UNIVERSITÀ
DI PARMA

Relazione progetto tecnologie internet

Corso a cura del professore Michele Amoretti

Ingegneria informatica, elettronica e delle telecomunicazioni 2021/2022

Progetto realizzato da Cristian Cervellera 305459 e Stefano Ruggiero 305880

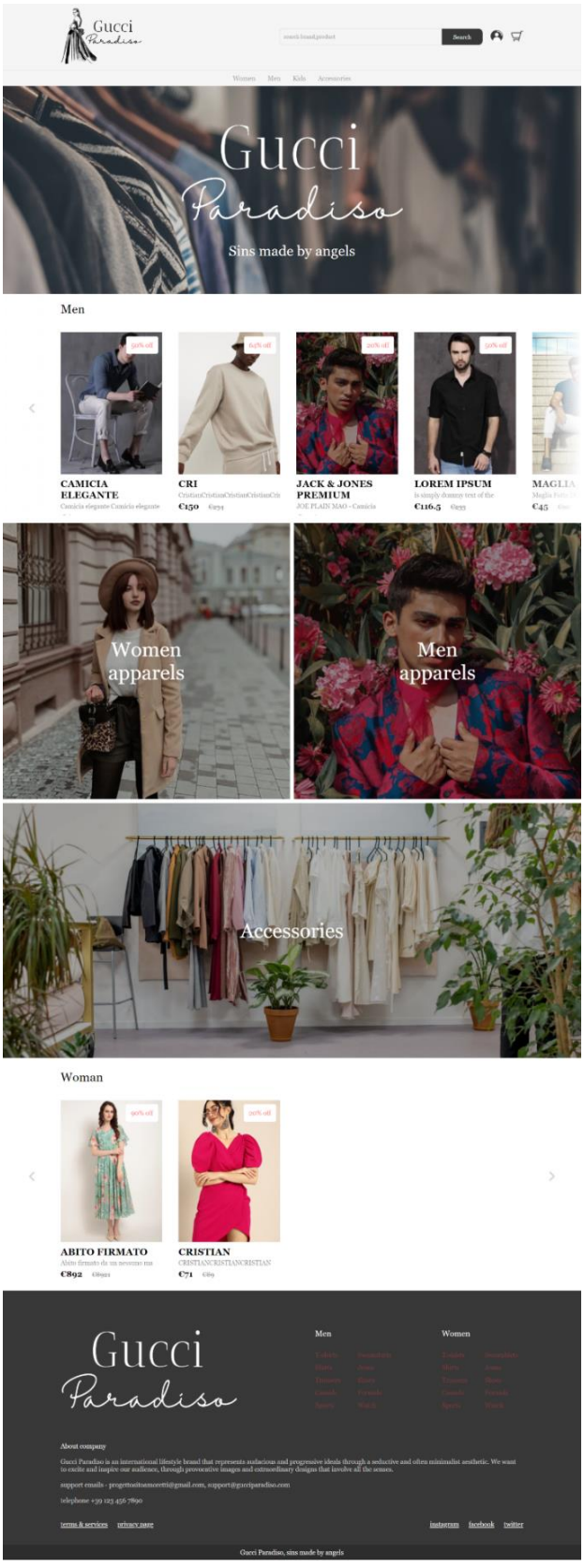
Introduzione

Il progetto realizzato è un e-commerce di prodotti del settore moda, con la possibilità sia di acquistarli che di metterli in vendita. Il nome scelto è “Gucci Paradiso”.

L’home della piattaforma si presenta in questo modo.

Da qui l’utente può registrarsi, ricercare prodotti, acquistarli e diventare un venditore per aggiungere i propri prodotti da mettere in vendita

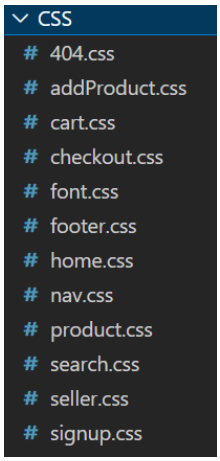
L' e-commerce si basa su un’architettura client-server, il client ha il compito di gestire gli input dell’utente, mostrare tutti i prodotti, acquistarli e aggiungerli. Ogni client scambia i messaggi con un server che gestisce i dati della singola sessione, fa richieste al database di dati e li fornisce al client.

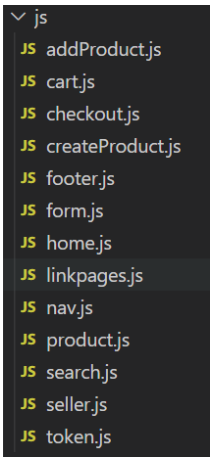


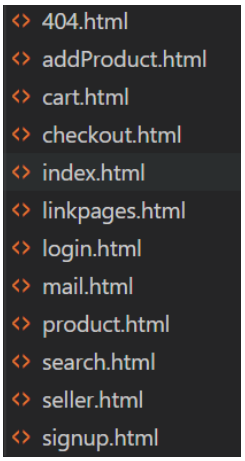
Tecnologie utilizzate

I linguaggi utilizzati per la realizzazione del progetto sono: HTML, JS e CSS. Per la gestione dei dati è stata utilizzata la piattaforma Firebase, mentre per la gestione delle immagini è stato utilizzato AWS.

- **HTML**: È un linguaggio di markup, utilizzato principalmente per il disaccoppiamento della struttura logica di una pagina web (definita appunto dal markup) e la sua rappresentazione, gestita tramite gli stili CSS.
- **CSS**: Linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML. Quindi viene utilizzato principalmente per definire la struttura grafica delle pagine html.
- **JavaScript**: È un linguaggio di programmazione multi-paradigma orientato agli eventi, comunemente utilizzato nella programmazione Web lato client (esteso poi anche al lato server con Node.js) per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso.
- **Firebase**: è un database NoSQL ospitato nel cloud che consente di archiviare e sincronizzare i dati tra gli utenti in tempo reale.
- **AWS**: Amazon Simple Storage Service è un servizio di storage di oggetti che offre disponibilità di archiviare ed ottenere immagini.



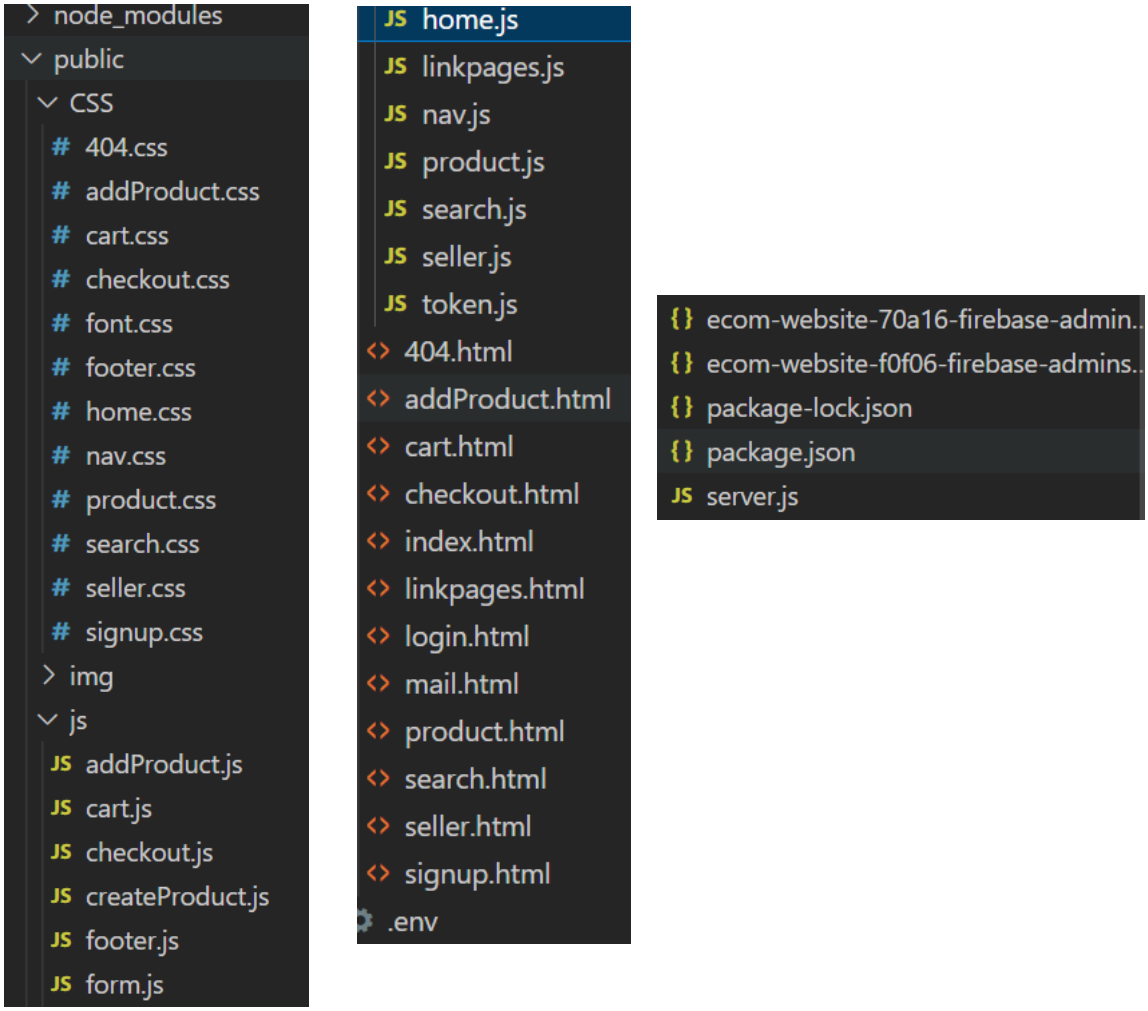




Inoltre, abbiamo utilizzato le seguenti librerie:

- **Node.js e Express**: per la realizzazione del server che gestisce tutta la logica della piattaforma, express è un framework che permette il rendering degli static-file lato client.
- **dotenv**: per la connessione al database AWS.
- **bcrypt**: per criptare tramite hash le password inserite dagli utenti e confrontare la password nel momento del login.
- **aws-sdk**: per usare AWS in Javascript con Nodejs
- **nodemailer**: modulo per le applicazioni Node.js per consentire invio di e-mail.

Struttura progetto



Il file server.js inizializza il client ed il server chiamando i file index.html (root pages) delle rispettive cartelle, inoltre stabilisce la connessione con il database Firebase e con AWS.
Nella cartella public ci sono: tutti i file html dell'e-commerce, a partire dalla home page (index.html) e ci sono anche altre 3 cartelle:

- img: cartella che contiene le immagini presenti nel sito iniziale, utilizzate per i popup e per l'intestazione
- CSS: contiene tutti i file .css del sito, utilizzati per comporre la grafica delle pagine html.
- js: contiene tutti i file .js utili per rendere dinamico il sito, ognuno di loro contiene le varie funzioni che possono essere invocate dall'utente sulle rispettive pagine. (funzionalità spiegate nel dettaglio successivamente)

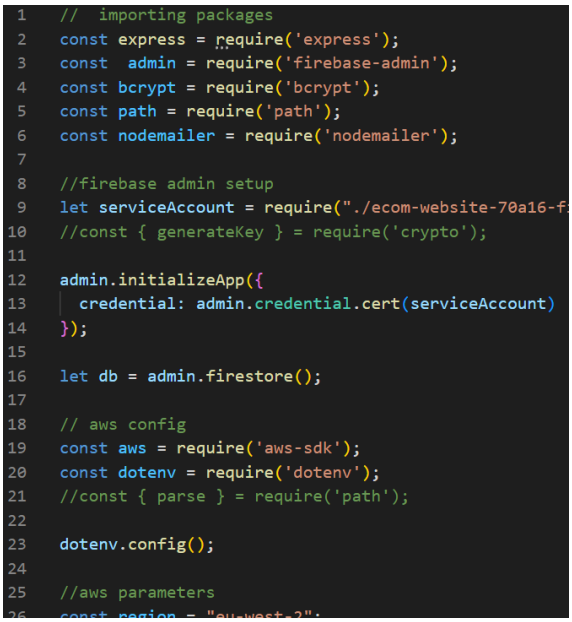
Nella cartella public è contenuto:

- il file principale server.js utile per la creazione di una sessione per ogni utente che vi accede, l'indirizzamento delle pagine ed il richiamo delle funzioni in base alle richieste da parte del client.
- un file .env contenente tutti i dati di configurazione utili per l'accesso ad AWS e per l'invio delle mail da parte del sito

- il file ecom-website contiene tutti i dati per la configurazione di Firebase
- package.json è un file di configurazione che contiene i dati utili alla configurazione del server e l'indirizzamento alla pagina iniziale index.html

Dettagli sul codice

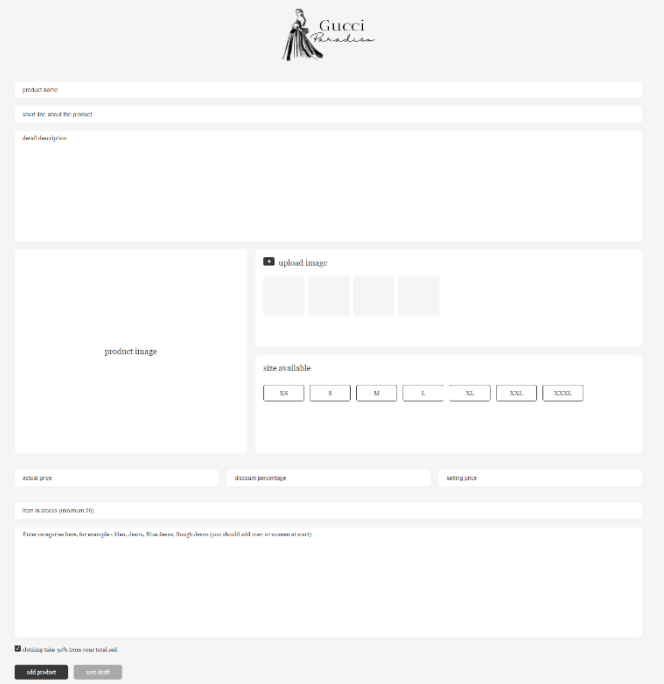
server.js: gestisce il server, importa tutti i pacchetti necessari, configura aws e firebase, gestisce l'indirizzamento delle pagine, del database e del cloud storage e l'invio delle e-mail di riepilogo dell'ordine effettuato.



```
//  importing packages
const express = require('express');
const  admin = require('firebase-admin');
const bcrypt = require('bcrypt');
const path = require('path');
const nodemailer = require('nodemailer');

//firebase admin setup
let serviceAccount = require("./ecom-website-70a16-f
//const { generateKey } = require('crypto');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

let db = admin.firestore();

// aws config
const aws = require('aws-sdk');
const dotenv = require('dotenv');
//const { parse } = require('path');

dotenv.config();

//aws parameters
const region = "eu-west-2";
```

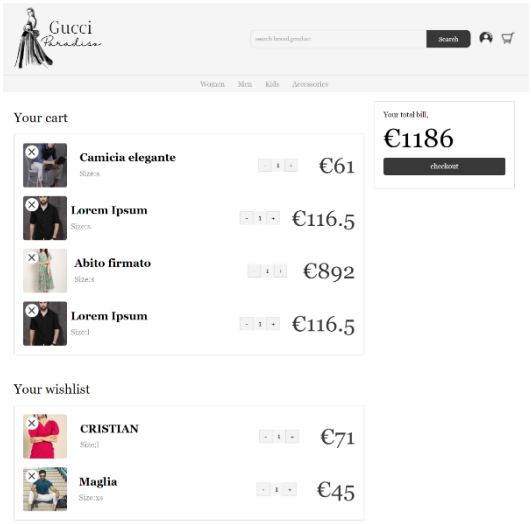
```
//place an order and sand email
app.post('/order' , (req, res) => {
    const {order, email, add } = req.body;

    let data=JSON.parse(order);
    //console.log(data);
    let start = '<h2 class="product-category">Your order:</h2> <ul> ';
    let middle = ''; // this will contain the products ordered
    let end = '</ul>';
    for(let i= 0; i < data.length; i++) {
        middle +=  `<li> ${data[i].item} ${data[i].name} taglia ${data[i].size} </li>`
    }
    let list_html= start+middle+end;


    let transporter = nodemailer.createTransport({
        service: 'gmail',
        auth: {
            user: process.env.EMAIL,
            pass: process.env.PASSWORD
        }
    })

    const mailOption ={
        from: 'progettositoamoretti@gmail.com' ,
        to: email,
```

addProduct.js: file javascript che gestisce il funzionamento della pagina html (addProduct.html) utilizzata nel momento in cui un utente Seller decide di aggiungere un proprio prodotto in vendita. Questo file js gestisce tutti i vari controlli legati ai campi di input presenti (Validate form) nella pagina html e infine permette l'inserimento del prodotto all'interno del database (Firebase), per renderlo univoco rispetto agli altri prodotti già presenti è stato aggiunto un id (nome prodotto + numero casuale) al nome del prodotto in modo tale che vi possono essere due prodotti con lo stesso nome ma completamente diversi. Una funzionalità molto utile di questo file è quella legata al calcolo del prezzo di sconto e della percentuale, inserendo il prezzo effettivo e quello in sconto, la pagina caricherà dinamicamente il valore della percentuale. L'inserimento del prodotto comprende anche l'inserimento delle immagini ad esso legato infatti questo file js permette, oltre a far scegliere all'utente delle immagini, di caricare le immagini su AWS e quindi rendere il prodotto disponibile sull'e-commerce a tutti gli utenti. Questo file js gestisce anche la possibilità della modifica di un prodotto già inserito, modificando uno dei campi presenti all'interno del form addProduct.html.



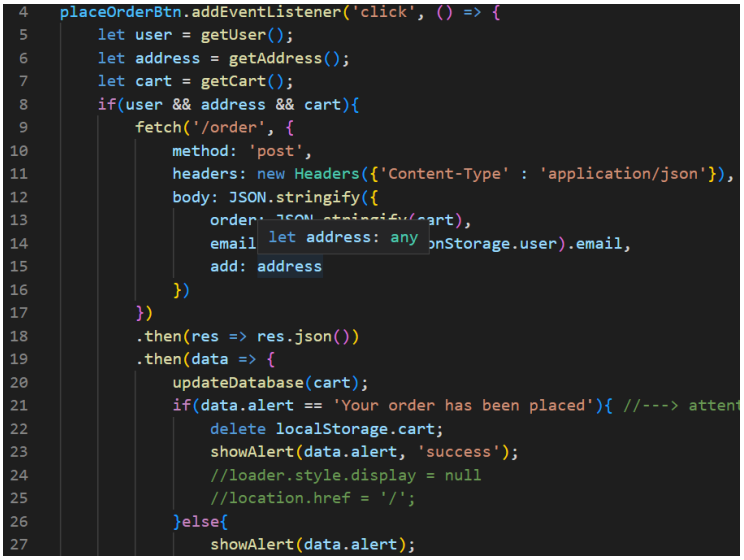

```
146  addProductBtn.addEventListener('click',() => {
147      storeSizes();
148      // validate form
149      if(validateForm()){  //validateForm return true or false while doing validation
150          loader.style.display = 'block';
151          let data = productData();
152          if(!data.id){
153              data.id = data.name.toLowerCase()+'-'+Math.floor(Math.random() * 5000);
154          }
155          sendData('/add-product', data);
156      }
157  })
```

cart.js: gestisce il funzionamento della pagina cart.html. In questa pagina è presente il carrello con tutti i prodotti che l'utente vorrebbe o è in procinto di acquistare, con al a fianco il totale della spesa nel carrello.




```
46  const setupEvents = (name) => {
47      // setup counter event
48      const counterMinus = document.querySelectorAll( `.${name} .decrement`);
49      const counterPlus = document.querySelectorAll( `.${name} .increment`);
50      const counts = document.querySelectorAll( `.${name} .item-count`);
51      const price = document.querySelectorAll( `.${name} .sm-price`);
52      const deleteBtn = document.querySelectorAll( `.${name} .sm-delete-btn`);
53
54      let product = JSON.parse(localStorage.getItem(name));
55
56      counts.forEach((item, i) => {
57          let cost = Number(price[i].getAttribute('data-price'));
58          counterMinus[i].addEventListener( 'click' , () => {
59              if(item.innerHTML > 1){
60                  item.innerHTML--;
61                  totalBill -= cost;
62                  price[i].innerHTML = `€${item.innerHTML * cost}`;
63                  if(name == 'cart' ){
64                      updateBill()
65                  }
66                  product[i].item = item.innerHTML;
```

checkout.js: file javascript che gestisce il funzionamento della pagina html (checkout.html) caricata nel momento in cui un utente decide di terminare l'acquisto di un prodotto presente nel carrello. Questo file, quindi, controlla che tutti i campi relativi ad una possibile spedizione del pacco siano compilati, se il prodotto sia stato selezionato in modo corretto e l'utente sia registrato. Gestisce il caricamento in modo dinamico della pagina e quindi il caricamento dei dati legati all'ordine dell'utente loggato. La parte più importante di questo file è l'aggiornamento del database con la diminuzione della quantità del prodotto e l'invio della e-mail con il riepilogo dell'ordine fatto.



createProducts.js: questo file invece contiene lo script necessario per la creazione della carta prodotto nella sezione dei prodotti in vendita dell'utente seller.

home.js: questo file gestisce diversi aspetti utili in molteplici ambiti. Infatti, viene utilizzato per creare e popolare sezioni di prodotti in slider e tabella, ottenere prodotti dal database in base al tag, gestione del carrello con i prodotti selezionati dall'utente e la ricerca di prodotti con tag multipli.

```
22   // fetch product cards
23   const getProducts = (tag) => {
24       tag=tag.toLowerCase();
25       return fetch('/get-products', {
26           method: "post",
27           headers: new Headers({"Content-Type":"application/json"}),
28           body: JSON.stringify({tag: tag})
29       })
30       .then(res => res.json())
31       .then(data => {
32           return data;
33       })
34   }
```

```
118 const add_product_to_cart_or_wishlist = (type, product) => {
119     let data = JSON.parse(localStorage.getItem(type));
120     if(data == null){
121         data = [];
122     }
123     product = {
124         item: 1,
125         id: product.id,
126         stock: product.stock,
127         name: product.name,
128         sellPrice: product.sellPrice,
129         size: size || null,
130         shortDes: product.shortDes,
131         image: product.images[0]
132     }
133     if(product.size != null ){
134         if (count_stock(data,product)+1 <= product.stock) { //check if you car
135             for(let i= 0; i < data.length; i ++){
136                 if (data[i].id == product.id && data[i].size == product.size)
137                 {
138                     let count_size = Number(data[i].item)+1;
139                     product.item=count_size;
140                     data.splice(i,1); // delete the element in position i
```

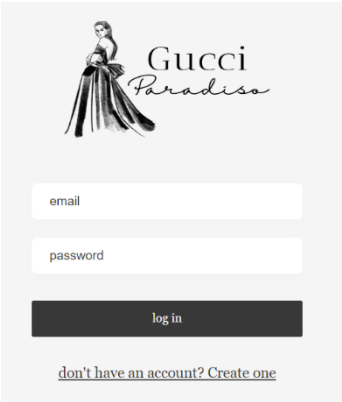
```
173   // search with multiples keys
174   const crossSearch = (list_keys) => {
175       getProducts(list_keys[0]).then(data => crossRecursive(data,list_keys));
176   }
177
178   const crossRecursive = (data,list_keys) => {
179       if (data != 'no products' ){
180           list_keys.forEach((key,k) => {
181               data=createCross(data,key); //substitutes the old data with the key f
182           })
183           data=normalizeData(data);
184           createProductCards(data, '.card-container');
185           suggestionRecursive(data,list_keys);
186       }
187       else{
188           let prova=[]
189           suggestionRecursive(prova,list_keys);
190       }
191
192   }
```

footer.js: file javascript che gestisce il caricamento del footer in ogni pagina del sito, esso permette quindi la creazione del footer all'interno della pagina in cui viene richiamato e contiene link per l'indirizzamento a pagine precise che viene gestito dal file linkpages.js.


```
const createFooter = () =>{
    let footer = document.querySelector('footer');
    footer.innerHTML = `
            <div class="footer-content">
                <img src="../img/light-logo.png" class="logo" alt="">
                <div class="footer-ul-container">
                    <ul class="category">
                        <li class="category-title">Men</li>
                        <li><a href="../linkpages.html?id=Men,T-shirts" class="footer-link">T-shirts</a></li>
                        <li><a href="../linkpages.html?id=Men,Sweatshirts" class="footer-link">Sweatshirts</a></li>
                        <li><a href="../linkpages.html?id=Men,Shirts" class="footer-link">Shirts</a></li>
                        <li><a href="../linkpages.html?id=Men,Jeans" class="footer-link">Jeans</a></li>
```

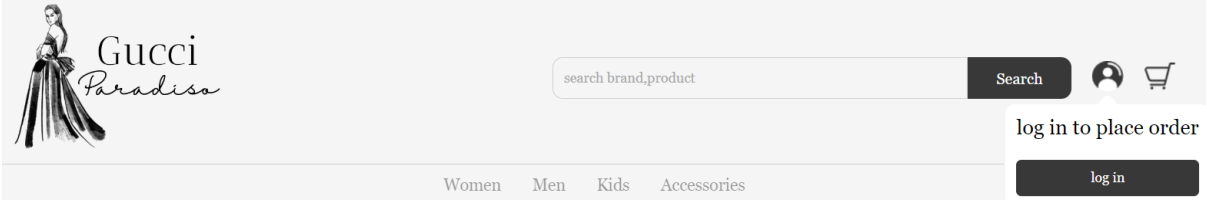
linkpages.js: questo file gestisce il caricamento dinamico della pagina in base al link selezionato all'interno del footer e della navbar. In questo file la parte più importante è nella chiamata della funzione crossSearch contenuta in home.js per la ricerca di tag multipli.

form.js: file javascript che gestisce il funzionamento della pagina html (login.html) caricata nel momento in cui un utente decide di fare il login. Gestisce i controlli legati alla mail e alla password controllando se esse sono contenute all'interno di Firebase. Nel caso in cui avviene il login con successo esso caricherà la pagina antecedente al richiamo del login con le credenziali legate all'utente. Se l'utente non è registrato e preme il link per la registrazione, questo file caricherà la pagina legata ad essa (signup.html).




```
        }else if(!tac.checked){
            showAlert('you must agree to our terms and conditions');
        }else{
            //submit form
            loader.style.display = 'block';
            sendData('/signup', {
                name: name.value,
                email: email.value,
                password: password.value,
                number: number.value,
                tac: tac.checked,
                notification: notification.checked,
                seller: false
            })
        }
    }
    else{
        //login page
        if(!email.value.length || !password.value.length){
            showAlert('fill all the inputs');
        } else{
            loader.style.display = 'block';
            sendData('/login', {
                email: email.value,
                password: password.value,
```

nav.js: file javascript che gestisce il caricamento della navbar in ogni pagina e il funzionamento di essa. Permette quindi il caricamento della pagina dei prodotti in base a ciò che è stato inserito all'interno del campo di ricerca, questa ricerca può comprendere più parole. Una parte molto importante di questo file è la gestione della ricerca dove oltre ad inserire i prodotti che contengono tutti i dati inseriti nella ricerca, carica anche prodotti simili alla ricerca fatta in modo da poter fornire una lista di prodotti esaustiva alla ricerca dell'utente; per fare ciò viene richiamata la stessa funzione utilizzata in linkpages.js ossia crossSearch. Le parole inserite nella ricerca vengono confrontate con i tag presenti all'interno di ogni prodotto all'interno del database (Firebase).

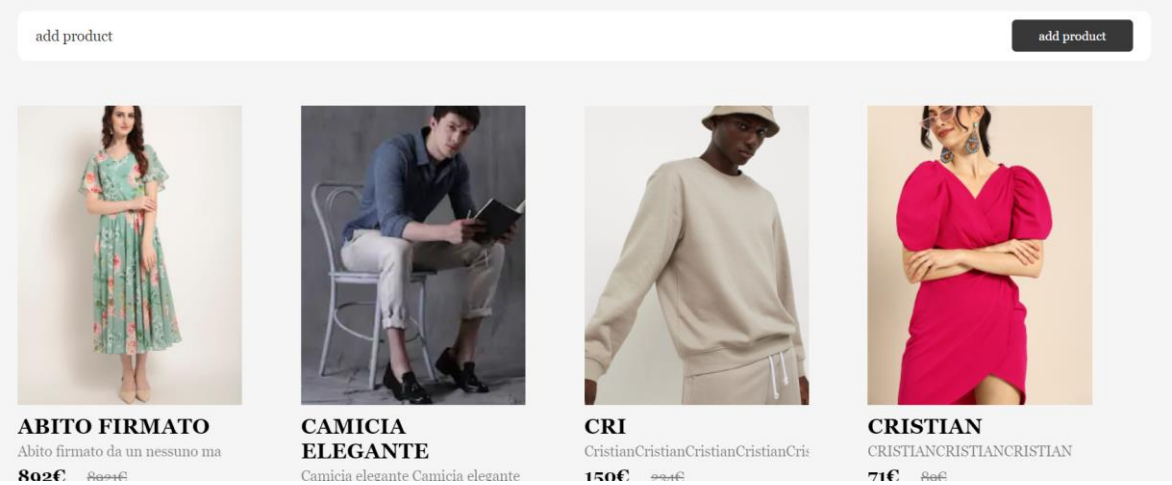



```
window.onload = () => {
    let user = JSON.parse(sessionStorage.user || null);
    if(user != null){
        // means user is logged in
        popuptext.innerHTML = `log in as, ${user.name}`;
        areaBtn.removeAttribute("hidden");
        areaBtn.innerHTML = `Selling area <br></br><a href="/seller">
        actionBtn.innerHTML = 'log out';
        actionBtn.addEventListener('click', () =>{
            sessionStorage.clear();
            localStorage.clear();
            location.reload();
        })
    } else{
        //user is logged out
        popuptext.innerHTML = 'log in to place order';
        actionBtn.innerHTML = 'log in';
        actionBtn.addEventListener('click', () =>{
            location.href = '/login';
        })
    }
}
```

search.js: file javascript utilizzato nel momento in cui viene fatta una qualsiasi ricerca che può avvenire o dalla navbar o dai link del footer. Gestisce le due ricerche: quella singola dove l'utente ha inserito una sola parola all'interno della ricerca e quella multipla con l'inserimento di più parole (crossSearch).

product.js: gestisce la creazione dinamica della pagina product.html del singolo prodotto selezionato, con tutte le caratteristiche e foto caricate dal seller. Quindi gestisce le richieste per Firebase. È possibile, inoltre, inserire i prodotti nel carrello o nella wishlist, controllando se è stata selezionata una taglia.

seller.js: file javascript che gestisce il funzionamento della pagina html (seller.html) caricata nel momento in cui un utente decide di andare nella propria selling-area. Questa pagina permette due funzioni principali: la prima consiste nella registrazione dell'utente come seller quindi avviene il controllo sui campi inseriti come seller e quindi l'update della tabella degli user nella riga dell'utente loggato (lo rende oltre che user anche seller); la seconda funzione principale viene richiamata nel momento in cui l'user è già un seller e quindi avviene il caricamento dinamico della pagina con tutti i prodotti aggiunti dal seller in questione con la possibilità di poterli modificare o eliminare.



token.js: file javascript utilizzato nel momento in cui un utente si vuole registrare viene generato un token in base alla e-mail, questo token viene utilizzato nel momento in cui si vuole verificare che l'utente è effettivamente lui; infatti, poi nel metodo dove si compara l'algoritmo esso prende in input l'e-mail ed il token e verifica se è effettivamente lui. Questo file gestisce l'invio dei popup come alert nelle varie pagine, il caricamento dei dati e la processazione di essi nel momento in cui si vuole interagire con il database (Firebase) o con AWS.