



# Università degli Studi di Ferrara

UNIVERSITÀ DEGLI STUDI DI FERRARA  
CORSO DI LAUREA IN INFORMATICA

---

*Implementazione di una Progressive  
Web Application in VueJS per la  
gestione delle scorte di magazzino*

---

*Relatore:*

Prof. Giacomo PIVA

*Laureanda:*

Ilaria ZUCCHINI

ANNO ACCADEMICO 2023 – 2024



---

# *Indice*

---

	Page
<b>Introduzione</b>	<b>4</b>
<b>1 Cos'è una PWA?</b>	<b>7</b>
1.1 Descrizione . . . . .	7
1.2 Aspetti . . . . .	8
<b>2 Tecnologie usate</b>	<b>11</b>
2.1 Progettazione e Sviluppo front-end . . . . .	12
2.1.1 Figma . . . . .	12
2.1.2 HTML . . . . .	12
2.1.3 CSS . . . . .	13
2.1.4 Bootstrap 5 . . . . .	13
2.1.5 JavaScript . . . . .	14
2.2 Manifest e Service Worker . . . . .	14
2.2.1 Manifest . . . . .	14
2.2.2 Service Worker . . . . .	16
2.3 API . . . . .	16
2.4 Testing . . . . .	17
2.4.1 L'esecuzione su localhost . . . . .	17
2.4.2 Chrome DevTools . . . . .	17

<b>3</b>	<b>Implementazione della PWA</b>	<b>19</b>
3.1	App.vue . . . . .	19
3.2	routers.js . . . . .	20
3.3	Login . . . . .	22
3.4	Clienti . . . . .	24
3.5	Menù e profile card . . . . .	27
3.5.1	Menù . . . . .	27
3.5.2	Profile card . . . . .	28
3.6	Home e ordini aperti . . . . .	29
3.7	Creare e accettare un ordine . . . . .	32
3.8	Modale . . . . .	35
<b>4</b>	<b>Risultati e conclusione</b>	<b>39</b>

---

# *Introduzione*

---

Oggigiorno, l'accesso a internet e il suo utilizzo sono indispensabili per ciascuno di noi, in ogni ambito e in ogni momento.

Nella sfera privata, ad esempio, internet viene utilizzato per svago (videogiochi, musica, ecc.), per restare in contatto con le persone, per pagare servizi, per rimanere informati, per acquistare prodotti e via dicendo.

Nell'ambito lavorativo, internet viene utilizzato per restare in contatto con fornitori e clienti. Andando sempre di più verso un mondo digitalizzato si rende necessaria per le aziende la possibilità di comunicare con clienti e fornitori in modo immediato e automatico, proprio per questo motivo molte realtà si appoggiano a software che, ad esempio, mandano una email automatica all'evasione di un ordine, comunicando al cliente che il proprio ordine è stato inviato.

Inoltre internet è ormai indispensabile anche per dare la possibilità ai dipendenti di accedere alle informazioni di loro competenza su un dispositivo direttamente dalla loro postazione, senza dover ricorrere a raccoglitori o fogli stampati, situati in altre stanze magari lontane da loro. L'accesso a queste informazioni in modo immediato ha sicuramente un ruolo cruciale nell'ottimizzazione dei tempi all'interno delle aziende.

Ma come poter dare la possibilità ai dipendenti di accedere a queste informazioni?

Una soluzione per le aziende può essere l'utilizzo di una Progressive Web App (PWA), la quale permette di essere utilizzata su computer e su dispositivi mobili adattandosi allo schermo sul quale viene aperta in modo da essere sempre ordinata e intuitiva, si può accedere ad essa dal browser nonostante ci sia anche la possibilità di installarla sul dispositivo.

Il testo seguente tratterà proprio dello sviluppo di una Progressive Web App (PWA) destinata all'utilizzo da parte dei tecnici autorizzati per la creazione e l'accettazione degli ordini.

---

## *Cos'è una PWA?*

---

Prima di approfondire gli aspetti pratici, è importante chiarire che cosa sia una PWA.

Le PWA (Progressive Web Apps per esteso, che in italiano viene tradotto come Applicazioni Web Progressive), sono delle applicazioni realizzate tramite le tecnologie del web. Queste applicazioni pur risiedendo su browser riescono a offrire un'esperienza utente del tutto simile a quella di un'applicazione nativa installata su un dispositivo.

Inoltre, grazie alla loro versatilità, possono funzionare su svariate tipologie di dispositivi e su diversi browser partendo da un unico codice base, permettendone l'utilizzo a un pubblico elevato.

### **1.1 Descrizione**

Le PWA, a differenza delle tradizionali applicazioni native che troviamo installate sui dispositivi, sono un ibrido tra queste e le tradizionali pagine web. Durante l'implementazione di questo tipo di applicazioni, si cerca di sfruttare al meglio le potenzialità offerte dai browser, combinandole con la comodità e i vantaggi delle applicazioni native nell'uso su dispositivi mobili.

Il termine "*Progressive*" implica che questo tipo di applicazioni possono essere eseguite su qualsiasi tipologia di dispositivo e su qualsiasi browser, questo, per ogni utente.

Le PWA sono applicazioni installabili; il browser infatti può proporre all'utente di installare la PWA sul dispositivo.

Una volta installata, la si potrà visualizzare nella pagina principale del dispositivo tramite la relativa icona. L'utente in questo modo, percepirà la PWA appena installata proprio come un'applicazione nativa.

## 1.2 Aspetti

Le PWA, inoltre, presentano diversi aspetti da tenere in considerazione:

- **Progressive:** come spiegato in precedenza, le PWA funzionano per ogni utente su ogni browser e su ogni dispositivo.
- **Responsive:** si adattano alle dimensioni dello schermo del dispositivo in cui vengono utilizzate (smartphone, tablet, desktop, ecc.), questo permette una visualizzazione ottimale dell'applicazione, sia su schermi piccoli che su schermi grandi.
- **Sicure:** funzionano su protocollo HTTPS<sup>1</sup>, il che impedisce che informazioni o contenuti vengano alterati.
- **Installabili:** l'installazione è possibile effettuarla sia dall'app store (nel caso il produttore la pubblichi) che direttamente dal web, dove apparirà una notifica pop-up all'apertura dell'applicazione da browser che chiederà all'utente se la si vuole installare.

Una volta installata il suo comportamento è analogo a un'applicazione nativa, la visualizzazione avverrà a schermo intero mentre la barra degli indirizzi verrà nascosta.

- **Aggiornate:** queste applicazioni, risiedendo sul web, possono essere aggiornate in background, consentendo agli utenti di avere accesso sempre all'ultima versione.

---

<sup>1</sup>HyperText Transfer Protocol over Secure Socket Layer



- **Integrate:** possono essere integrate con i sistemi operativi, il che comporta la possibilità di avere funzionalità attraverso l'utilizzo di API<sup>2</sup>, come, ad esempio, la PWA potrà accedere alla fotocamera, al GPS o al microfono del dispositivo, il quale potrà anche ricevere delle notifiche push.
- **Offline:** funzionano anche offline o in background grazie alla presenza dei Service Worker; questo garantisce continuità all'utente.
- **Linkabili:** sono facilmente condivisibili tramite l'URL<sup>3</sup>. [7]

---

<sup>2</sup>Application programming interface

<sup>3</sup>Uniform Resource Locator



---

# *Tecnologie usate*

---

Sviluppare questa *Progressive Web App* è stato possibile integrando diverse tecnologie selezionate per garantire un'interfaccia intuitiva e un'esperienza utente fluida. In questa capitolo vengono descritte le principali tecnologie adottate e il loro ruolo nel progetto.

La progettazione dell'interfaccia è stata effettuata con Figma, mentre lo sviluppo è stato realizzato con *HTML*, *CSS*, *Bootstrap* e *JavaScript*, avvalendosi del framework *Vue.js*. Per la comunicazione con le API è stato utilizzato *Axios*, mentre l'implementazione delle funzionalità PWA è stata resa possibile grazie a *Manifest* e *Service Worker*. Infine, per il testing è stato impiegato *Chrome DevTools*.

*Vue.js* è un framework cioè una struttura usata come base per facilitare lo sviluppo software. Al posto del classico DOM<sup>1</sup>, *Vue.js* usa un Virtual DOM in modo da poter aggiornare solo le parti necessarie di una pagina senza necessità di dover ricaricare tutto.

---

<sup>1</sup>Document Object Model

## 2.1 Progettazione e Sviluppo front-end

### 2.1.1 Figma

*Figma* è un software di progettazione UI/UX, utilizzato per aggiungere blocchi e colori, definendo la struttura e l'aspetto visivo delle diverse pagine. Sebbene non siano state utilizzate funzionalità avanzate, Figma ha fornito un ambiente utile per visualizzare e organizzare le idee progettuali prima di passare allo sviluppo vero e proprio.

Al termine del progetto, la PWA ha un aspetto completamente diverso rispetto alla prima bozza, poiché alcune parti sono state modificate per risultare più leggere e gradevoli agli occhi degli utenti.

### 2.1.2 HTML

*HyperText Markup Language* (in italiano, Linguaggio di marcatura ipertestuale) è il componente più basilare del Web, definisce il significato e la struttura delle pagine web. Altre tecnologie oltre a HTML sono generalmente utilizzate per descrivere l'aspetto (CSS) o la funzionalità/comportamento (JavaScript) di una pagina web. "Ipertestuale" si riferisce ai link che collegano le pagine web tra loro, sia all'interno di un singolo sito web che tra siti web.

L'HTML utilizza il "markup" per annotare testo, immagini e altri contenuti da visualizzare in un browser Web. Il markup HTML include "elementi" come <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, <div>, e molti altri.

Un elemento HTML è separato da altro testo in un documento da "tag", che consistono nel nome dell'elemento circondato da < e >. Il nome di un elemento all'interno di un tag non è sensibile alle maiuscole e alle minuscole. Vale a dire, può essere scritto in maiuscolo, minuscolo o in una combinazione. Tuttavia, la convenzione e la pratica consigliata è di scrivere i tag in minuscolo. [5]

### 2.1.3 CSS

*Cascading Style Sheets*, in italiano "Fogli di stile a cascata" è un linguaggio che descrive come gli elementi HTML devono essere visualizzati (colori, layout, font, ecc.). Ad esempio:

```
1 p{  
2   color: blue;  
3   font-size: 16px;  
4 }
```


Questo renderà il testo presente nei tag `<p>` (paragraph) di colore blu e di una grandezza di 16 pixel.

*Cascading* si riferisce al fatto che le regole del linguaggio seguono una gerarchia. Il CSS è uno dei linguaggi principali del web ed è standardizzato secondo le specifiche W3C<sup>2</sup>.[\[4\]](#)

### 2.1.4 Bootstrap 5

*Bootstrap* nasce nel 2010, è un framework front-end open source ed è una raccolta di strumenti grafici, stilistici e di impaginazione che possono essere utilizzati per lo sviluppo di applicazioni web responsive. Per usarlo basta inserire all'interno dei tag HTML la classe predefinita di cui si ha bisogno. Un esempio:

```
1 <div class="bg-dark rounded">  
2   <p class="text-white fw-bold">Lorem ipsum  
   dolor sit amet.</p>  
3 </div>
```



**Lorem ipsum dolor sit amet.**

Figura 2.1: Risultato utilizzo classi

---

<sup>2</sup>W3C: World Wide Web Consortium

Nel tag `<div>`, le classi aggiunte:

- *bg-dark*: imposta lo sfondo nero.
- *rounded*: impostare l'arrotondamento degli angoli dei bordi.

Al tag `<p>`, invece, sono state aggiunte delle classi per:

- *text-white*: impostare il colore del testo.
- *fw-bold*: aumentare lo spessore del testo.

### 2.1.5 JavaScript

JavaScript è un linguaggio di programmazione leggero e interpretato (*just-in-time-compiled*). È principalmente conosciuto per l'utilizzo nella programmazione di pagine web, ma può anche essere usato in ambienti non browser.

JavaScript, inoltre, è un linguaggio che supporta diverse tipologie di programmazione, un esempio, la programmazione ad oggetti.[\[6\]](#)

## 2.2 Manifest e Service Worker

Le PWA si basano su due file fondamentali che ne garantiscono il funzionamento: il `manifest.json` e il `serviceWorker.js`.

### 2.2.1 Manifest

Il `manifest.json` è un file JSON contenente le informazioni di base sulla PWA (nome, url di inizio, colore dello sfondo, ecc.) e su come questa dovrà poi comportarsi una volta installata, contiene anche le icone (nelle varie dimensioni) e (non obbligatoriamente) gli screenshot dell'applicazione.

Di seguito, il `manifest.json` della PWA, il nome presente non è il nome originale, ma è un nome fittizio.

```
1 {  
2   "name": "ExamplePWA",  
3   "short_name": "Example",
```

```

4  "theme_color": "#022970",
5  "icons": [
6    {
7      "src": "./img/icons/favicon48x48.ico",
8      "sizes": "48x48",
9      "type": "image/x-icon"
10   },
11   {
12     "src": "./img/icons/favicon96x96.png",
13     "sizes": "96x96",
14     "type": "image/png"
15   },
16   {
17     "src": "./img/icons/favicon192x192.png",
18     "sizes": "192x192",
19     "type": "image/png"
20   },
21   {
22     "src": "favicon.ico",
23     "sizes": "256x256",
24     "type": "image/x-icon"
25   },
26   {
27     "src": "./img/icons/favicon512x512.png",
28     "sizes": "512x512",
29     "type": "image/png"
30   }
31 ],
32 "start_url": "/",
33 "display": "standalone",
34 "background_color": "#022970",
35 "screenshots": [
36   {
37     "src": "./img/screenshots/Desktop.png",
38     "sizes": "3061x1759",
39     "type": "image/png",
40     "platform": "wide"
41   }
42 ]
43 }

```

Come si può notare sono presenti le icone in tutte le dimensioni necessarie, è presente l'URL di inizio che sarà il primo URL aperto all'avvio dell'applicazione, c'è il colore del tema e infine è presente uno screenshot dell'applicazione.

### 2.2.2 Service Worker

Il `serviceWorker.js` è un file JavaScript, agisce da server remoto e si trova tra la PWA, il browser e il network. Il suo compito è quello di intercettare le richieste di rete, prendere decisioni in base alla disponibilità del network e aggiornare le risorse che risiedono sul server. I service workers sono eventi asincroni e vengono eseguiti in un maniera separata dal thread principale dell'applicazione.

Consente, infine, la sincronizzazione in background delle API<sup>3</sup> e l'accesso alle notifiche push. [8]

## 2.3 API

Le *Application Programming Interface* (in italiano, Interfaccia di programmazione delle applicazioni) o semplicemente API, sono un insieme di procedure che consentono la comunicazione tra prodotti e servizi senza dover necessariamente conoscere l'implementazione del prodotto o del servizio con cui si sta comunicando, consentendo così un grande risparmio sia di tempo che di denaro.

Definendo la modalità d'interazione tra applicazioni, le API consentono l'invio e la ricezione di dati.[2]

All'interno del progetto, le chiamate API sono state implementate tramite l'uso di Axios, una libreria JavaScript che consente di connettersi con le API di backend e gestire le richieste tramite il protocollo HTTP.

Essendo promise-based, Axios supporta l'implementazione di codice asincrono, il programma potrà quindi inviare la richiesta API senza dover interrompere l'esecuzione di altre operazioni.

---

<sup>3</sup>API: Application Programming Interface



## 2.4 Testing

Per testare il funzionamento della PWA ci si è serviti di due strumenti: l'esecuzione su localhost e Chrome DevTools.

### 2.4.1 L'esecuzione su localhost

Per questo tipo di esecuzione viene avviato un server di sviluppo locale tramite il comando `npm run serve` (su Vue.js) che permette la visualizzazione in tempo reale dell'applicazione e delle modifiche effettuate.

### 2.4.2 Chrome DevTools

*Chrome DevTools* è un'insieme di strumenti per sviluppatori web direttamente all'interno di Google Chrome.

Attraverso questo strumento è possibile modificare istantaneamente parti di codice della pagina che si sta visualizzando. La modifica non avviene in maniera definitiva e al ricaricamento della pagina le modifiche vengono annullate se non precedentemente riportate sul codice nell'editor che si sta utilizzando (in questo caso Visual Studio Code).

È possibile aprire Chrome DevTools tramite tasto destro del mouse e selezionare "ispeziona" oppure con la sequenza di tasti: `Ctrl+Maiusc+C`, se ci si trova su Mac basterà premere Opzione invece del tasto Maiusc.<sup>[1]</sup>



---

## Implementazione della PWA

---

La PWA oggetto di questa tesi è composta da svariati file, ognuno dei quali è composto da due o tre parti: `<template></template>` per il codice HTML, `<script></script>` per il codice JavaScript e `<style></style>` (non sempre è presente) per il codice CSS.

### 3.1 App.vue

*App.vue* è il file base della *Progressive Web App*, ogni componente (in Vue, ciò che abitualmente è conosciuta come pagina è in realtà un componente Vue) per essere visualizzato, viene caricato passando attraverso questo file.

Ne viene riportato un brevissimo pezzo di codice:

```
1 <template>
2   <router-view/>
3 </template>
```

`<router-view/>` è il componente di Vue.js che permette la visualizzazione del contenuto del componente in base all'URL corrente.

Ad esempio, se l'utente visita la pagina Home, il componente Home.vue verrà reindirizzato da Vue Router all'interno del tag che a sua volta caricherà la pagina e la mostrerà.

*App.vue* presenta all'interno del tag `<script>` un'API, la sua chiamata inizia automaticamente e effettua un controllo sul server, assicurandosi che sia online.

Infine, all'interno del tag `<style>` contiene il codice CSS per quasi tutta l'applicazione, tra cui il font per l'intera applicazione.

## 3.2 routers.js

All'interno del file *routers.js* vengono create le rotte per raggiungere i componenti dell'applicazione.

Ogni componente viene importato all'inizio del file e ad ognuno di essi viene assegnato un nome e un path, questo permette di realizzare i vari reindirizzamenti necessari tra le pagine.

Il path verrà visualizzato nella barra degli indirizzi qualora la PWA venisse aperta su browser.

Viene inoltre creata anche un'istanza che gestisce la cronologia del browser e le modifiche degli URL, ciò permette di non ricaricare completamente la pagina, ma di ricaricare solo gli elementi dinamici. Il metodo utilizzato per la creazione dell'istanza è `createWebHistory()`. Si riporta il codice del file *routers.js*:

```
1 import Home from "./components/Home.vue";
2 import Login from "./components/Login.vue";
3 import Order from "./components/Order.vue";
4 import Accept from "./components/Accept.vue";
5 import Client from "./components/Client.vue";
6 import OpenOrder from "./components/OpenOrder.vue";
7 import { createRouter, createWebHistory } from 'vue-router'
8 const routes = [
9   {
10     path: '/',
11     redirect: '/login'
```

```
12 },
13 {
14   name: "Login",
15   component: Login,
16   path: "/login",
17 },
18 {
19   name: "Home",
20   component: Home,
21   path: "/home",
22 },
23 {
24   name: "Order",
25   component: Order,
26   path: "/order",
27 },
28 {
29   name: "Accept",
30   component: Accept,
31   path: "/accept",
32 },
33 {
34   name: "Client",
35   component: Client,
36   path: "/client",
37 },
38 {
39   name: "OpenOrder",
40   component: OpenOrder,
41   path: "/open_order"
42 }
43 ];
44
45 const router = createRouter({
46   history: createWebHistory(),
47   routes
48 });
49
50 export default router;
```

All'interno di `routes` è possibile notare che il primo elemento è:

```
{ path: '/', redirect: '/login' }
```

questo è dovuto al fatto che all'apertura, la *Progressive Web App* cercherà di aprire '/', ma non trovando alcun reindirizzamento possibile reindirizzerà a '/login'.

### 3.3 Login

La pagina login è visivamente molto pulita, presenta un form contenente gli input per inserire email e password.

Una volta inseriti i dati richiesti, è possibile controllare di aver inserito la password correttamente tramite il checkbox dedicato.

Il checkbox tramite la funzione `showPassword()` trasforma il tipo del tag da "password" a "text" e viceversa quando viene selezionato o deselezionato.

```
1 showPassword() {  
2   var pw = document.getElementById("password");  
3   if (pw.type === "password") {  
4     pw.type = "text";  
5   } else {  
6     pw.type = "password";  
7   }  
8 }
```

Dopo di che, premendo invio sulla tastiera o cliccando il bottone login viene chiamata l'API che, in caso di risposta positiva, restituisce i dati dell'utente e crea un token che permette l'accesso all'applicazione e reindirizza l'utente alla pagina di visualizzazione dei clienti.

In questa fase, viene controllato il valore restituito che deve essere uguale a 200 e viene controllato inoltre che l'array restituito non sia vuoto. Il controllo viene effettuato tramite il metodo:

```
Object.keys().length
```

`Object.keys()` viene utilizzato perchè i dati inizialmente restituiti sono di tipo oggetto, ma, per poter effettuare questo controllo usando `.length` si necessita di un array.

Questo metodo, quindi, trasforma i dati in un array.

Infine viene controllata la lunghezza di quest'ultimo, con, appunto, `.length`.

I dati dell'utente vengono salvati nella memoria locale del browser tramite il metodo `JSON.stringify()` di JavaScript, il quale converte i dati JavaScript in dati JSON.

Convertirli è necessario per salvarli e conseguentemente per permettere alla PWA di utilizzarli per mantenere l'accesso e per chiamare le API successive.

Di seguito il codice dei controlli e del salvataggio dei dati:

```
1   if (response.status == 200 && Object.keys(response.data).length
2       > 0) {
3       localStorage.setItem("token", JSON.stringify(response.data.
4           token));
5       localStorage.setItem("id", JSON.stringify(response.data.user.
6           id));
7       localStorage.setItem("nome_azienza", JSON.stringify(response.
8           data.user.nome_azienza));
9       localStorage.setItem("nome_referente", JSON.stringify(response
10          .data.user.nome_referente));
11      localStorage.setItem("email", JSON.stringify(response.data.
12          user.email));
13      localStorage.setItem("logo", JSON.stringify(response.data.user
14          .logo));
15      this.$router.push({ name: 'Client' });
16  } else {
17      console.warn("Login failed, invalid data..");
18  }
```

In caso di risposta negativa, viene restituito un messaggio di errore che l'utente visualizzerà in rosso tra il checkbox e il bottone di login. Il messaggio di errore riporterà "E-mail o password errati".

Qualora l'utente cercherà di accedere senza inserire uno dei due campi o di scrivere solo in modo parziale la propria email, non verrà restituito alcun messaggio di errore, ma apparirà un tooltip che informerà l'utente della necessità di compilare il campo. Il tooltip viene generato automaticamente inserendo (nel codice HTML) "required" alla fine del tag `<input>`.

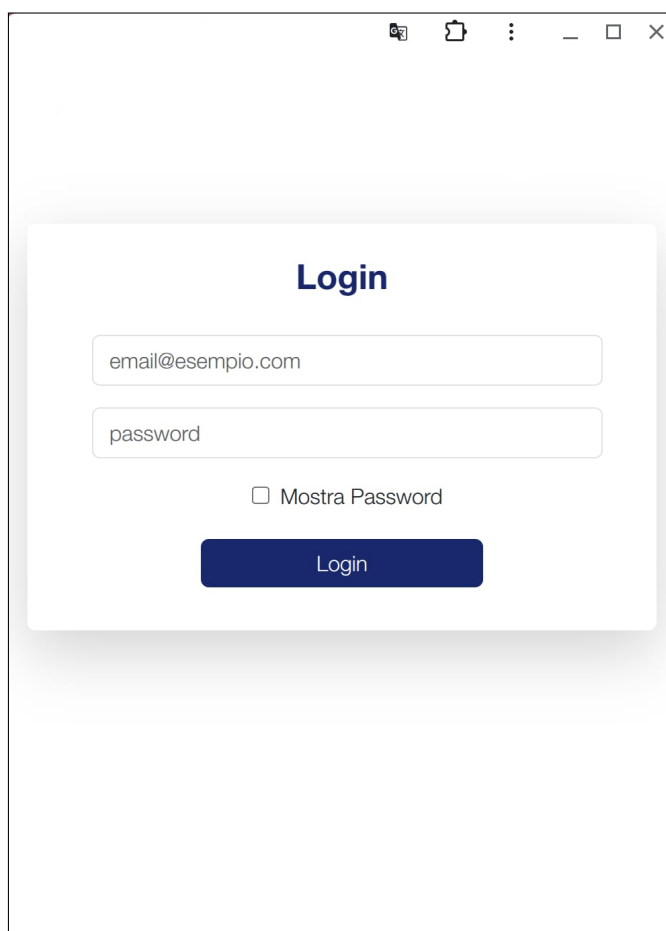
A screenshot of a web browser window displaying a login form. The browser's address bar is empty. The login form is a white card with a dark blue title "Login". It features two text input fields: the first contains "email@esempio.com" and the second contains "password". Below the password field is a checkbox with the label "Mostra Password". At the bottom of the card is a dark blue button with the text "Login" in white. The browser window has standard OS controls (minimize, maximize, close) in the top right corner.

Figura 3.1: Pagina di login

## 3.4 Clienti

Una volta giunti in questa pagina l'utente potrà effettuare la scelta del cliente per cui vuole effettuare un'operazione. Al caricamento della pagina, un'API verrà chiamata in maniera automatica per recuperare i dati relativi ai clienti dell'utente



appena loggato.

Ricevuta risposta positiva, tra le informazioni ricevute, l'unica che verrà visualizzata dall'utente sarà il nome dei clienti.

I nomi saranno visualizzati in un elenco, generato con la direttiva di Vue.js, `v-for=""`.

Questa direttiva funziona in una maniera molto simile a un ciclo for, ma con il vantaggio di essere reattiva: ogni volta che i dati cambiano, Vue aggiorna automaticamente il DOM senza bisogno di interventi manuali.

In questo modo, l'interfaccia rimane sempre sincronizzata con i dati disponibili.

Di seguito una parte del codice HTML:

```
1 <li v-for="customer in customers" :key="customer.id" class="client
  -list list-group-item">
2   <a @click="choose(customer)" class="text-decoration-none"> {{
     customer.nome_azienza }}
3     <i class="bi bi-chevron-right"></i>
4   </a>
5 </li>
```

La funzione chiamata al clic sul nome del cliente è `choose(customer)` (@click, in Vue.js equivale a "on click") e si occupa di gestire la selezione dell'utente eseguendo due operazioni principali.

In primo luogo, salva nella memoria locale del browser sia l'ID che il nome del cliente selezionato, assicurandosi che queste informazioni possano essere recuperate e utilizzate successivamente.

Subito dopo, effettua un reindirizzamento alla pagina home.

L'utilizzo di:

```
<i class="bi bi-chevron-right"></i>
```

permette di avere una freccia che punta verso destra subito dopo il nome del cliente, la quale migliora la chiarezza visiva dell'interfaccia e comunica in modo intuitivo che, cliccando sul nome, si accede a un'azione specifica. Infine, l'uso di una freccia orientata verso destra è comune nelle interfacce utente, rendendo l'esperienza più

fluida e familiare per l'utente.

Quindi per riassumere: l'utente sceglierà il cliente per cui vuole compiere un'azione, lo seleziona e verrà reindirizzato alla pagina Home, durante questo caricamento verranno salvati nella memoria del browser il nome e il codice identificativo del cliente.

L'interfaccia utente che verrà visualizzata:

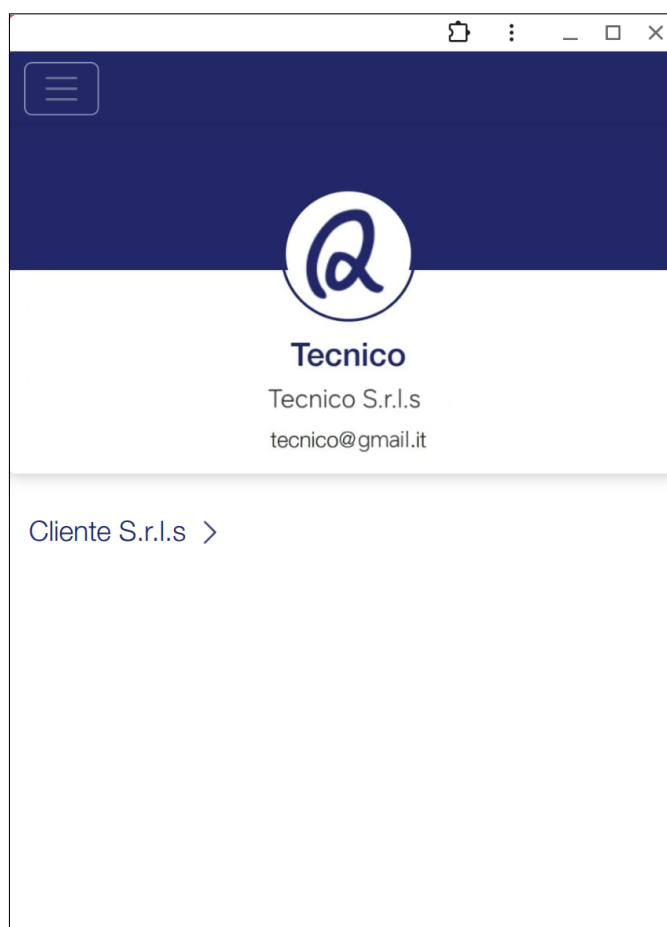


Figura 3.2: La scelta del cliente

## 3.5 Menù e profile card

L'header della PWA è composto unicamente dal menù e dalla profile card dell'utente, implementati tramite HTML e Bootstrap 5.

### 3.5.1 Menù

Nel menù sono presenti i collegamenti con la pagina Clienti, con gli ordini aperti (all'interno della pagina Home) e il logout.

Per quanto riguarda il logout, una volta cliccato nel menù, verrà chiamata l'API apposita che si occuperà di controllare il token dell'utente, scaricandolo dalla memoria locale del browser, e, ricevuta la risposta positiva verranno eliminati i dati dalla memoria del browser tramite il metodo:

```
localStorage.clear()
```

e l'utente verrà reindirizzato alla pagina di login.

Il menù ha la particolarità di essere reattivo alla grandezza dello schermo, infatti se visualizzato da desktop, quindi, da uno schermo grande, i componenti verranno mostrati uno di seguito all'altro, mentre se visto da uno schermo più piccolo (come uno smartphone o un tablet) o semplicemente ridimensionando la schermata da desktop, il menù verrà racchiuso in un menù a tendina.

In gergo tecnico, la classe `navbar-expand-lg` indica che il menù su schermi grandi sarà espanso mentre sarà collassato su schermi più piccoli, garantendo un'esperienza ottimale agli utenti.

Nel menù inoltre prima di ogni stringa è presente un'icona, la quale è stata inserita come classe bootstrap del tag `<a>`, questo tag (si trova all'interno di elenco) permetterà la reindirizzazione alla pagina scelta.

### 3.5.2 Profile card

La profile card è stata implementata con le classi dedicate di Bootstrap: `card`, `profile-card`, `profile-header` e `profile-body`. Se ne riporta il codice:

```
1 <div class="card profile-card">
2 <div class="profile-header">&nbsp;</div>
3 <div class="profile-body">
4 <div class="image-area">
5 
6 </div>
7 <div class="content-area">
8 <h3 class="my-2">{{ nome_referente }}</h3>
9 <p class="my-1">{{ nome_azienza }}</p>
10 <p>{{ email }}</p>
11 </div>
12 </div>
13 </div>
```

All'interno di `nome_referente`, `nome_azienza` e `email` vengono salvati i dati scaricandoli dalla memoria del browser con il metodo:

```
localStorage.getItem().replace(/"/g, '')
```

`replace()` è stato utilizzato perchè sulla memoria locale del browser i dati vengono salvati tra virgolette (") e con queste, vengono anche scaricati dalla memoria. La funzione sopra riportata, sostituisce le virgolette con il valore all'interno di "", che in questo caso è vuoto, perciò semplicemente, rimuove le virgolette. La lettera 'g' indica "globale", le virgolette verranno quindi cercate e rimosse in tutta la stringa.

La soluzione migliore per l'implementazione di questi due elementi (menù e profile card) è ricaduta sulla creazione di un componente Header.

Questa scelta è avvenuta rispetto a comodità e ordine all'interno del codice.

Questo componente, infatti, una volta creato viene semplicemente richiamato negli altri componenti tramite il proprio tag `<Header />`.

La visualizzazione del menù espanso su schermi piccoli e la visualizzazione della profile card:

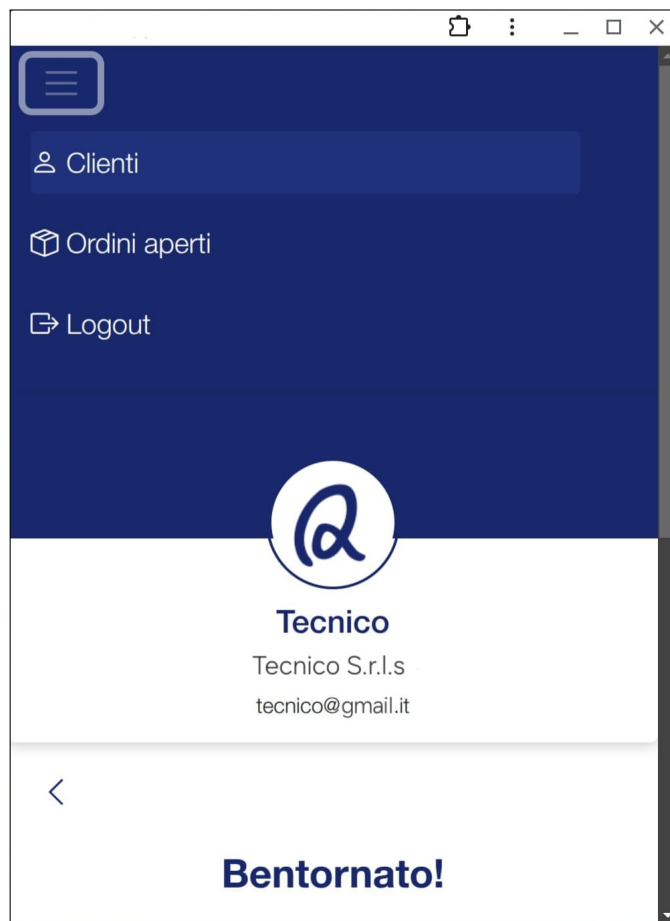


Figura 3.3: Il menù espanso e la profile card

Come si può notare, la profile card mostra le informazioni dell'utente loggato: il logo dell'azienda, il nome del referente, il nome dell'azienda e infine la email del referente.

## 3.6 Home e ordini aperti

Nella pagina Home è possibile decidere l'operazione da effettuare e visualizzare gli ordini aperti scorrendo infondo alla pagina.

Qui, si potrà anche essere certi di aver selezionato il cliente corretto in quanto è presente un sottotitolo <h2> che riporta il nome del cliente selezionato.

Il metodo utilizzato per visualizzare il nome del cliente è:

```
JSON.parse(localStorage.getItem('selectedCostumer'))
```

quindi, il nome viene preso dalla memoria locale del browser (dove era stato salvato poco prima) e tramite `JSON.parse()` viene trasformato in un valore JavaScript.

Le operazioni effettuabili tra cui l'utente potrà scegliere sono due:

- la creazione di un nuovo ordine
- l'accettazione di un ordine

L'implementazione dei bottoni per accedere alle operazioni è avvenuta tramite il tag HTML <button> e cliccandoli si verrà reindirizzati alla pagina di creazione o di accettazione, in base al bottone che si è cliccato.

Per visualizzare gli ordini aperti vengono presi dalla memoria locale del browser il token e l'id del cliente e vengono salvati in un array, dopodiché si procede con la chiamata API, la quale necessita di questi dati per restituire gli ordini aperti di quel determinato cliente.

Il codice che permette la visualizzazione degli ordini aperti nella pagina home:

```
1  async openOrders() {
2    let token = localStorage.getItem('token').replace(/"/g, '');
3    try {
4      let response = await axios.get('https://api/${this.
        customer.id}/orders', {
5        headers: {
6          'authorization': 'Bearer ${token}',
7          'accept': 'application/json',
8        },
9      });
10     console.log(response);
11     this.orders = response.data.orders;
12   } catch (error) {
13     console.warn("Error: ", error)
```

14  
15

```
}  
}
```

Nella pagina home sarà possibile vedere solo l'ID degli ordini aperti, per poter visionare il contenuto di un ordine sarà necessario cliccare sull'ID interessato per essere rimandati alla pagina dedicata **openOrder**, dove sarà possibile la visualizzazione dell'ordine per intero, visualizzando quindi i dettagli dell'ordine, come, ad esempio, gli articoli mancanti al completamento. In caso non siano presenti ordini aperti, verrà visualizzata la frase "Nessun ordine aperto".

Come conclusione della descrizione di questa pagina è utile informare che, in alto a sinistra (sotto la profile card, che si ricorda essere presente in tutte le pagine ad eccezione della pagina login) è presente una freccia che consente tornare alla pagina precedente.

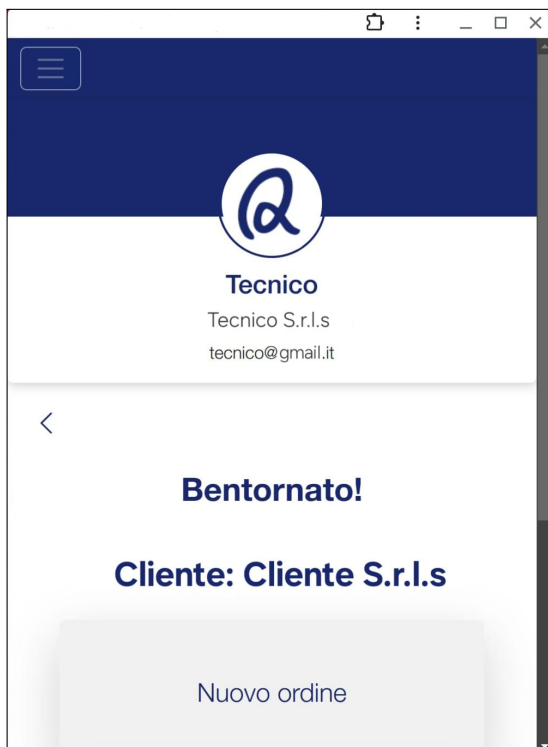


Figura 3.4: Pagina Home 1

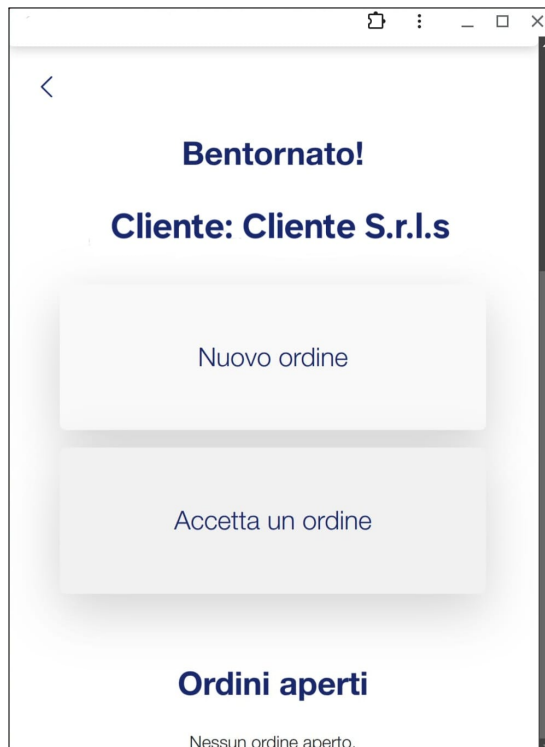


Figura 3.5: Pagina Home 2

## 3.7 Creare e accettare un ordine

Effettuata la decisione dell'operazione da effettuare si viene reindirizzati in una delle due pagine, la creazione di un nuovo ordine o l'accettazione di un ordine.

Entrambe le pagine presentano un'interfaccia utente identica, in entrambe, quindi, è presente: il titolo della pagina "Nuovo ordine" o "Accetta un ordine", un elenco puntato riportante la spiegazione di come effettuare l'operazione scelta nel modo corretto, un'area di testo in cui inserire i codici a barre dei prodotti, un bottone per inviare o accettare l'ordine e infine il bottone che permette la cancellazione per intero del contenuto dell'area di testo.

L'inserimento dei codici dei prodotti da ordinare o da accettare è possibile attraverso due vie: l'inserimento manuale o l'inserimento tramite dispositivo esterno:

- L'inserimento manuale: inserimento tramite tastiera del dispositivo su cui viene utilizzata la PWA (tastiera di un computer o la tastiera di uno smartphone piuttosto che quella di un tablet).
- L'inserimento tramite dispositivo esterno: s'intende tramite il collegamento Bluetooth di un lettore di codice a barre al dispositivo che si sta utilizzando. Ogni codice letto con il lettore verrà automaticamente inserito nell'area di testo.

Una volta che tutti i codici sono stati inseriti e che l'ordine è pronto per essere inviato o per essere accettato, si dovrà premere il bottone apposito, "Invia" o "Accetta".

Al premere del bottone verrà creato un file di tipo ".dat" con all'interno i codici riportati nell'area di testo.

Il codice per la creazione del file:

```
1 let token = localStorage.getItem('token').replace(/"/g, '');  
2 const blob = new Blob([this.textContent], { type: 'text/plain' });  
3 const file = new File([blob], 'test.dat', { type: 'text/plain' });  
4 const formData = new FormData();  
5 formData.append('file', file);
```



La creazione di questo file è stata possibile usando il metodo:

```
new Blob([this.textContent], type: 'text/plain' ).
```

Questo metodo crea un "blob" con i dati contenuti nell'area di testo. Blob, letteralmente significa "Binary Large Object", è quindi un oggetto simile a un file di dati grezzi e immutabili.<sup>[3]</sup>

All'interno di `this.textContent` c'è il testo scritto nell'area di testo.

`type: 'text/plain'`, specifica il tipo di MIME (Multipurpose Internet Mail Extensions) che altro non è un'etichetta che permette al browser o al server di assicurarsi della tipologia di file che sta leggendo, in modo da interpretarlo correttamente.

Dopo aver creato un blob di dati si crea il file `.dat` tramite il metodo:

```
new File([blob], 'test.dat', type: 'text/plain' )
```

`blob` viene passato come array perchè `File()` come primo parametro in ingresso richiede un'array di parti di dati (è possibile passargli anche stringe o `ArrayBuffer`), il secondo parametro in ingresso è il nome che si vuole assegnare al file creato (in questo caso, `'test.dat'`) e infine vengono passate le opzioni aggiuntive, in questo caso si ripete la tipologia di MIME ( `type: 'text/plain'` ).

Infine `formData()` crea un oggetto usato per inviare dati tramite post, che in questo caso è il metodo utilizzato per la chiamata API, e aggiunge il file rinominandolo con `'file'`.

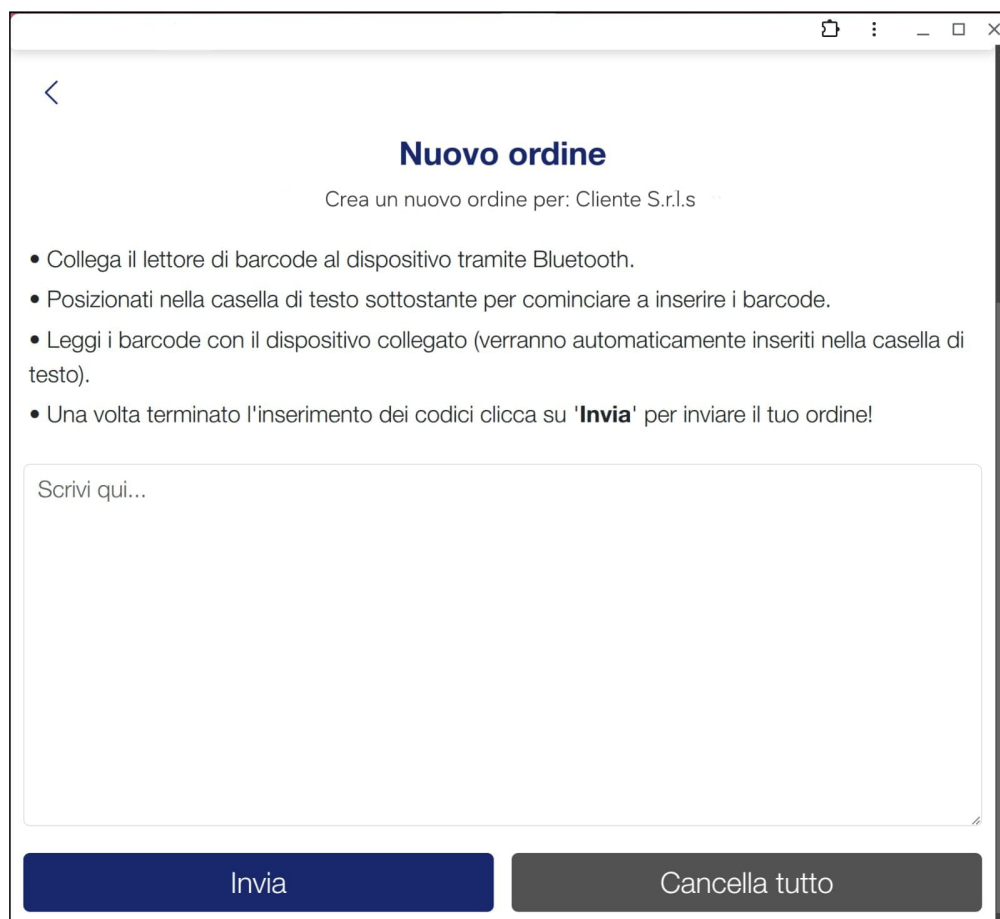
A questo punto, il file è stato creato con successo ed è possibile chiamare l'API che si occuperà dell'invio o dell'accettazione dell'ordine.

L'API per funzionare richiede l'ID del cliente, il token dell'utente e ovviamente, il `formData` appena creato contenente i dati dell'ordine.

**Risposta positiva** In caso di risposta positiva, l'ordine viene inviato o accettato, l'area di testo viene svuotata.

**Risposta negativa** In caso di risposta negativa, l'ordine non viene né inviato né accettato, l'area di testo non viene svuotata.

In entrambi i casi, appare un modale, il suo funzionamento verrà spiegato nella sezione successiva.



<

## Nuovo ordine

Crea un nuovo ordine per: Cliente S.r.l.s

- Collega il lettore di barcode al dispositivo tramite Bluetooth.
- Posizionati nella casella di testo sottostante per cominciare a inserire i barcode.
- Leggi i barcode con il dispositivo collegato (verranno automaticamente inseriti nella casella di testo).
- Una volta terminato l'inserimento dei codici clicca su '**Invia**' per inviare il tuo ordine!

Scrivi qui...

Invia

Cancella tutto

Figura 3.6: La pagina per effettuare un nuovo ordine

Concludendo la spiegazione di questa parte della PWA è necessario ricordare che l'immagine inserita è solo una perchè la pagina di accettazione di un ordine presenta un'interfaccia utente identica a quella mostrata per la creazione di nuovo ordine, questo permette di mantenere continuità e coerenza nell'insieme dell'applicazione.

## 3.8 Modale

Il modale che viene visualizzato al termine di una delle due operazioni è stato implementato tramite *Bootstrap*.

Per la sua creazione è stato creato un componente, *Modal.vue*, al suo interno è presente il codice *JavaScript* e il codice *HTML*, il quale viene riportato per intero:

```
1 <div class="modal fade" id="modalId" data-bs-backdrop="static"
  data-bs-keyboard="false" tabindex="-1" aria-hidden="true" ref="
  modalElement">
2 <div class="modal-dialog">
3   <div class="modal-content">
4     <div class="modal-header">
5       <h5 class="modal-title">{{ title }}</h5>
6       <button type="button" class="btn-close" data-bs-dismiss
          ="modal" aria-label="Close"></button>
7     </div>
8     <div class="modal-body">
9       <slot></slot> //permette al componente genitore di
          inserire contenuto
10    </div>
11    <div class="modal-footer">
12      <button type="button" class="btn btn-secondary" data-bs-
          dismiss="modal">Close</button>
13    </div>
14  </div>
15 </div>
16 </div>
```

*Modal.vue* crea quindi un modale tramite il sistema di modali di *Bootstrap 5*. Lo scopo è quello di mostrare contenuti personalizzati all'interno di una finestra, con un titolo e un pulsante di chiusura.

Creando un componente a parte è possibile permettere ad altri componenti dell'applicazione di utilizzare il modale senza dover riscrivere ogni volta la struttura.

Due attributi particolari sono stati aggiunti al modale:

`data-bs-backdrop="static"`

impedisce la chiusura del modale cliccando al di fuori della finestra che si crea, mentre

```
data-bs-keyboard="false"
```

impedisce la chiusura tramite il tasto "Esc" presente sulle tastiere dei computer.

Tornando brevemente al discorso affrontato nella sezione precedente, qualora la risposta all'invio o all'accettazione di un ordine fosse positiva verrebbe aperto il modale contenente un messaggio di successo nell'operazione.

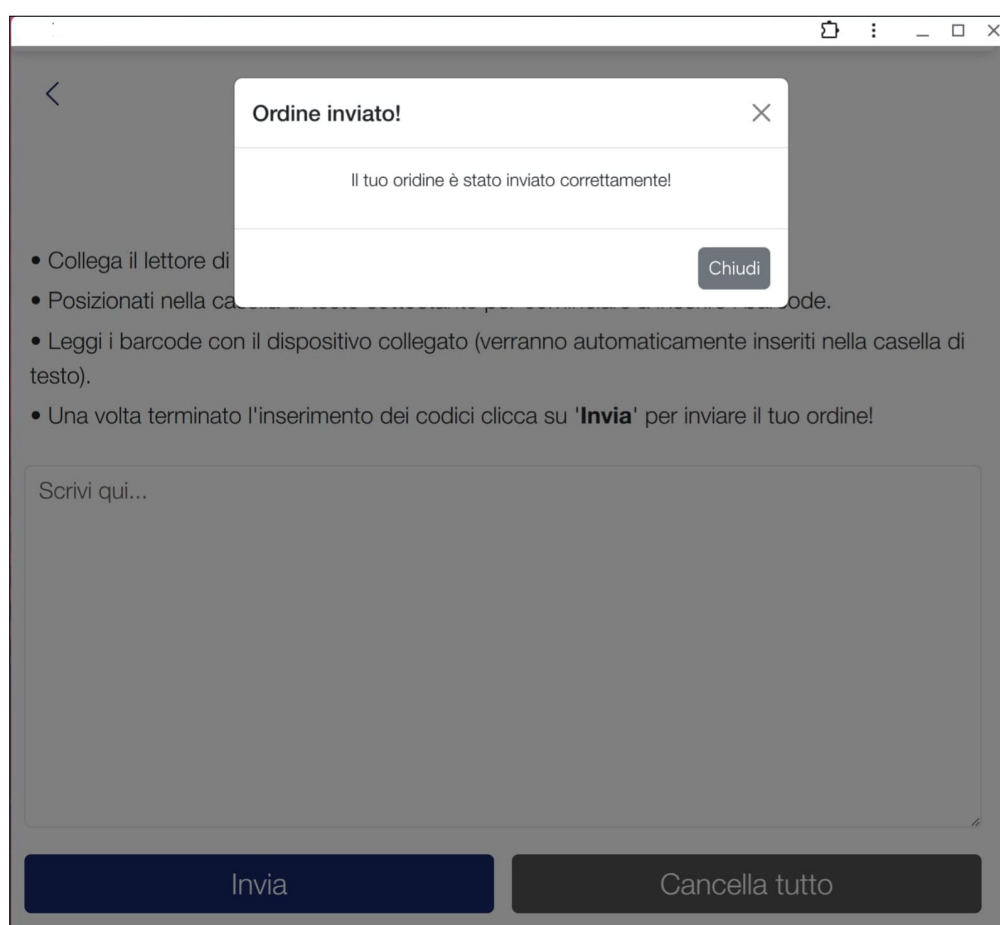


Figura 3.7: Modale di successo

In caso contrario, quindi nel caso in cui l'operazione purtroppo non abbia avuto successo, ma anzi sia fallita, il modale si aprirà e mostrerà un messaggio di errore.

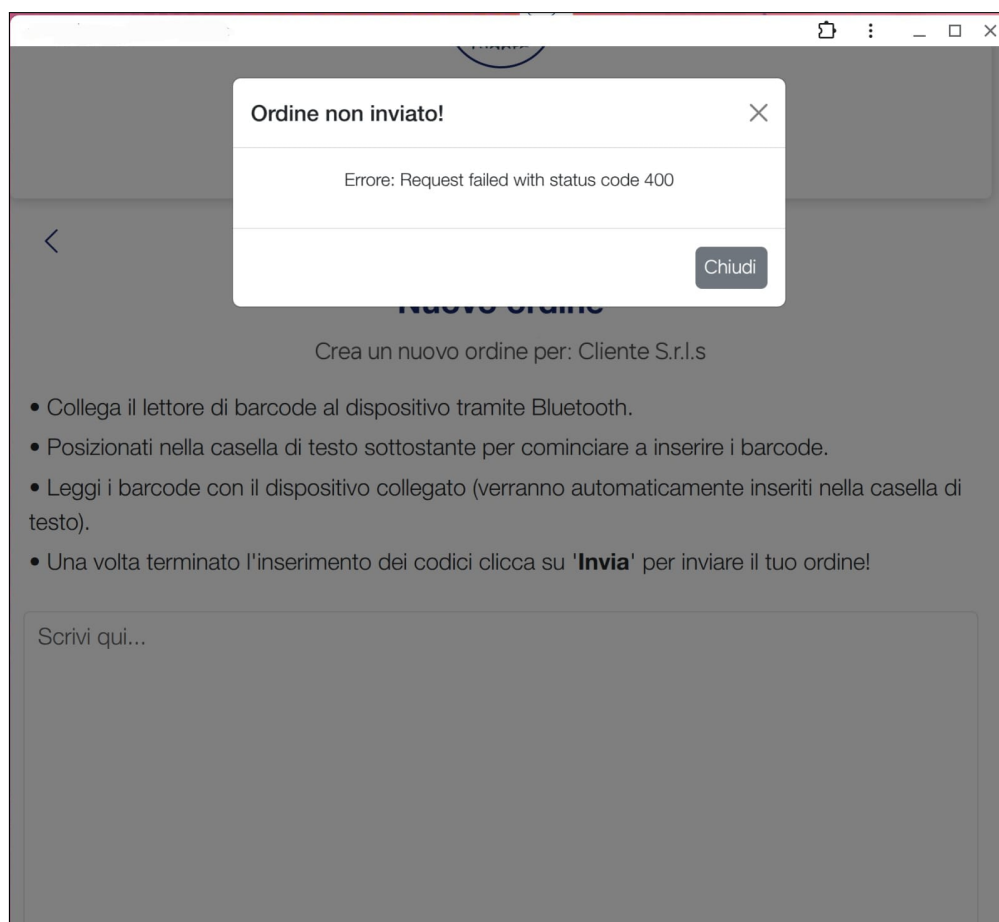


Figura 3.8: Modale di errore



---

## *Risultati e conclusione*

---

Giunti al termine dell'elaborato è indispensabile concludere analizzando i risultati ottenuti. Una volta completata, la *Progressive Web App* si presenta come un'applicazione intuitiva, veloce, reattiva, funzionante, pronta all'uso e, una volta installata, del tutto simile a un'applicazione nativa.

In futuro, la PWA potrà essere facilmente adattata a diverse lingue, senza la necessità di ristrutturare l'intero sistema, poiché il sistema consente l'utilizzo di sistemi di traduzione.

Sarà sufficiente modificare i testi visibili agli utenti, rendendo l'espansione a nuove lingue rapida e semplice.

Il codice scritto in lingua inglese e la veloce modifica dei testi presenti in italiano permetterà di aprire la strada a un'applicazione globale, facilmente accessibile da utenti di diverse nazionalità.





---

## Sitografia

---

- [1] Google Chrome. *Chrome DevTools*. 16 febbraio 2025. URL: <https://developer.chrome.com/docs/devtools/overview?hl=it>.
- [2] Red Hat. *Cosa sono le API (Application Programming Interface)*. 16 febbraio 2025. URL: <https://www.redhat.com/it/topics/api/what-are-application-programming-interfaces#:~:text=Il%20termine%20API%2C%20acronimo%20di,l'integrazione%20di%20applicazioni%20software..>
- [3] MDN. *Blob*. 18 febbraio 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Blob>.
- [4] MDN. *CSS: Cascading Style Sheets*. 02 febbraio 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [5] MDN. *HTML: HyperText Markup Language*. 02 febbraio 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [6] MDN. *JavaScript*. 16 febbraio 2025. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [7] MDN. *Progressive Web Apps*. 26 gennaio 2025. URL: [https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps).
- [8] MDN. *ServiceWorker*. 26 gennaio 2025. URL: [https://developer.mozilla.org/en-US/docs/Web/API/Service\\_Worker\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API).



---

## *Elenco delle figure*

---

2.1	Risultato utilizzo classi . . . . .	13
3.1	Pagina di login . . . . .	24
3.2	La scelta del cliente . . . . .	26
3.3	Il menù espanso e la profile card . . . . .	29
3.4	Pagina Home 1 . . . . .	31
3.5	Pagina Home 2 . . . . .	31
3.6	La pagina per effettuare un nuovo ordine . . . . .	34
3.7	Modale di successo . . . . .	36
3.8	Modale di errore . . . . .	37