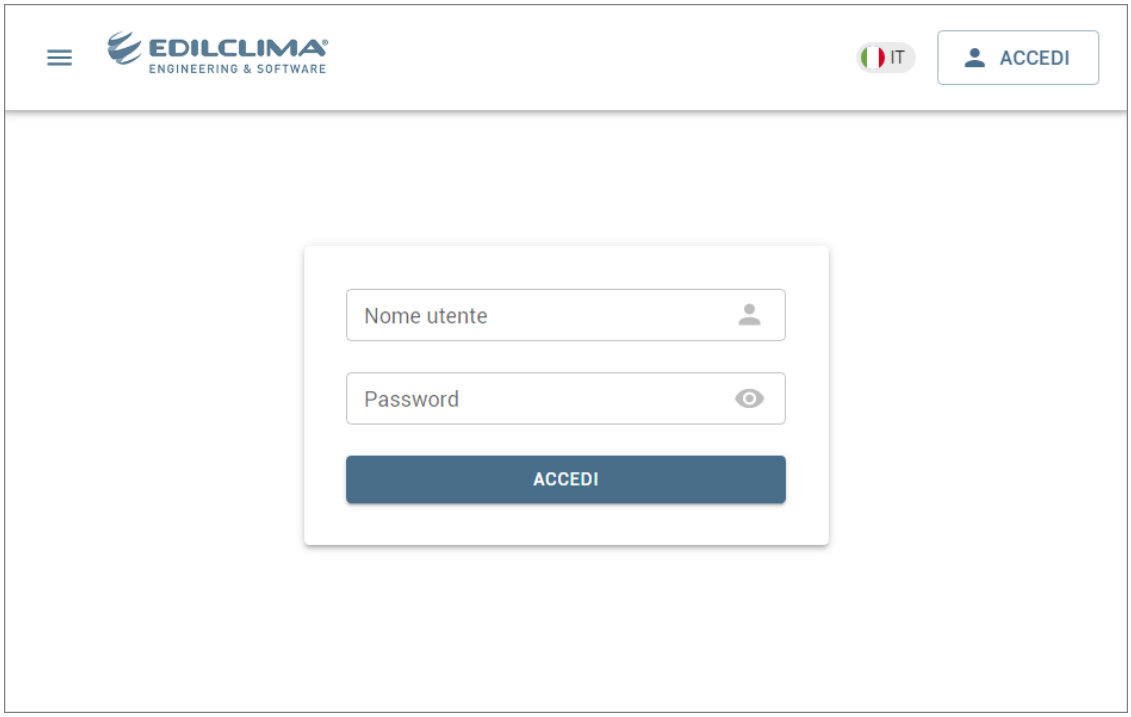

Figura 6.2: Pagina Login di *Archivio Produttori*

6.2.1 Area utente

L'utente autenticato visualizza nella barra di navigazione, in alto, il proprio avatar, che rappresenta le iniziali del nome e del cognome. Cliccando sull'avatar si apre un menù contestuale che permette di visualizzare l'area utente o di effettuare il logout.

L'area utente, mostrata in figura, ha un'interfaccia molto semplice: mostra un messaggio di benvenuto e due `Tab`: una contiene l'elenco degli ultimi cento prodotti visualizzati di recente (dei quali è stata visitata la pagina del dettaglio dell'elemento) mentre l'altra contiene l'elenco dei prodotti preferiti.

Per ognuno dei due elenchi è possibile gestire l'eliminazione di un singolo elemento dalla raccolta o l'eliminazione massiva di tutti gli elementi tramite l'apposito bottone `Elimina Tutti` presente nella toolbar della tabella.

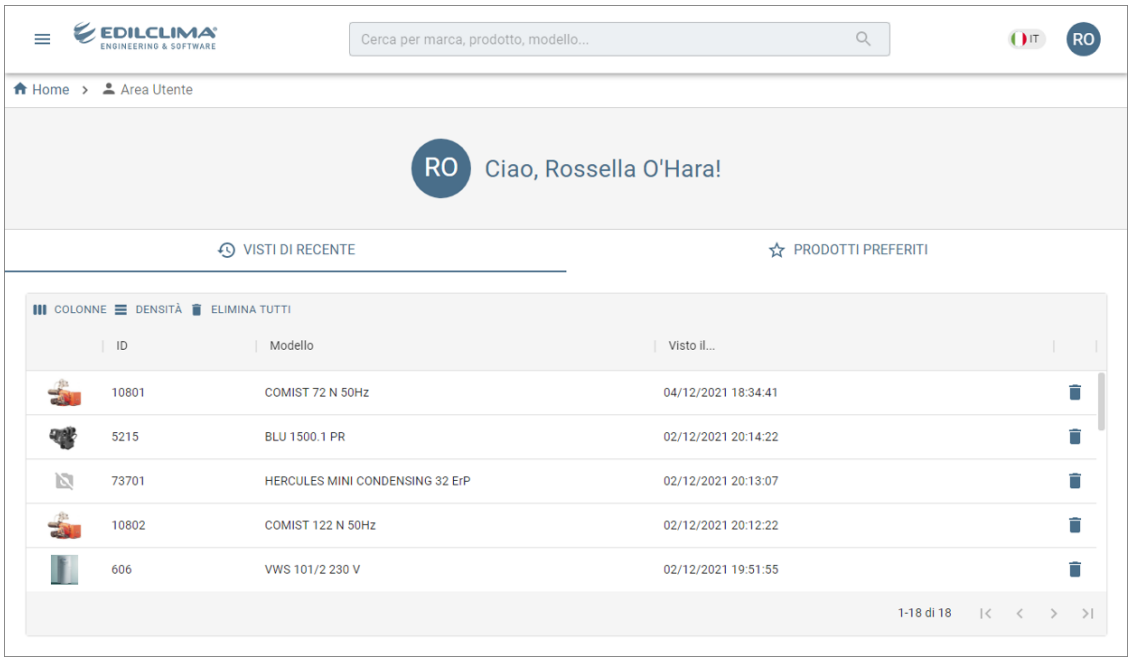

Figura 6.3: Pagina Area utente di *Archivio Produttori*

6.3 Navigazione Archivio

In questo paragrafo verrà descritta l'esperienza di navigazione realizzata attraverso le pagine Homepage, Area, Categoria, Gruppo, Elemento.

Una buona navigazione deve mettere sempre gli utenti in condizione di non sentirsi smarriti e di avere indicazioni chiare sui percorsi da raggiungere per visualizzare i contenuti desiderati. Per perseguire questo obiettivo, sono stati realizzati componenti e funzionalità che supportano l'utente durante tutta la navigazione.

AppBar

La barra dell'applicazione è sempre fissa in alto in tutte le pagine dell'applicativo. Essa ha due funzioni fondamentali: permette di eseguire azioni che

possono risultare utili in ogni momento della navigazione e funge da guida per l'utente a cui conferisce la certezza di essere nello stesso sito.

Tale barra contiene:

- un pulsante per l'apertura del `Drawer`;
- il logo di *Edilclima S.r.l.*, il quale con un clic riporta alla Homepage;
- la barra di ricerca;
- il menù della lingua;
- l'avatar utente che permette di accedere al menù utente.

Breadcrumbs

Il `Breadcrumbs` (dall'inglese "briciole di pane") ha una duplice funzione: comunica all'utente il percorso effettuato per raggiungere una determinata pagina e permette la navigazione a ritroso fra i livelli attraversati. Esso è posizionato nella parte alta della pagina, subito sotto il componente `AppBar`. Esso contiene una coppia icona-etichetta per ogni livello di navigazione. Ogni coppia è separata dalla successiva da uno speciale carattere separatore. Tutte le etichette sono cliccabili in modo da permettere all'utente di risalire nella gerarchia dei contenuti consentendogli di saltare i livelli intermedi.

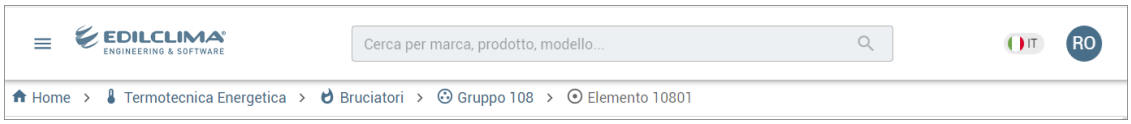

Figura 6.4: Application Bar e Breadcrumbs di *Archivio Produttori*

Drawer

Un altro componente a supporto della navigazione è il `Drawer`, la cui apertura è controllata dall'apposito bottone caratterizzato dall'icona con le tre linee presente in `AppBar`. Esso permette l'accesso alle pagine "Area" del catalogo che rappresentano il livello gerarchico più alto nell'alberatura dei contenuti, e la possibilità di visualizzare altre pagine incluse nel sito, tra cui News, Produttori e FAQ.

Nella parte inferiore del `Drawer` è indicato il numero di versione del front-end, aggiornato ad ogni nuovo rilascio ed utile in fase di sviluppo per avere sempre un riferimento della versione in esecuzione.

Il `Drawer` è un utile supporto quando si accede all'applicazione da PC o tablet, mentre è di fondamentale importanza quando si utilizza lo smartphone perché, a causa delle dimensioni ridotte del display, è l'unico strumento di navigazione che resta visibile all'utente e quindi utilizzabile.

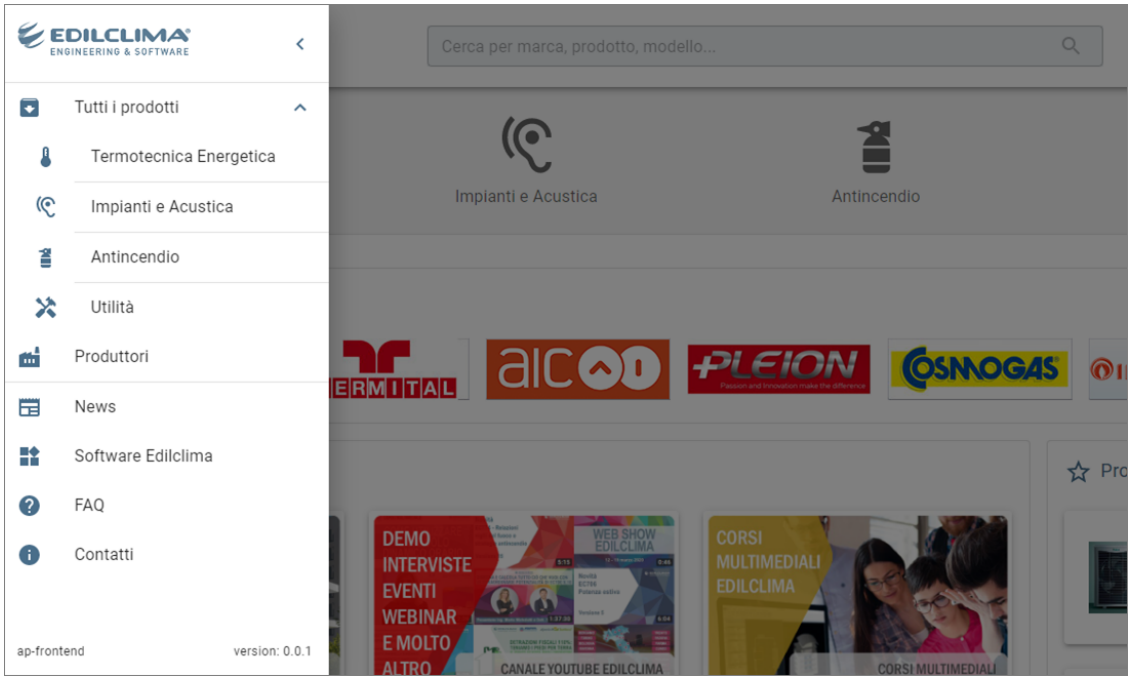

Figura 6.5: Drawer di *Archivio Produttori*

6.3.1 Routing

Nel capitolo 3 è stata affrontata l'importanza dell'architettura di routing nel contesto di una Single Page Application. Infatti, poiché la pagina HTML visualizzata è sempre la stessa, diventa fondamentale introdurre un meccanismo che simuli la navigazione multi-pagina, per dare sempre l'idea che l'URL cambi per ogni pagina, evitando che si provi un senso di smarrimento durante la navigazione. Per realizzare questo artefatto è stata utilizzata la libreria *React Router* che permette, utilizzando l'API della cronologia offerta dai browser, di aggiornare dinamicamente l'URL e di realizzare attraverso codice JavaScript il cambiamento di pagina.

Sulla base della struttura gerarchica delle pagine individuata in fase di progettazione, il meccanismo di routing è stato implementato nel seguente modo:

```
<BrowserRouter>
  <Switch>
    <Route path="/login" exact component={Login}/>
    <Route path="/news" exact component={News}/>
    <Route path="/software" exact component={Software}/>
    <Route path="/faq" exact component={FAQ}/>
    <Route path="/contacts" exact component={Contacts}/>
    <PrivateRoute path="/" component={HomePage}/>
  </Switch>
</BrowserRouter>
```

Per gestire le pagine che si possono visualizzare solo se l'utente è autenticato è stato appositamente implementato un componente `PrivateRoute` che, prima di renderizzare la route corretta, verifica se questa condizione è rispettata, altrimenti esegue un redirect alla pagina di login:

```
const PrivateRoute = ({component: Component, ...rest}) => {
  const [state] = useContext(UserContext);
  const {loginStatus} = state;
  return (
    <Route
      {...rest}
      render={props => loginStatus === LOGIN_STATUS_LOGGED_IN ?
        <Component {...props} /> :
        <Redirect
          to={{
            pathname: "/login",
            state: {referrer: props.location}
          }}
        />
      }
    />
  );
}
```

Una lista esaustiva di tutte le route raggiungibili è riportata nella seguente tabella con l'indicazione se per accedere ad ognuna di esse è necessario che l'utente sia autenticato.

Pagina	URL	Autenticato
Homepage	`/`	Sì
Area utente	`/user`	Sì
Produttori	`/companies`	Sì
Produttore	`/companies/:company`	Sì
Area	`/:area`	Sì
Categoria	`/:area/:category`	Sì
Gruppo	`/:area/:category/:group`	Sì
Elemento	`/:area/:category/:group/:element`	Sì
Confronto	`/:area/:category/compare`	Sì
Ricerca	`/search`	Sì
Login	`/login`	No
News	`/news`	No
Software	`/software`	No
FAQ	`/faq`	No
Contacts	`/contacts`	No

Tabella 6.1: Corrispondenze Pagina - URL

Un'altra caratteristica realizzata a supporto del routing per simulare la navigazione multi-pagina è stata l'associazione di un titolo di pagina per ogni diverso URL, in modo da dare all'utente una corretta indicazione della pagina in cui si trova.
L'attribuzione dei nomi di pagina è avvenuta attraverso l'utilizzo della libreria *React Helmet*. Tecnicamente, all'interno del layout di ogni pagina è stato iniettato un componente `Helmet` incaricato di modificare il document title HTML e di attribuire così alla pagina il giusto titolo. Questa funzionalità è molto utile ai motori di ricerca nella fase di indicizzazione tramite SEO.

6.3.2 Homepage

La Homepage è stata implementata utilizzando il componente `Grid`; questo fa sì che il layout della pagina possa adattarsi con facilità a risoluzioni e a dispositivi differenti. Essa è composta dalle seguenti sezioni:

- **Navigation Bar**: una barra nella quale sono visibili le quattro aree principali: Termotecnica Energetica, Impianti e Acustica, Antincendio

e Utilità; al passaggio del mouse su ognuna di esse la sezione rispettiva della barra viene evidenziata e in una zona sottostante vengono mostrate le categorie appartenenti all'area;

- **Produttori**: un'area all'interno nella quale vengono mostrati i loghi dei produttori che hanno aderito all'iniziativa di *Edilclima S.r.l.* inserendo i loro prodotti in archivio e ai quali si vuole dare visibilità;
- **News**: uno spazio dedicato alle ultime notizie pubblicate dall'azienda;
- **Software Edilclima**: area dedicata alla promozione dei software di calcolo e di progettazione proprietari dell'azienda;
- **Prodotti Preferiti**: se l'utente ha selezionato alcuni prodotti preferiti, vengono mostrati in questa sezione;
- **Visti di recente**: sono mostrati in questa sezione gli ultimi quattro elementi dell'archivio che sono stati visualizzati.

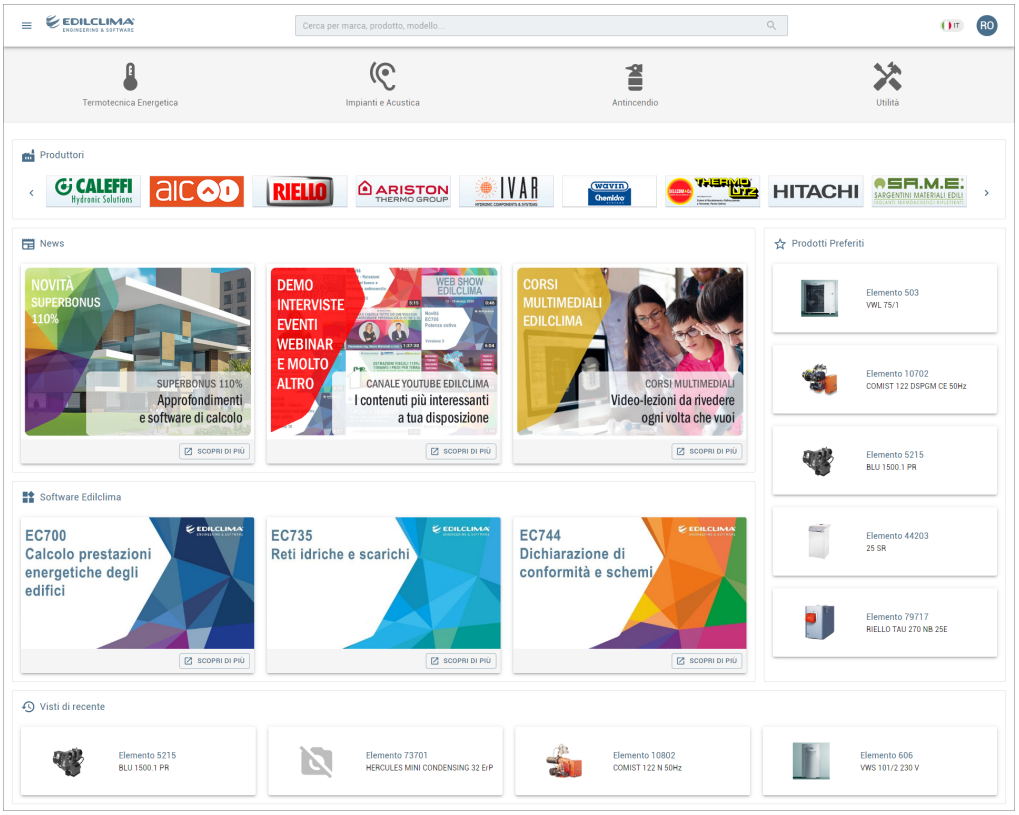

Figura 6.6: Homepage di *Archivio Produttori*

I prodotti preferiti e i visti di recente vengono visualizzati tramite un componente `Card` che riassume le informazioni essenziali che caratterizzano il singolo prodotto: immagine, nome, modello o serie. Cliccando sull'elemento è possibile raggiungere direttamente la pagina relativa al dettaglio delle sue specifiche tecniche.

6.3.3 Produttori

La pagina produttori contiene le card che mostrano al loro interno le informazioni rilevanti per le aziende convenzionate. I dati mostrati sono restituiti da una chiamata all'endpoint `/archives/brands`. Ad ogni produttore corrisponde un componente `Card` in cui è riportato il nome dell'azienda, il codice ad essa associato e il logo o un'immagine. Tramite un bottone collocato nella barra delle azioni della card, è possibile aprire in una nuova scheda del browser la pagina dedicata al produttore.

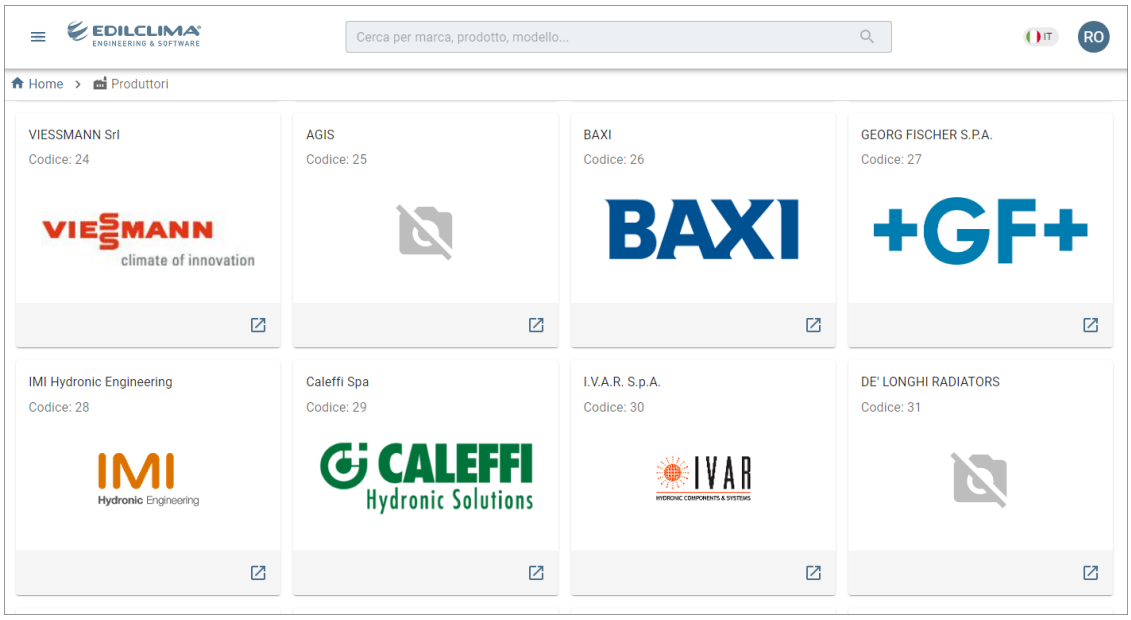

Figura 6.7: Pagina Produttori di *Archivio Produttori*

La pagina del dettaglio produttore è organizzata nel seguente modo: nella parte superiore è indicato il nome dell'azienda. Le frecce consentono all'utente di spostarsi rispettivamente alla scheda del produttore precedente o al successivo; nella parte sottostante vengono mostrate: l'immagine o il logo sulla sinistra, mentre sulla destra la tabella con i dati dell'azienda.

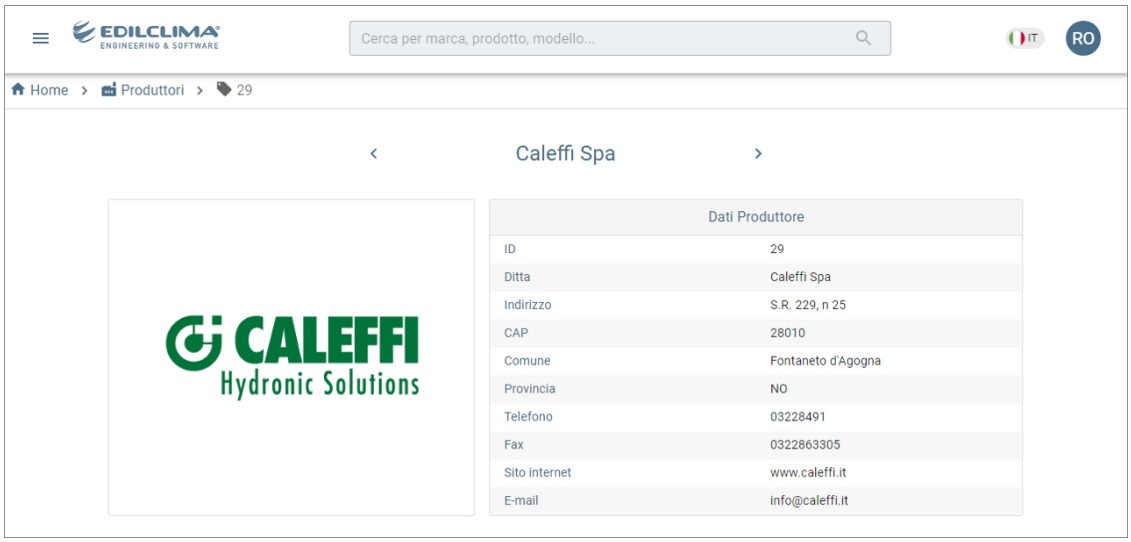

Figura 6.8: Pagina Dettaglio produttore di *Archivio Produttori*

6.3.4 Area

La pagina Area è raggiungibile cliccando su una delle quattro apposite zone del componente `NavigationBar` oppure tramite la corrispondente voce del `Drawer`.

Figura 6.9: Pagina Area di *Archivio Produttori*

Nella parte superiore della pagina viene indicato il nome dell'area e la relativa icona che la rappresenta.

Il contenuto della pagina è composto da un insieme di card, ognuna delle quali rappresenta una categoria dell'area, identificata anch'essa da un nome e da un'opportuna icona. Cliccando sulla card viene mostrata la pagina della categoria selezionata.
La corrispondenza area-categoria è caratterizzata da una relazione molti a molti: ad ogni area appartengono più categorie; ogni categoria può appartenere a più aree.

Nella prima fase di implementazione sono state considerate cinque delle trentadue categorie. Le relazioni con le rispettive aree sono riassunte nella seguente tabella:

Area	**Categoria**
Termotecnica Energetica	Bruciatori Generatori Pompe Pompe di calore Sistemi Ibridi
Impianti e Acustica	Bruciatori Generatori Pompe Pompe di calore
Antincendio	Pompe
Utilità	Bruciatori Generatori Pompe

Tabella 6.2: Corrispondenze Area - Categoria

6.3.5 Categoria

In base all'organizzazione gerarchica dell'archivio la categoria è composta da un insieme di gruppi. Ogni gruppo a sua volta è il "contenitore" di un insieme di elementi. All'interno della pagina Categoria vengono rappresentate tutte le informazioni dei gruppi di una data categoria.

Tali informazioni vengono visualizzate attraverso un particolare componente della libreria MUI: il `DataGrid`, realizzato per visualizzare dati in forma tabellare tramite l'utilizzo di griglie. Esso sfrutta la potenza di React per fornire una buona esperienza utente mentre si visualizzano i dati ed offre una serie di funzionalità integrate che ne facilitano l'esplorazione.

Il componente `DataGrid` è composto da tre parti:

- **Header**: contiene a sua volta una toolbar di azioni che agiscono sul DataGrid modificandone lo stato: il bottone `Colonne` mostra un menù contestuale all'interno del quale è possibile selezionare le colonne che si vogliono visualizzare; il bottone `Densità` permette di scegliere la spaziatura tra una riga e l'altra per agevolare la lettura dei dati;

- **Content**: contiene i dati in forma tabellare dei gruppi appartenenti alla categoria; più dettagliatamente, gli header delle colonne rappresentano gli attributi e in ogni riga vengono mostrati i valori corrispondenti ad un determinato gruppo. Tra gli attributi rilevanti vi sono un ID per identificare univocamente il gruppo, l'immagine e la scheda tecnica che rimanda al collegamento del datasheet del produttore, se previsto. Inoltre, per alcuni campi particolari potrebbe essere previsto un bottone che consente di aprire una maschera per visualizzare dati aggiuntivi, grafici sul dato o eseguire calcoli avanzati.

- **Footer**: contiene un componente per navigare le pagine della tabella e visualizzare i gruppi successivi.

Cliccando sugli header delle colonne, è possibile ordinare tutti gli elementi in base al criterio definito internamente dal `DataGrid` sulla base del tipo di dato contenuto.

Il layout della pagina è stato realizzato con l'ausilio di un ulteriore componente che prende il nome di `Autosizer`. Tale componente, incluso nel progetto tramite una libreria di terze parti, si occupa di regolare automaticamente la larghezza e l'altezza del componente figlio in modo che occupi tutta la pagina. In questo modo la tabella occupa tutto lo spazio disponibile. Il clic su una riga del `DataGrid` rimanda alla pagina specifica del gruppo, che conterrà la lista di tutti gli elementi ad esso appartenenti.

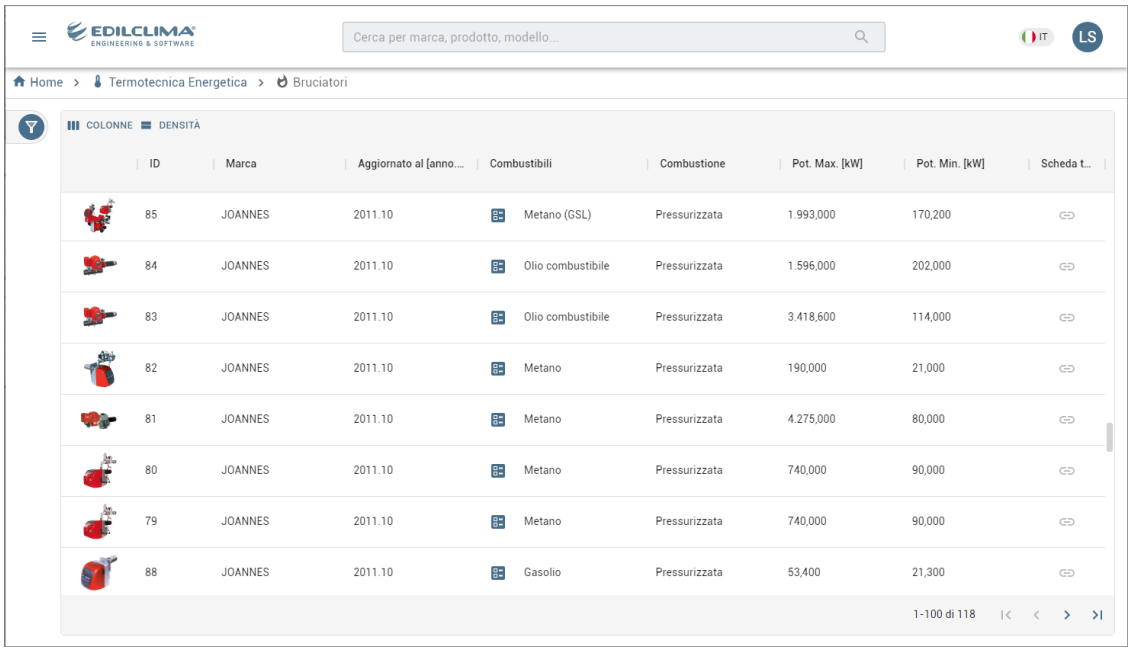

Figura 6.10: Pagina Categoria di *Archivio Produttori*

Filtri

Per facilitare l'esplorazione dei gruppi all'interno del `DataGrid` sono stati implementati accurati meccanismi di filtraggio che possono essere applicati tramite il pannello dei filtri collocato sulla sinistra della pagina ed espandibile tramite il clic sull'icona identificata da un imbuto.

Il pannello viene popolato dinamicamente sulla base di alcuni attributi più significativi del gruppo e ai relativi valori possibili. Per ognuno degli attributi significativi individuati è stato realizzato un `Accordion` che contiene all'interno i valori possibili da filtrare.

Nel caso il tipo di dato corrispondente all'attributo è testuale i valori su cui filtrare sono selezionabili tramite opportune checkbox; mentre, se il tipo è numerico c'è la possibilità di usare il componente `Slider`, che permette di selezionare i valori numerici in un range continuo di valori, oppure un componente `TextField` nel quale inserire manualmente il valore.

Per ogni filtro selezionato all'interno del pannello, sulla parte superiore della tabella compaiono: una label che indica l'attributo per cui si sta filtrando e un elenco di `Chip` interattive che mostrano i filtri selezionati per quel particolare attributo. Cliccando sulla `x` all'interno della `Chip` è possibile eliminare il

relativo filtro. Per resettare contemporaneamente tutti i filtri è stato inserito un apposito bottone `Reset Filtri` mostrato in alto a destra rispetto al `DataGrid`.

Tutti i filtri selezionati vengono anche utilizzati per parametrizzare l'URL tramite la libreria *query-string* in modo che, se si condivide il link o si ricarica la pagina, si riesce ad ottenere una riproduzione fedele della configurazione dei parametri di filtraggio.

Nella seguente figura viene mostrato un caso d'uso in cui sono stati applicati i filtri per visualizzare i bruciatori il cui combustibile è "olio combustibile", con tipologia di combustione "pressurizzata" e con un range di potenza compreso tra 10 e 600 kW.

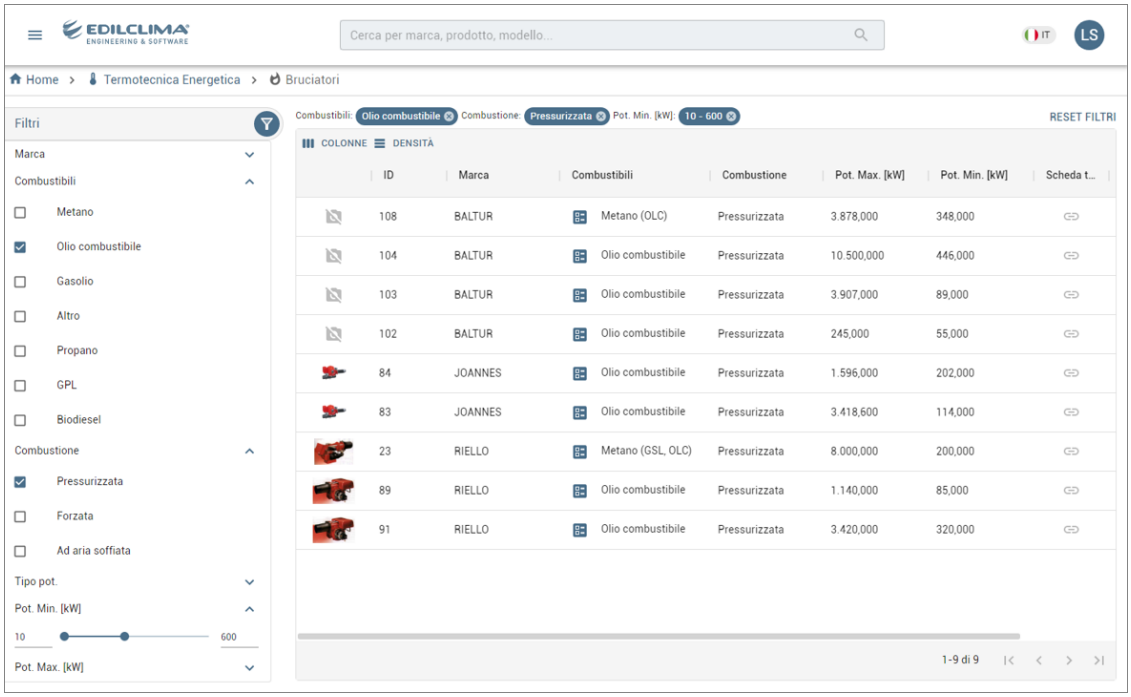

Figura 6.11: Applicazione dei filtri sui gruppi

6.3.6 Gruppo

La pagina Gruppo viene utilizzata per rappresentare al suo interno i dati degli elementi contenuti nel gruppo. Il layout della pagina è molto simile a quello della pagina Categoria: è stato utilizzato allo stesso modo il componente `DataGrid` per visualizzare i dati relativi agli elementi, il pannello dei filtri

laterale (che in questo caso viene popolato dinamicamente con gli attributi e i valori degli elementi contenuti nella `DataGrid`) ed è stato utilizzato anche in questo caso il componente `Autosizer`.

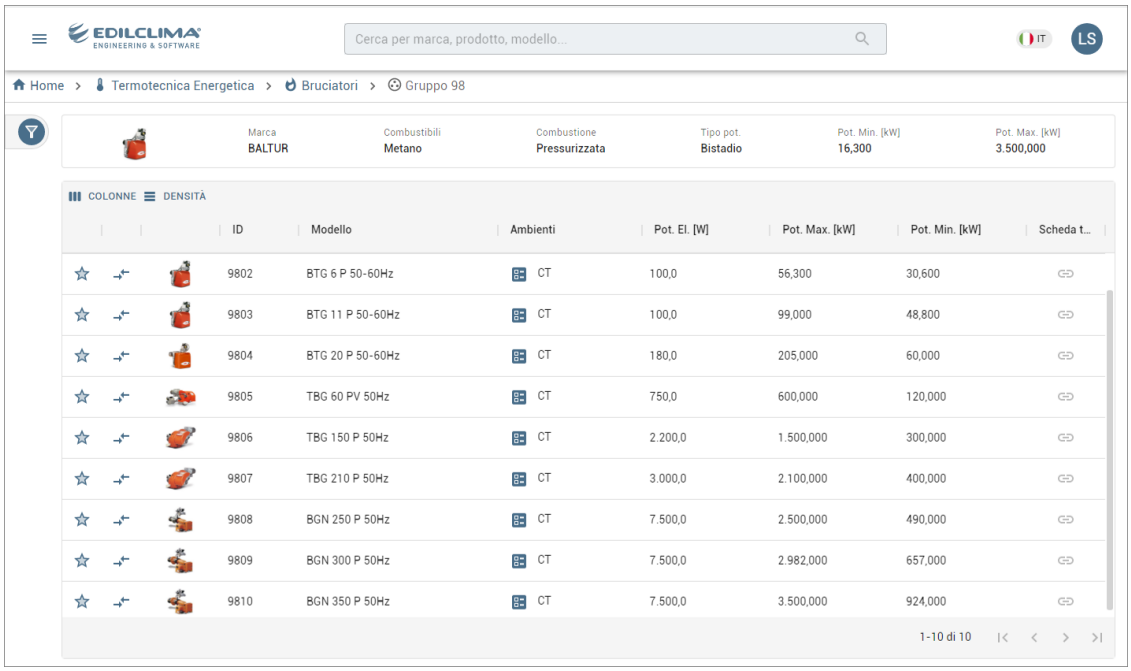

Figura 6.12: Pagina Gruppo di *Archivio Produttori*

La descrizione di tale pagina verrà quindi esposta nel seguito per differenza rispetto alla pagina Categoria.

Nell'area appena superiore alla tabella viene mostrato un componente denominato `GroupInfo` che racchiude al suo interno gli attributi e i relativi valori caratterizzanti per il gruppo, tra cui sicuramente troveremo l'immagine e la marca. Questo componente può essere molto utile per avere sempre un riferimento fisso che consenta all'utente di identificare il gruppo che in quel momento si sta visualizzando.

In questa pagina, all'interno del componente `DataGrid`, oltre ai dati specifici di ogni elemento, sono stati aggiunti due bottoni in corrispondenza delle prime due colonne: `Aggiungi ai preferiti` e `Aggiungi al confronto` che consentono rispettivamente di aggiungere l'elemento alla raccolta dei preferiti o di selezionarlo per un successivo confronto con altri elementi. Queste due azioni, essendo specificatamente possibili solo sugli elementi e non sul gruppo, vengono mostrate solo in questo componente e non compaiono invece in quello della pagina Categoria.

Anche in questa pagina, tramite appositi bottoni inseriti in corrispondenza dei valori che lo prevedono, possono essere richiamate delle maschere dedicate a mostrare dati aggiuntivi, talvolta contenenti grafici o funzioni per svolgere calcoli tecnici su alcuni parametri degli elementi.

La progettazione delle pagine Categoria e Gruppo ha richiesto un importante coinvolgimento del committente in quanto l'applicativo web, diversamente da quello desktop, per esigenze di spazio e di visualizzazione posiziona in due pagine differenti i gruppi e gli elementi.

6.3.7 Elemento

La pagina dell'elemento è il livello più in basso della gerarchia di navigazione. Al suo interno vengono mostrate le specifiche tecniche di un prodotto dell'archivio e sono richiamabili tutte le funzionalità correlate ad un generico elemento. Anche in questo caso il layout è stato costruito tramite i componenti `Grid`. Nella parte alta della pagina è stato inserito il componente `GroupInfo`, che riassume le informazioni più importanti del gruppo a cui l'elemento appartiene.

Il contenuto centrale della pagina include diversi componenti:

- `ElementController`: mostra il nome del componente racchiuso tra due frecce alle due estremità, tramite le quali è possibile esplorare gli elementi successivi o precedenti che appartengono allo stesso gruppo dell'elemento corrente;
- `ElementCard`: mostra l'immagine del prodotto e una barra delle azioni richiamabili sull'elemento;
- `ElementDetail`: mostra la tabella delle specifiche tecniche con la corrispondenza attributo-valore. Dalla tabella delle specifiche tecniche, in corrispondenza degli attributi che lo prevedono, è anche possibile visualizzare apposite finestre di dialogo che consentono di eseguire calcoli avanzati sul relativo valore o di visualizzare informazioni aggiuntive.

Tramite i bottoni presenti nell'apposita barra è possibile eseguire le seguenti azioni:

- apertura, in una nuova scheda del browser, della scheda tecnica dell'elemento fornita dal produttore;

- aggiunta dell'elemento alla raccolta degli elementi preferiti;
- copia dell'URL del prodotto nella clipboard degli appunti per permettere di condividere rapidamente il link del prodotto;
- aggiunta dell'elemento alla barra del confronto (si veda il paragrafo 6.5).

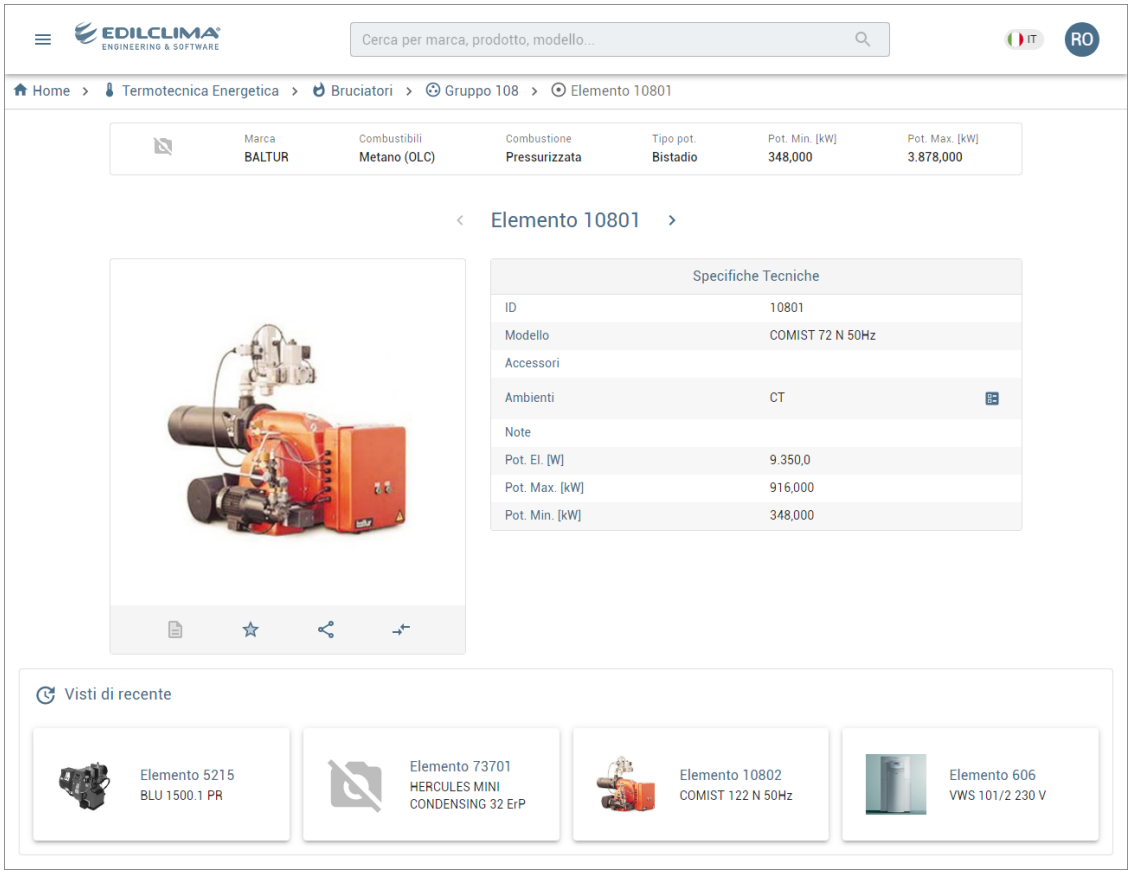

Figura 6.13: Pagina Elemento di *Archivio Produttori*

Ogni volta che si visualizza la pagina del dettaglio elemento, viene eseguita una chiamata automatica di tipo `POST` all'endpoint `/users/me/history` attraverso la quale l'elemento viene aggiunto alla cronologia degli ultimi elementi visualizzati.

In fondo alla pagina c'è una sezione dedicata agli elementi visualizzati di recente nella quale sono mostrati gli ultimi quattro elementi della cronologia visualizzati precedentemente a quello corrente. Ogni elemento in questa sezione è visualizzato attraverso un componente `Card` cliccabile che rimanda all'elemento e di cui ne visualizza il modello o la serie, l'immagine e il nome.

6.4 Ricerca

Una parte molto importante ed interessante dell'applicazione web è sicuramente la ricerca. L'utente ha la possibilità di cercare informazioni in archivio digitando una o più parole chiave all'interno del campo di testo posto nella barra dell'applicazione. Il placeholder del campo di testo mira a dare un'indicazione ben precisa all'utente su quelle che potrebbero essere delle buone parole chiave atte a garantire risultati in linea alle aspettative: marca, prodotto o modello.

La presenza di testo all'interno del campo abilita la ricerca che può essere lanciata tramite la digitazione del tasto `Invio` o cliccando sull'apposito bottone posto alla fine del campo. Un ulteriore bottone, che appare solo dopo aver digitato almeno un carattere, consente di ripulire il campo.

A questo punto, l'URL viene aggiornato con il percorso assegnato alla pagina della ricerca e con il parametro `q` il cui valore è il testo prelevato dal campo. Nel DOM viene quindi caricato il componente incaricato di mostrare i risultati, che a sua volta invoca l'endpoint `/search/wide`. Questo accetta in ingresso il parametro `q` e restituisce un dizionario con quattro valori:

- `results`: array di oggetti contenenti le chiavi necessarie per reperire i dati relativi al gruppo o all'elemento;
- `total`: numero totale di risultati;
- `total_elements`: numero totale di risultati di tipo elemento;
- `total_groups`: numero totale di risultati di tipo gruppo;

Viene quindi effettuata la chiamata all'endpoint `/search/items-by-id` che prende in ingresso i primi venti elementi dell'array `results` e che restituisce a sua volta un array contenente i dettagli relativi ad ogni risultato;

Infine viene renderizzata la pagina dei risultati.

Nella parte alta è riportato il testo oggetto della ricerca e sono presenti delle `Chip` cliccabili che consentono di visualizzare i risultati nella loro totalità o di poterli filtrare per gruppi o elementi. In ogni bottone inoltre è riportato tra parentesi un'indicazione sul numero di risultati relativi al tipo di filtro.

Nella parte centrale della pagina sono mostrati i primi venti risultati all'interno di `Card` cliccabili. Queste riportano informazioni essenziali quali: serie

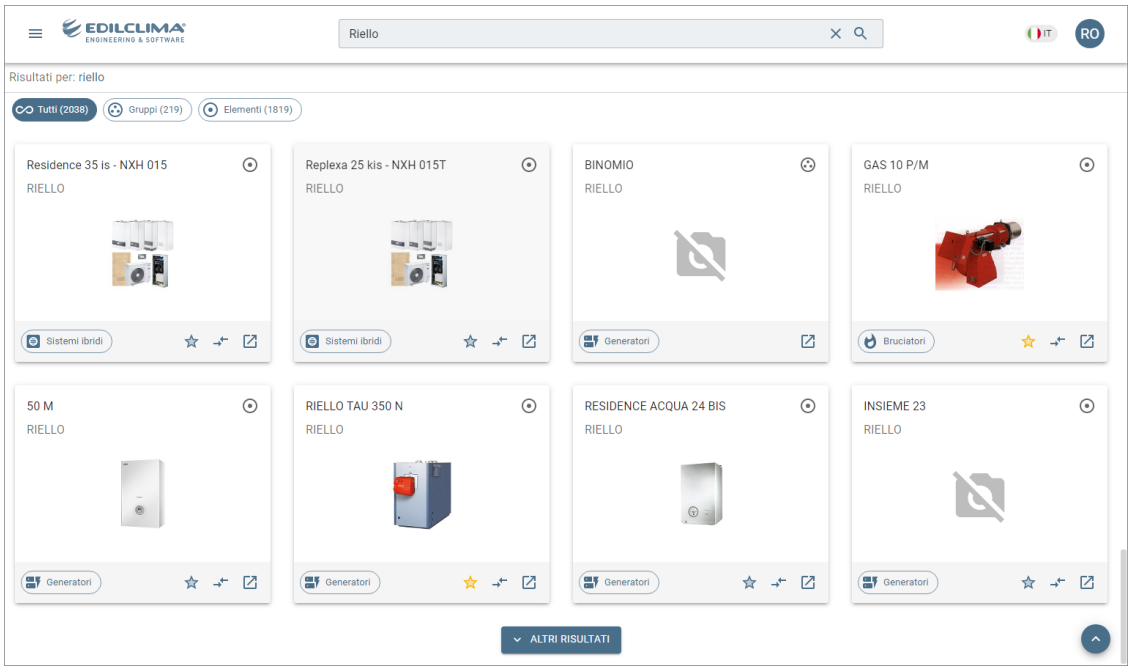

Figura 6.14: Pagina Ricerca di *Archivio Produttori*

o modello, marca, immagine e tramite l'icona posta nell'angolo in alto a destra, se si tratta di un risultato di tipo gruppo o di tipo elemento. Nella parte basse di ogni `Card` è riportata la categoria di appartenenza ed in base al tipo del risultato diverse possibilità di interazione:

- **Gruppo**: vi è esclusivamente la possibilità di aprire in una nuova tab del browser la pagina dei dettagli del gruppo;

- **Elemento**: oltre alla possibilità di aprire in una nuova tab del browser la pagina dei dettagli dell'elemento, è possibile aggiungere al confronto l'elemento e/o salvarlo nella lista degli elementi preferiti dell'utente.

Al fondo della pagina, il bottone `Altri risultati` consente di caricare e visualizzare ulteriori risultati, qualora ce ne siano, nel massimo di ulteriori venti unità. Infine il comodo `FAB` (Floating Action Button) consente di tornare rapidamente ai primi risultati della pagina.

6.5 Confronto

Il confronto è una funzionalità introdotta nell'archivio web per permettere agli utenti di avere una rappresentazione compatta e ravvicinata delle caratteristiche tecniche dei prodotti, al fine di paragonare i diversi parametri e gli attributi che li caratterizzano. Il confronto può avvenire tra un minimo di due e un massimo di quattro elementi appartenenti alla stessa categoria. Per implementare questa funzionalità è stato necessario realizzare:

- un componente `CompareBar`, che tiene traccia degli elementi aggiunti al confronto;
- una pagina appositamente dedicata alla visualizzazione dei dati comparati.

Un elemento può essere aggiunto tra quelli da sottoporre a confronto dalle seguenti pagine: Gruppo, Elemento e Ricerca.

CompareBar

Il componente CompareBar viene visualizzato nell'interfaccia dell'applicazione solo dopo aver cliccato almeno uno dei tanti bottoni presenti nell'applicazione che consentono di comparare l'elemento. Mantiene quindi traccia degli elementi che l'utente ha intenzione di confrontare. È posizionata in modo fisso nella parte bassa delle pagine dalle quali è possibile aggiungere un elemento al confronto ed è composta da quattro slot e da due bottoni: `Svuota` e `Confronta`. Cliccando su `Svuota` e se si conferma l'intenzione, tramite un `Alert`, vengono eliminati tutti gli elementi che erano stati selezionati per il confronto e la barra viene nascosta. Il bottone `Confronta`, invece, permette di procedere alla pagina dedicata alla comparazione delle specifiche se ci sono almeno due elementi da confrontare, altrimenti risulta disattivato. Ogni slot della barra è destinato ad un elemento da sottoporre a confronto e ne vengono rappresentati sempre quattro per dare un'indicazione all'utente del numero massimo di elementi che possono essere confrontati.

Un elemento può essere aggiunto tra quelli da confrontare tramite un bottone identificato da due frecce presente:

- nella barra delle azioni della card di un risultato di ricerca;

- nel `Datagrid` della pagina Gruppo;
- nella barra delle azioni nella pagina Elemento.

Ogni volta che si clicca sul bottone `Aggiungi al confronto` uno slot del componente `CompareBar`, inizialmente bianco, viene sostituito con un componente `Card` che rappresenta l'elemento da sottoporre a confronto e che contiene l'immagine e il nome del prodotto. Tramite il bottone (X) posto nell'angolo in alto a destra del componente è possibile eliminare un singolo elemento dalla barra liberando uno slot.

Figura 6.15: Compare Bar di *Archivio Produttori*

Il caso d'uso esemplificativo è il seguente. L'utente:

1. individua due o più elementi da aggiungere al confronto;
2. clicca sul bottone `Aggiungi al confronto`;
3. il prodotto viene aggiunto alla barra del confronto;
4. clicca sul bottone `Confronta` visualizzando così la pagina con i dati degli elementi comparati.

Pagina Confronto

La pagina del confronto è organizzata nel seguente modo: in alto è rappresentata un'intestazione di pagina fissa, nella quale vengono rappresentate le Card relative agli elementi che si stanno confrontando. Come previsto per la barra del confronto anche in questa pagina è possibile utilizzare il bottone (X) nell'angolo in alto a destra della `Card` per eliminare un elemento da quelli da confrontare.

Il contenuto della pagina è rappresentato da tre componenti `Accordion` che permettono di espandere o comprimere diversi insiemi di attributi da confrontare:

- **Gruppo**: racchiude tutti gli attributi relativi al gruppo di appartenenza degli elementi;

- **Elemento**: racchiude tutti gli attributi relativi al dettaglio dell'elemento;

- **Normative**: racchiude le certificazioni e norme di un elemento.

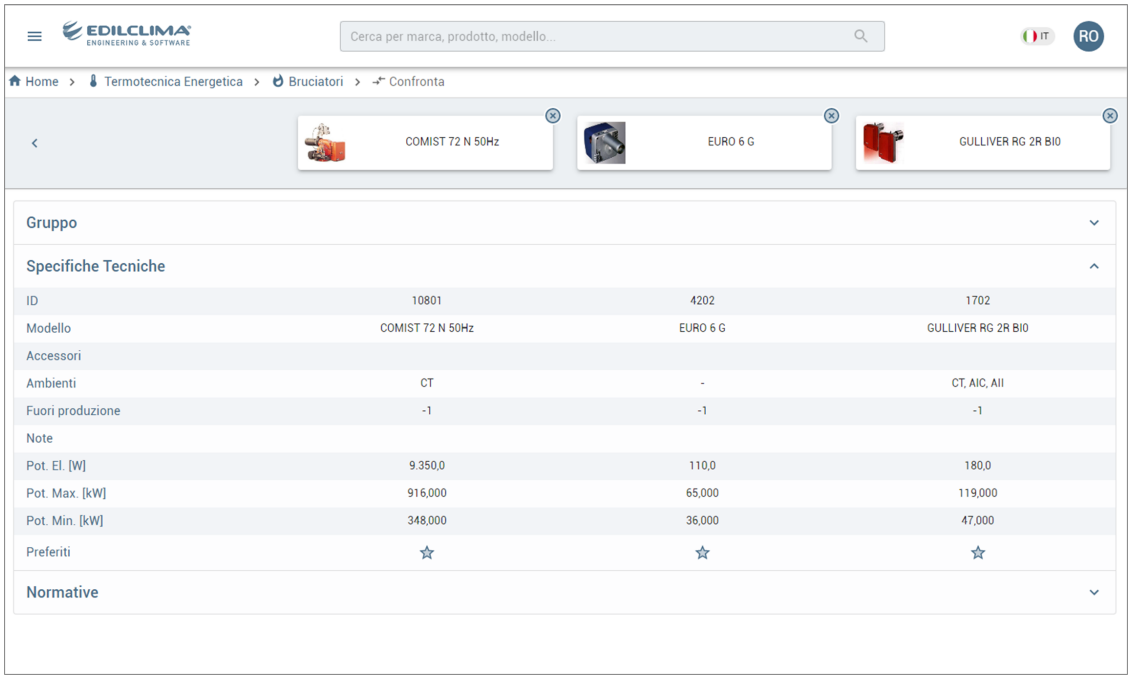

Figura 6.16: Pagina Confronto di *Archivio Produttori*

All'apertura della pagina, l'unico `Accordion` espanso è quello relativo ai dettagli tecnici dell'elemento; gli altri due presenti possono essere espansi qualora l'utente volesse avere una visione più ampia di tutti i dati ad esso relativi ovvero i dati del gruppo a cui appartiene e le eventuali normative di applicazione.

Gli attributi su cui è possibile confrontare i valori sono indicati sulla destra; mentre, in corrispondenza delle colonne, sotto ognuno degli elementi posti a confronto sono mostrati i relativi valori. Una lettura della tabella da destra verso sinistra permette subito di individuare per uno stesso attributo quali sono le differenze tra un valore e l'altro.
Tra i valori confrontabili è stata inserita una riga con il comando "Preferiti" che consente di aggiungere o rimuovere un elemento alla raccolta dei preferiti cliccando sul bottone identificato dalla stellina. Le funzionalità di questo comando sono state aggiunte anche in questa pagina poiché, molto spesso,

la fase di confronto potrebbe terminare con la scelta di uno o più prodotti preferiti rispetto ad altri.

Per poter condividere tramite URL i risultati di una comparazione, nonchè i dati degli elementi oggetto di tale operazione, è stata utilizzata la libreria *query-string* che ha permesso di inserire nell'URL i parametri necessari alla riproduzione fedele della pagina. Ogni volta che si esegue un caricamento o un aggiornamento dell'URL tramite il parse dei parametri contenuti nella query string, possono essere eseguite in parallelo le richieste agli endpoint al fine di ottenere tutti i dati necessari alla corretta visualizzazione della pagina.

Local storage

La funzionalità del confronto utilizza il *Local Storage* del browser per memorizzare al suo interno i prodotti aggiunti al confronto. In questo modo, anche in caso di chiusura accidentale o non del browser, al ripristino della sessione, i prodotti che erano stati selezionati per il confronto vengono mantenuti nella `CompareBar` e possono essere immediatamente confrontati.

Messaggi di errore

Nel caso di errori correlati all'aggiunta o alla rimozione di elementi dal confronta fuori dai limiti minimi e massimi di due e quattro, e nel caso di tentativo di aggiunta di prodotti appartenenti a categorie differenti, attraverso il componente `SnackBar` vengono mostrati i relativi messaggi di errore.

6.6 Editing

La funzionalità di modifica del catalogo è riservata agli utenti il cui ruolo prevede la possibilità di modificare di inserire o di eliminare gruppi e/o elementi del catalogo.

Allo stato attuale della soluzione questa funzionalità è presente in forma sperimentale per l'utente amministratore Edilclima e permette la modifica di elementi o gruppi già esistenti per due delle cinque categorie prese in analisi.

Il processo di modifica, implementato sotto forma di wizard, avviene all'interno di una finestra di dialogo dedicata che viene mostrata a partire dalla

pagina Categoria o Gruppo quando si clicca sul bottone modifica, identificato dall'icona di una matita.

La finestra di dialogo contiene:

- un **header**, nel quale è stato introdotto il componente `Stepper` che indica all'utente in quale fase della modifica si trova e le operazione che è chiamato a fare;
- un **content**, spesso un form con campi editabili, diverso per ogni fase di modifica prevista dal wizard;
- un **footer**, che contiene il bottone `Annulla`, quelli di navigazione che consentono di muoversi in avanti e indietro tra le fasi e il bottone `Salva` che permette di avviare la procedura di validazione dei form e di salvare i dati immessi.

Le fasi dello stepper sono state costruite attraverso lo hook personalizzato `useSteps` che permette al componente `Stepper` di conoscere il numero di passi da inserire nell'header della finestra di dialogo, i quali variano per le diverse categorie, per i gruppi e per gli elementi.

Le fasi sono navigabili in maniera sequenziale tramite i bottoni posizionati in basso nel footer della finestra di dialogo oppure accedendo direttamente alla fase che contiene i dati da modificare cliccando sulla relativa indicazione nell'header della finestra di dialogo. Ad ogni fase può corrispondere un insieme di campi testuali e/o numerici di valori relativi al gruppo o all'elemento oppure una maschera di dettaglio che può includere grafici o tabelle che eseguono calcoli automatizzati sulla modifica di alcuni parametri.

La modifica dei dati avviene tramite appositi form presenti in ogni fase prevista dal wizard. Questi sono opportunamente validati e consentono di prelevare e gestire i valore inseriti dall'utente in modo da poter impacchettare un oggetto contenente le modifiche da inviare all'endpoint dedicato sul server. Per la costruzione e validazione dei form sono state utilizzate le librerie *Formik* e *YUP*.

Queste si occupano di segnalare in maniera opportuna errori in fase di compilazione dei form e permettono di attivare i meccanismi che consentono ad esempio di evidenziare i campi errati in rosso, di mostrare un breve messaggio di testo di aiuto per l'utente e di segnalare al wizard la presenza di errori durante la modifica del dato.

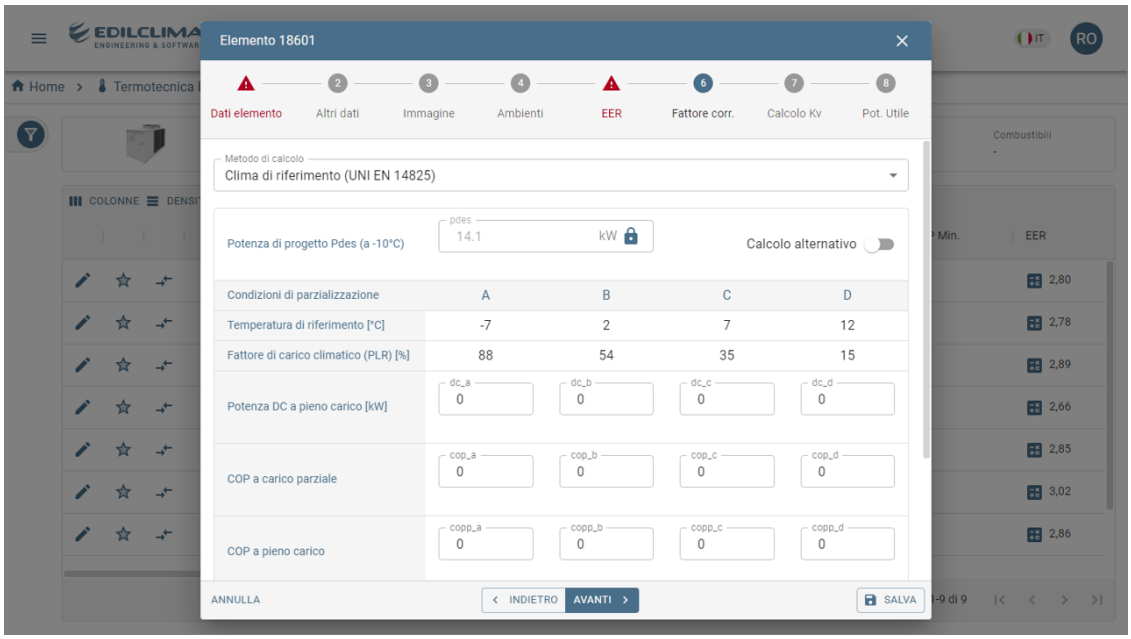

Figura 6.17: Wizard di modifica di un elemento di *Archivio Produttori*

6.7 Gestione degli errori

Uno dei principi più importanti dell'usabilità consiste nel fornire un adeguato feedback all'utente per ogni azione che viene eseguita. Per rispettare questo principio sono stati utilizzati i `Loader` e gli `Skeleton` durante gli stati di caricamento delle informazioni e in caso di errori o problemi improvvisi durante la navigazione è stato implementato un componente generalizzato `MessageScreen` per mostrare la pagina di errore offrendo un feedback visivo all'utente. Tale componente, in base alla tipologia dell'errore che si verifica mostra un'icona, un messaggio, una descrizione ed eventualmente un bottone che rimanda l'utente alla Homepage. Tale componente viene renderizzato nei seguenti casi:

- viene digitata un URL errato al quale non corrisponde alcuna pagina;
- si verifica un errore interno al server;
- il dato richiesto non è disponibile;
- la ricerca non restituisce nessun risultato.

Figura 6.18: MessageScreen di *Archivio Produttori*

6.8 Altre librerie

Durante lo sviluppo del prodotto sono state incluse nel progetto le seguenti librerie utili alla risoluzione di alcuni task:

- **country-flag-icons**: mette a disposizione componenti React per la visualizzazione delle bandiere utilizzate nel menù per la selezione della lingua;
- **Lodash**: semplifica ulteriormente il linguaggio JavaScript fornendo funzioni modulari per iterare array, oggetti e stringhe;
- **query-string**: per inglobare o estrapolare parametri nell'URL; utilizzata nell'applicazione dei filtri nelle pagine Categoria, Gruppo e Ricerca e per mantenere gli id dei gruppi e degli elementi nella pagina Confronto;
- **Recharts**: per la costruzione di grafici all'interno di maschere di dettaglio di gruppi e/o di elementi.

6.9 Revisione del software

Durante lo sviluppo, il software è stato sottoposto a continue revisioni. Per perseguire gli obiettivi di scalabilità e modularità e per conferire al prodotto una solidità strutturale sono state necessarie diverse fasi di refactoring.

Contestualmente al rilascio di nuove versioni delle librerie utilizzate, sono stati eseguiti gli aggiornamenti necessari. Uno tra gli aggiornamenti più importanti ha riguardato la libreria grafica Material-UI che, in seguito al rilascio della nuova versione major, è stata rinominata MUI.

Tale aggiornamento ha comportato una significativa fase di migrazione affinché si potesse godere dei miglioramenti del nuovo motore di styling e beneficiare di tutte le correzioni di bug introdotte da MUI.

Capitolo 7

Conclusioni

Lo svolgimento di questo lavoro di tesi è stata un'importante e preziosa occasione di crescita dal punto di vista professionale e personale, durante la quale ci si è imbattuti in un progetto con un'elevata complessità strutturale e di informazioni.

L'implementazione front-end di *Archivio Produttori* è stata una sfida tecnologica durante la quale sono state applicate competenze e conoscenze acquisite durante il percorso di studi, ma soprattutto è stata anche l'occasione di apprendere nuove tecnologie, tra cui React che ad oggi è molto utilizzata e per via delle sue potenzialità nutre larga approvazione nel mondo dello sviluppo web.

Personalmente, invece, l'esperienza è stata un'opportunità di crescere, di relazionarsi con una realtà lavorativa di gruppo dinamica, in cui la fondamentale collaborazione e contaminazione di idee diventano vettori di arricchimento personale.

Inoltre, il confronto diretto con i committenti è un modo di ottenere riscontri immediati e di superare le barriere comunicative evitando così fraintendimenti o insoddisfazione.

7.1 Sviluppi futuri

Per il futuro i propositi di crescita e di miglioramento del progetto sono ancora molti. In questo paragrafo vengono illustrate le principali proposte e idee

necessarie alla finalizzazione e al rilascio definitivo del prodotto, suddivise per tematiche.

Navigazione: L'archivio verrà esteso per consentire la navigazione di tutte le trentadue categorie presenti nel database. La scalabilità della soluzione dovrebbe consentire l'inserimento in maniera quasi automatizzata e richiederà unicamente la gestione di eventuali attributi o casistiche particolari.

Editing: La funzionalità verrà estesa a tutti i gruppi ed elementi di ogni categoria. Inoltre, bisognerà realizzare le funzionalità di inserimento ed eliminazione di un prodotto e gestire in maniera appropriata l'editing sulla base dei diversi ruoli: produttore, utente abituale o amministratore. Per quanto riguarda l'inserimento di nuovi dati da parte dei produttori è prevista anche la realizzazione di un'apposita dashboard per la gestione del flusso di approvazione dei dati che, prima di essere integrati nel database pubblico dell'archivio, necessitano il controllo e l'approvazione da parte dell'utente amministratore.

Ricerca: è prevista l'implementazione della "quick search" in supporto alla funzionalità di ricerca che fornisce istantaneamente un sottoinsieme di risultati coerenti al testo digitato nella barra di ricerca. Per migliorare ulteriormente questa funzionalità potrebbe essere gestita la cronologia di ricerca, in modo da salvare le ricerca frequenti dell'utente e, infine, nella pagina dei risultati di ricerca si potrebbe anche aggiungere la possibilità di ordinare, secondo diversi criteri, i risultati ottenuti.

Area Utente: verrà ampliata, aggiungendo la possibilità di modificare i dati personali associati all'utente, di stabilire diverse preferenze, tra cui la scelta di colonne da visualizzare di default per le `Datagrid` che contengono i dati dell'archivio o la possibilità di inserire un'immagine di profilo come avatar.

Infine, l'aspetto più importante tra gli sviluppi futuri riguarderà la pubblicazione online dell'archivio e l'integrazione dello stesso nel contesto dei software di calcolo proprietari di *Edilclima S.r.l.* affinché gli utilizzatori possano usufruire di tutti i vantaggi in termini di usabilità che sono stati curati durante lo sviluppo della soluzione web.

Appendice A

Icone

Pagina	Icona
Produttori	
Produttore	
News	
Software Edilclima	
FAQ	
Contacts	
Gruppo	
Elemento	
Confronto	

Tabella A.1: Icone delle pagine

Area	**Icona**
Termotecnica Energetica	
Impianti e Acustica	
Antincendio	
Utilità	

Tabella A.2: Icone delle aree

Categoria	**Icona**
Bruciatori	
Generatori	
Pompe	
Pompe di calore	
Sistemi ibridi	

Tabella A.3: Icone delle categorie

Bibliografia

[1] Anthony Accomazzo, N. M., *Fullstack React: The Complete Guide to ReactJS and Friends*, Fullstack.io, 2017

[2] Frost, O. R., *React Native*, Independently published, 2020

[3] Miller Daniel E., *React Native*, Independently Published, 2021

[4] Jacopo Pasquini, S. G., *Web Usability*, Milano, Hoepli, 2014

[5] Sidelnikov, G., *React.js Book*, Independently published, 2017

[6] Jain, R., *Infographic: Top JavaScript Frameworks To Look Out For In 2019*,
URL: `https://www.lambdatest.com/blog/infographic-top-javascript-frameworks-for-2019`

[7] Raymond Paulina, *Life as a UX designer*,
URL: `https://www.asocietygroup.com/news/ux-designer-news-a-society`

Sitografia

[1] Axios,
URL: https://axios-http.com

[2] Create React App,
URL: https://create-react-app.dev

[3] country-flag-icons,
URL: https://gitlab.com/catamphetamine/country-flag-icons

[4] Edilclima S.r.l.,
URL: https://www.edilclima.it

[5] Formik,
URL: https://formik.org

[6] i18next,
URL: https://www.i18next.com

[7] Introduzione a JSX,
URL: https://it.reactjs.org/docs/introducing-jsx.html

[8] JavaScript,
URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[9] JWT,
URL: https://jwt.io

[10] Lodash,
URL: https://lodash.com

[11] Material Design,

URL: https://material.io/design

[12] MUI,
URL: https://mui.com

[13] Npm,
URL: https://npmjs.com

[14] query-string,
URL: https://github.com/sindresorhus/query-string

[15] React,
URL: https://it.reactjs.org

[16] React Helmet,
URL: https://github.com/nfl/react-helmet

[17] React Router,
URL: https://reactrouter.com

[18] React Virtualized AutoSizer,
URL: https://github.com/bvaughn/react-virtualized-auto-sizer

[19] Recharts,
URL: https://recharts.org

[20] State of JS 2020,
URL: https://2020.stateofjs.com

[21] Swagger,
URL: https://swagger.io

[22] YUP,
URL: https://github.com/jquense/yup