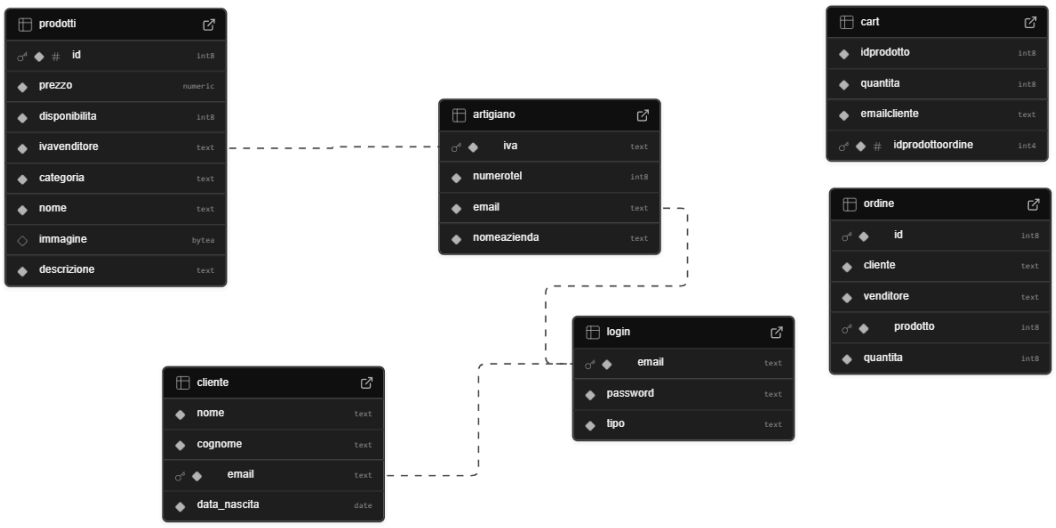
Manuale Tecnico Web

Diagrammi E-R Database

Per lo sviluppo del database ci siamo affidati a Supabase che, grazie al collegamento diretto con pgAdmin, ci ha permesso di lavorare su un ambiente condiviso basato su un database relazionale SQL.



Descrizione delle tabelle

1. Tabella `prodotti`

Contiene le informazioni sui prodotti in vendita:

- `id`: identificativo univoco del prodotto (**chiave primaria**)
- `prezzo`: prezzo del prodotto
- `disponibilità`: quantità disponibile in magazzino
- `ivavenditore`: riferimento all'artigiano che vende il prodotto (**foreign key** verso `artigiano.iva`)
- `categoria`: categoria del prodotto (es. abbigliamento, ceramica, ecc.)
- `nome`: nome del prodotto
- `immagine`: immagine del prodotto (tipo `bytea`, quindi in formato binario)
- `descrizione`: descrizione testuale del prodotto

---

2. Tabella `artigiano`

Contiene i dati degli artigiani/venditori:

- `iva`: partita IVA (**chiave primaria**)
- `numerotel`: numero di telefono
- `email`: indirizzo email (**foreign key** verso `login.email`)
- `nomeazienda`: nome dell'azienda dell'artigiano

---

3. Tabella `login`

Gestisce l'autenticazione degli utenti (sia clienti che artigiani):

- `email`: identificativo dell'utente (**chiave primaria**)
- `password`: password associata
- `tipo`: tipo di utente (`cliente` o `artigiano`)

---

4. Tabella `cliente`

Contiene le informazioni personali dei clienti:

- `nome`: nome del cliente
- `cognome`: cognome del cliente
- `email`: identificativo univoco (**chiave primaria**, **foreign key** verso `login.email`)
- `data_nascita`: data di nascita del cliente

---

5. Tabella `cart` (carrello)

Gestisce il contenuto del carrello per ogni cliente:

- `idprodottoordine`: identificativo univoco della voce nel carrello (**chiave primaria**)

- `idprodotto`: identificativo del prodotto aggiunto al carrello (**foreign key** verso `prodotti.id`)
- `quantita`: quantità del prodotto nel carrello
- `emailcliente`: riferimento al cliente (**foreign key** verso `cliente.email`)

---

6. Tabella `ordine`

Memorizza gli ordini effettuati:

- `id`: identificativo univoco dell'ordine (**chiave primaria**)
- `cliente`: riferimento al cliente che ha effettuato l'ordine (**foreign key** verso `cliente.email`)
- `venditore`: riferimento all'artigiano venditore (**foreign key** verso `artigiano.iva`)
- `prodotto`: identificativo del prodotto acquistato (**foreign key** verso `prodotti.id`)
- `quantita`: quantità acquistata del prodotto

Panoramica del Sistema

Il sistema è un'applicazione e-commerce sviluppata con **Node.js** e **Express.js** che permette a:

- **Clienti**: registrarsi, navigare prodotti, gestire carrello, effettuare ordini
- **Artigiani**: registrarsi, gestire prodotti, visualizzare ordini ricevuti

Tecnologie Utilizzate

- **Backend**: Node.js, Express.js
- **FrontEnd**:Html,CSS external e inline,javascript
- **Database**: PostgreSQL con appoggio a supabase per collegamento in cloud
- **Autenticazione**: Sistema custom con tabella login e registrazione

---

Architettura

Componenti Principali

1. **Server Express**: Gestisce routing e middleware
2. **Client PostgreSQL**: Connessione al database
3. **Multer**: Gestione upload immagini

4. **Funzioni di Utilità**: Query database incapsulate

Configurazione Database

Connessione

```
const client = new Client({
  user: 'postgres.ajexsiyipavyjrkseedr',
  host: 'aws-0-eu-central-1.pooler.supabase.com',
  database: 'postgres',
  password: 'Sviluppo2025',
  port: 6543,
});
```

Autenticazione

```
app.post('/loginCliente', async (req, res) => {
  const { email, password } = req.body;
  try {
        // Verifica se esiste un login valido
        const result = await client.query(
        'SELECT * FROM login WHERE email = $1 AND password = $2',
        [email, password]
        );
```

```
        if (result.rowCount > 0) {
        return res.json({ success: true });
        } else {
        return res.json({ success: false, message: 'Credenziali non valide' });
        }
  } catch (err) {
    console.error('Errore loginCliente:', err);
        return res.json({ error: 'Errore del server' });
  }
});
```

Questo codice è un esempio della gestione col database, negli altri metodi cambia l'uso delle query:

```
app.post('/loginCliente', async (req, res) => {
```

Definisce una rotta POST all'endpoint /loginCliente. Quando viene fatta una richiesta a questo URL, questa funzione viene eseguita.

```
  const { email, password } = req.body;
```

Estrae email e password dal corpo della richiesta (inviati dal frontend, ad esempio tramite una form HTML o una chiamata fetch o axios).

```
  try {
```

Inizia un blocco try...catch per gestire eventuali errori nell'accesso al database.

```
const result = await client.query(
        'SELECT * FROM login WHERE email = $1 AND password = $2',
        [email, password]
        );
```

Esegue una query SQL sul database, cercando un record nella tabella login dove l'email e la password combaciano.

Usa $1 e $2 per evitare SQL injection

await fa sì che l'app aspetti il risultato prima di proseguire

```
        if (result.rowCount > 0) {
```

```
        return res.json({ success: true });
        } else {
        return res.json({ success: false, message: 'Credenziali non valide' });
        }
```

Se ha trovato almeno una riga (rowCount > 0), significa che le credenziali sono corrette → restituisce { success: true }.

Altrimenti, restituisce { success: false, message: 'Credenziali non valide' }.

```
  } catch (err) {
        console.error('Errore loginCliente:', err);
        return res.json({ error: 'Errore del server' });
  }
```

Se si verifica un errore nella query (es. il database non risponde), lo stampa in console e invia una risposta con errore generico.

Autentica un cliente.

```
{
  "email": "cliente@email.com",
  "password": "password123"
}
```

Risposta: {success: true/false, message?: string}

POST /loginArtigiano

Autentica un artigiano.

```
{
  "email": "artigiano@email.com",
  "password": "password123",
  "iva": "12345678901"
```

}

POST /registrazioneCliente

Registra un nuovo cliente.

{

"nome": "Mario",

"cognome": "Rossi",

"email": "mario@email.com",

"dataNascita": "1990-01-01",

"password": "password123"

}

POST /artigianiRegistrazione

Registra un nuovo artigiano.

{

"nomeAzienda": "Artigianato Rossi",

"IVA": "12345678901",

"telefono": "1234567890",

"email": "artigiano@email.com",

"password": "password123"

}

Gestione Prodotti

POST /prodotti

Recupera tutti i prodotti con immagini convertite in base64. **Risposta**:

{

"success": true,

"prodotti": [

{

"id": 1,

```
    "nome": "Prodotto",
    "prezzo": 29.99,
    "immagine": "..."
      }
 ]
}
```

POST /ricercaProdottiNome

Cerca prodotti per nome.

```
{
 "nome": "ceramica"
}
```

POST /prodottobyId

Recupera prodotto per ID.

```
{
 "id": 1
}
```

POST /prodottoByCategoria

Recupera prodotti per categoria tramite query SQL.

```
{
 "query": "SELECT * FROM prodotti WHERE categoria = 'ceramica'"
}
```

Gestione Prodotti Artigiano

POST /prodotto/aggiungi

Aggiunge un nuovo prodotto (con upload immagine).

```
// Form-data multipart
{
 categoria: "ceramica",
```

prezzo: 29.99,

disponibilita: 10,

idVenditore: "12345678901",

nome: "Vaso Ceramica",

descrizione: "Bellissimo vaso",

immagine: [File]

}

PUT /prodotto/modifica

Modifica un prodotto esistente.

{

id: 1,

nome: "Nuovo Nome",

prezzo: 35.99,

disponibilita: 15,

descrizione: "Nuova descrizione",

categoria: "ceramica",

immagine?: [File] // opzionale

}

DELETE /prodotto/elimina

Elimina un prodotto.

{

"id": 1

}

Gestione Carrello

POST /aggiungiProdottoCarrello

Aggiunge prodotto al carrello o incrementa quantità.

{

```
  "idProdotto": 1,
  "email": "cliente@email.com"
}
```

POST /cart/modificaQuantita

Modifica quantità prodotto nel carrello.

```
{
  "clienteId": "cliente@email.com",
  "prodottoId": 1,
  "quantita": 3
}
```

POST /cart/rimuoviProdotto

Rimuove prodotto dal carrello.

```
{
  "clienteId": "cliente@email.com",
  "prodottoId": 1
}
```

GET /Cart/idProdotti?cliente=email

Recupera ID prodotti nel carrello del cliente.

POST /cart/svuotaCarrello

Svuota completamente il carrello.

```
{
  "clienteId": "cliente@email.com"
}
```

Sistema Ordini

POST /checkout

Processa il checkout del carrello.

```
{
```

"clienteId": "cliente@email.com"

}

Funzionalità:

1. Verifica disponibilità prodotti
2. Crea record ordine per ogni prodotto
3. Aggiorna disponibilità prodotti
4. Svuota carrello

GET /ordine/cliente?cliente=email

Recupera ID ordini del cliente.

GET /ordine/venditore?ivavenditore=iva

Recupera ordini per artigiano.

GET /ordine/prodotti?cliente=email&ordine=id

Recupera prodotti di un ordine specifico.

Statistiche

GET /stat/ordini?cliente=email

Numero totale ordini cliente.

GET /stat/quantita?cliente=email

Totale prodotti acquistati dal cliente.

GET /stat/spesa?cliente=email

Totale speso dal cliente.

GET /ordine/totale?cliente=email&ordine=id

Totale di un ordine specifico.

Funzioni Principali

getOrdiniCliente(email)

Recupera tutti gli ID ordini di un cliente.

getProdottiArtigiano(artigianoId)

Recupera tutti i prodotti di un artigiano.

getProdotti()

Recupera tutti i prodotti dal database.

getProdottiByNome(nome)

Cerca prodotti per nome (ILIKE pattern matching).

getProdottiCart(clienteId)

Recupera ID prodotti nel carrello del cliente.

Gestione Immagini

Upload

- Immagini salvate come BYTEA nel database
- Support per file singoli tramite upload.single('immagine')

Recupero

- Conversione da BYTEA a base64
- Formato: {base64_string}
- Endpoint dedicato: GET /prodotto/immagine?id=1

Sistema di Autenticazione

Struttura

- Tabella login centrale con email, password, tipo
- Due tipi di utente: cliente e artigiano

Flusso Autenticazione

1. Client invia credenziali
2. Server verifica in tabella login
3. Per artigiani: verifica aggiuntiva in tabella artigiano
4. Risposta success/error

Endpoint Utility

GET /utente/tipo?email=email

Determina se l'utente è cliente o artigiano.

Gestione Carrello

Logica Implementata

1. **Aggiunta Prodotto**: Se esiste, incrementa quantità; altrimenti, inserisce nuovo record
2. **Modifica Quantità**: UPDATE diretto della quantità
3. **Rimozione**: DELETE del record
4. **Checkout**: Verifica disponibilità, crea ordini, aggiorna stock

Validazioni

- Controllo disponibilità durante checkout
- Verifica esistenza prodotto prima della modifica
- Gestione errori per record non trovati

Sistema Ordini

Generazione ID Ordine

const ordineId = Number(Date.now().toString() + Math.floor(Math.random() * 1000).toString());

Struttura Dati

- Un record per ogni prodotto nell'ordine
- Collegamento cliente-venditore per ogni prodotto
- Quantità specifica per prodotto

Processo Checkout

1. Recupera carrello cliente
2. Verifica disponibilità tutti i prodotti
3. Inserisce record ordine
4. Aggiorna disponibilità prodotti
5. Svuota carrello

Frontend

1. Panoramica

Il frontend dell'applicazione è stato realizzato interamente in Javascript, HTML e CSS, con una struttura modulare divisa per responsabilità: login, registrazione, visualizzazione dei prodotti, gestione del carrello e dashboard utente.
Ogni funzionalità è stata distribuita in file separati e cartelle con nomi mirati per semplificare la manutenzione e migliorare la chiarezza.

2. Struttura delle cartelle frontend

RegistrationForm/

Contiene i file relativi alla registrazione di nuovi utenti (clienti e aziende):

- `formRegistrationAzienda.html`: form HTML per la registrazione di un'azienda/artigiano, inclusa la logica di interazione.
- `formRegistrationCliente.html`: form HTML per la registrazione di un cliente, comprensivo della logica di interazione.
- `styleRegistration.css`: file CSS che gestisce l'aspetto grafico delle pagine di registrazione.

LoginForm/

Contiene i file per il login degli utenti:

- `login-registration.html`: pagina HTML che presenta il form di login.
- `Esecution.js`: script JavaScript che implementa la logica di autenticazione e interazione con il backend.
- `styleLogin.css`: file CSS per la grafica della pagina di login.

ShopForm/

Contiene i file per la visualizzazione e gestione del negozio:

- `shop.html`: pagina HTML che mostra tutti i prodotti presenti nel database.
- `shopEsecutor.js`: script che gestisce l'aggiunta al carrello, i filtri, e la predisposizione degli articoli nello shop.

- **`shopStyle.css`**: foglio di stile CSS dedicato alla pagina dello shop.
- **`InfoProdottoSpec.html`**: pagina HTML per la visualizzazione dei dettagli di un singolo prodotto.
- **`specifiche.js`**: script JavaScript che gestisce la logica e il caricamento dinamico della pagina di dettaglio prodotto.
- **`infoSpecifiche.css`**: file CSS dedicato alla pagina delle specifiche prodotto.

CartForm/

Gestisce il carrello e le sue interazioni:

- **`cart.html`**: pagina HTML che mostra il contenuto del carrello.
- **`cart.js`**: script che gestisce i popup, le interazioni dell'utente e la comunicazione col backend.
- **`cartStyle.css`**: foglio di stile per l'interfaccia del carrello.

UserForm/

Contiene i file relativi alla dashboard utente (sia cliente che azienda):

- **`dashboard.html`**: pagina HTML che rappresenta la dashboard dell'utente o dell'artigiano.
- **`dashboard.js`**: script JavaScript che gestisce la logica della dashboard.
- **`dashboardStyle.css`**: file CSS per lo stile della dashboard.

3. Gestione del login

Per il login si è voluto organizzare un form con un animazione che ti permette di cambiare foirm da cliente ad azienda tramite codice CSS,l'azione del login iniziava quando l'utente va a premere sul tasto "Accedi" che fa partire un metodo nel file javascript che attraverso una chiamata post chiede al backend di verificare se esiste l'utente o no,se non esiste l'utente,verrà visualizzato a sachermo un alert message che indica il tipo di errore che può essere,"utente non trovato" o "credenziale non corrette",se invece l'utente è stato trovato allora verrà fatto un indirizzamento alla pagina dei prodotti e verrà poi fatta partire la sessione tramite email e tipodi Utente quindi se cliente o paziente

```
document.getElementById("login-cliente").addEventListener("submit", async function(event) {
    event.preventDefault();

    const email = document.getElementById("email").value;
    const password = document.getElementById("pass").value;

    try {
        const response=await fetch("/loginCliente",{
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ email, password })
        })
        const result = await response.json();
        console.log(result);
        {
            if(!result.success){
                document.getElementById("email").value = "";
                document.getElementById("pass").value = "";
                alert(result.message);
            }else{
                //window.location.href = `../ShopForm/shop.html?id=${email}`;
                sessionStorage.setItem("userEmail", email);
                sessionStorage.setItem("userRole", "cliente");
                window.location.href = `../ShopForm/shop.html?id=${email}`;
            }
        }
    } catch (error) {

    }
```

La logica dell'azienda è identica ma cambia il tipo di input dato al backends che non saraà più solo email e password ma anche partita iva

4. Registrazione

Se l'utente invece non è registrato allora deciderà come prima cosa di svolgere la registrazione dove vengono richiesti tutti i dati dell'utente come nome,cognome,data di nascita,email,password e conferma password i quali tramite una chiamata post verranno dati in pasto al body della richiesta il quale backend elaborerà e restituirà un messaggio se la risposta è andata a buon fine o no

```
<script>
    document.getElementById("registrazioneCliente").addEventListener("submit", async function (event) {
        event.preventDefault();
        console.log("Registrazione cliente in corso...");
        const nome = document.getElementById("nome").value;
        const cognome = document.getElementById("cognome").value;
        const dataNascita = document.getElementById("dataNascita").value;
        const email = document.getElementById("email").value;
        const password = document.getElementById("pass").value;
        const confermaPassword = document.getElementById("passFinale").value;
        console.log(nome, cognome, dataNascita, email, password, confermaPassword);
        if (password != confermaPassword) {
            alert("Le password non corrispondono");
            document.getElementById("pass").value = "";
            document.getElementById("passFinale").value = "";
        } else {

            try {
                const response = await fetch("/registrazioneCliente", {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json"
                    },
                    body: JSON.stringify({ nome, cognome, email, dataNascita, password })
                })
                const result = await response.json();
                console.log("Risposta dal server:", result);
                if (result.success) {
                    // window.location.href = `../ShopForm/shop.html?id=${email}`;
                    sessionStorage.setItem("userEmail", email);
                    console.log("userEmail salvato in sessionStorage:", sessionStorage.getItem("userEmail"));
                    window.location.href = "../ShopForm/shop.html";

                } else {
                    alert(result.message);
                }
            } catch (error) {
                console.error("Errore durante la registrazione:", error);
                alert("Si è verificato un errore. Riprova più tardi.");
            }
        }

    });
</script>
```

Se la registrazione avviene con successo allora verrà salvata la mail e il tipo di utente in sessione verrà fatta visualizzare la pagina dello shop
Per quanto riguarda l'azienda,il procedimento è il medesimo ma con attributi diversi

5. Visualizzazione dello shop

Una volta effettuata la registrazione o il login verrà caricata la pagine che tramite un listeerd di DOMContentLoaded ovvero,quando la pagina è completamente caricata,viene prelevata la mail di sessione per poi nnavigare tra le varie pagine

```
function getParametro() {
    const urlParams = new URLSearchParams(window.location.search);
    idUtente = urlParams.get('userEmail');

    if (idUtente && idUtente !== "null") {
        sessionStorage.setItem("userEmail", idUtente);
        console.log("userId salvato in sessionStorage:", idUtente);
    } else {
        idUtente = sessionStorage.getItem("userEmail");
        console.log("userId preso da sessionStorage:", idUtente);
    }

    return idUtente || null;
}
```

e verranno stampati tutti i prodotti facendo un post request al backend che risponderà con un json che verrà analizzato per stampare i prodotti nel modo che vogliamo e che abbiamo strutturato

```
async function restituisciTuttiProdotti() {
    try {
        const response = await fetch("/prodotti", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            }
        });

        const result = await response.json();
        return result.prodotti;
    } catch (error) {
        alert("Errore durante il recupero dei prodotti: " + error.message);
        return [];
    }
}
```

```
function stampaProdotti(prodotti) {
    const container = document.getElementById("prodotti");
    container.innerHTML = "";

    if (!prodotti || prodotti.length === 0) return;

    const fragment = document.createDocumentFragment();

    prodotti.forEach(prodotto => {
        const div = document.createElement("div");
        div.className = "product";
        div.onclick = () => {
            sessionStorage.setItem("idProduct", prodotto.id);
            location.href = `InfoProdottoSpec.html?idProdotto=${prodotto.id}&email=${idUtente}`;
        };

        const img = document.createElement("img");
        img.src = prodotto.immagine;
        img.alt = prodotto.nome;
        img.loading = "lazy";
        div.appendChild(img);

        const h3 = document.createElement("h3");
        h3.textContent = prodotto.nome;
        div.appendChild(h3);

        const price = document.createElement("p");
        price.innerHTML = `<strong> Prezzo: ${prodotto.prezzo} €</strong>`;
        div.appendChild(price);

        const btn = document.createElement("button");
        if (prodotto.disponibilita === "0") {
            btn.textContent = "Prodotto non disponibile";
            btn.disabled = true;
            btn.className = "disabled-button";
        } else {
            btn.textContent = "Aggiungi al Carrello";
            btn.onclick = (event) => {
                event.stopPropagation();
                aggiuntaProdottoCarrello(prodotto.id, idUtente);
            };
        }

        div.appendChild(btn);
        fragment.appendChild(div);
```

I prodotti sono contenuti in un div che appenda viene premuto verrà aperta una pagina che mostra la descrizione avanzata del prodotto con tutte le informazioni del caso,questo grazie all'id del prodotto che ricaviamo dal json di risposta

Troviamo poi un pulsante di aggiunta al carrello che ci permette di aggiungerlo direttamente al carrello senza entrare nella descrizione grazie all'email salvata in sessione e all'id del prodotto,
notiamo un event.stopPropagation perché copsì non andiamo a fare conflitto con il metodo che permette di visualizzare la pagina completa premendo sul riquadro

```
async function aggiuntaProdottoCarrello(idProdotto, idUtente) {
    try {
        const response = await fetch("/aggiungiProdottoCarrello", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ idProdotto, email: idUtente })
        });
        console.log("Aggiunta prodotto al carrello:", { idProdotto, email: idUtente });
        const result = await response.json();
        console.log("Risultato dell'aggiunta al carrello:", result);

        if (result.success) {
            alert("Prodotto aggiunto al carrello con successo!");
        } else {
            alert("Errore nell'aggiunta del prodotto al carrello: " + result.message);
        }
    } catch (error) {
        console.error("Errore durante l'aggiunta al carrello:", error);
        alert("Errore durante l'aggiunta del prodotto al carrello: " + error.message);
    }
}
```

Nello shop come per tutti gli e-commerce vengono ssviluppati e progettati i filtri,in questo caso troviamo un filtro per nome ovvero tramite una chiamata post dove vine mandata la stringa scritta vengono restituiti tutti i prodotti che hanno quella stringa contenuta

```
async function ricercaNome() {
    const filtroInput = document.getElementById("filtroNome");
    const nomeRicerca = filtroInput.value.toLowerCase().trim();
    if (nomeRicerca === "") {
        filtroInput.value = "";
        filtroInput.placeholder = "Cerca Prodotti....";

        const prodotti = await restituisciTuttiProdotti();
        return stampaProdotti(prodotti);
    } else {
        try {
            const response = await fetch("/ricercaProdottiNome", {
                method: "POST",
                headers: {
                    "Content-Type": "application/json"
                },
                body: JSON.stringify({ nome: nomeRicerca })
            });

            const result = await response.json();

            if (result.success) {
                stampaProdotti(result.prodotti);
            } else {
                alert("Nessun prodotto trovato con il nome: " + nomeRicerca);
                const prodotti = await restituisciTuttiProdotti();
                stampaProdotti(prodotti);
            }
        } catch (error) {
            alert("Errore durante la ricerca dei prodotti: " + error.message);
        }
    }
}
```

Ma anche una sezione per i filtri che attraverso una checkBox va a verificare che requisiti vuole il cliente e adattare la pagina con i prodotti risultanti:

```
async function applicaFiltriProdotti(filtri) {
    const categorie = [];

    if (filtri.categoria.uomo) categorie.push("'uomo'");
    if (filtri.categoria.donna) categorie.push("'donna'");
    if (filtri.categoria.bambino) categorie.push("'bambino'");
    if (filtri.categoria.bambina) categorie.push("'bambina'");
    if (filtri.categoria.unisex) categorie.push("'unisex'");

    let prodotti = [];

    try {
        if (categorie.length > 0) {
            const query = `SELECT * FROM prodotti WHERE categoria IN (${categorie.join(",")})`;
            const response = await fetch("/prodottoByCategoria", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ query })
            });

            const result = await response.json();
            if (result.success) {
                prodotti = result.prodotti;
            } else {
                alert("Errore nel filtro per categoria: " + result.message);
                return;
            }
        } else {
            prodotti = await restituisciTuttiProdotti();
        }

        if (filtri.disponibilita) {
            prodotti = prodotti.filter(p => p.disponibilita > 0);
        }

        if (filtri.prezzo.crescente) {
            prodotti.sort((a, b) => a.prezzo - b.prezzo);
        } else if (filtri.prezzo.decrescente) {
            prodotti.sort((a, b) => b.prezzo - a.prezzo);
        }

        stampaProdotti(prodotti);
        alert("Filtri applicati con successo!");
    } catch (error) {
        console.error("Errore nell'applicazione dei filtri:", error);
```

6. Funzionalità Carrello (CartForm/cart.js)

Il file cart.js implementa tutte le funzionalità legate alla gestione del carrello:

Recupero prodotti nel carrello da endpoint /Cart/idProdotti

Funzione che invia una richiesta GET al backend per ottenere l'elenco degli id dei prodotti presenti nel carrello del cliente.

const ris = await fetch(`/Cart/idProdotti?cliente=${email}`);
const idProdotto = await ris.json();

Visualizzazione dinamica dei prodotti in pagina

Per ogni prodotto nel carrello, crea dinamicamente un blocco HTML con nome, prezzo, quantità e pulsanti per rimuovere o modificare.

div.innerHTML = `

```
<div class="description">${prodotto.nome}</div>
<div class="price">€ ${prezzoTotale.toFixed(2)}</div>
<div class="quantity">x${quantita}</div>
`;
```

Rimozione di un prodotto dal carrello con la chiamata POST /cart/rimuovitProdotto

Invia una richiesta POST al backend per rimuovere un prodotto specifico dal carrello del cliente. In caso di successo, rimuove anche il relativo blocco HTML dalla pagina e ricalcola il totale.

```
await fetch('/cart/rimuoviProdotto', {
  method: 'POST',
  body: JSON.stringify({ clienteId: email, prodottoId: id }),
});
```

Modifica quantità con POST /cart/modificaQuantita

Permette all'utente di aggiornare la quantità di un prodotto nel carrello. Invia i nuovi dati al backend e riceve conferma. In caso positivo, la quantità viene aggiornata anche a livello di interfaccia.

```
await fetch('/cart/modificaQuantita', {
  method: 'POST',
  body: JSON.stringify({ clienteId: email, prodottoId: id, quantita }),
});
```

Calcolo totale del carrello in tempo reale

Somma dinamicamente il totale del carrello leggendo i prezzi e le quantità correnti. Se il carrello è vuoto, mostra un messaggio alternativo e nasconde il riepilogo.

```
function aggiornaTotale() {
  const items = document.querySelectorAll(".cart-item");
  let totale = 0;

  items.forEach(item => {
    const prezzo = parseFloat(item.querySelector(".price").textContent.replace("€",
"").trim());
    const quantita = parseInt(item.querySelector(".qty-value").textContent, 10);
    if (!isNaN(prezzo) && !isNaN(quantita)) {
      totale += prezzo * quantita;
    }
  });

  const totaleCart = document.getElementById("cart-total");
  if (totaleCart) {
```

```
  totaleCart.textContent = totale.toFixed(2);
 }
```

Procedura di checkout con POST /checkout
Al clic su “Paga Ora”, viene inviata una richiesta POST per processare l’ordine. Se va a buon fine, il carrello viene svuotato e l’interfaccia aggiornata con un messaggio di conferma.

```
try {
 const response = await fetch("/checkout", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ clienteId: email })
 });

 const result = await response.json();
 if (!result.success) {
  alert("Errore durante il pagamento: " + (result.error || "Contatta l'amministratore"));
  return;
 }
```

Gestione carrello vuoto con visualizzazione alternativa
Se il carrello è privo di prodotti, l’interfaccia nasconde il riepilogo e mostra un messaggio statico che invita a continuare lo shopping.

```
if (idC.length === 0) {
 document.querySelector(".cart-summary-wrapper").style.display = "none";
 document.querySelector(".cart-empty").style.display = "block";
 return;
}
```

7. Funzionalità Dashboard Utente UserForm/dashboard.js

Questa sezione gestisce la dashboard personalizzata sia per clienti che artigiani, a seconda del tipo utente ritornato da /utente/tipo.

Gestione autenticazione e ruolo
All'inizio di ogni pagina (DOMContentLoaded), il frontend controlla se sono presenti userEmail e userRole in sessionStorage. In caso contrario, l’utente viene reindirizzato al login.

```
 const email = getParametro();
 const role = getUserRole();

 try {
  if (!email || !role) {
   alert("Sessione non trovata. Effettua il login.");
   window.location.href = "../LoginForm/login-registration.html";
   return;
```

```
    }

    const response = await fetch(`/utente/tipo?email=${email}`);
    const data = await response.json();

    if (!data.tipo) {
      alert("Errore: tipo di utente non trovato.");
      window.location.href = "../LoginForm/login-registration.html";
      return;
    }
```

Inoltre per limitare l'accesso:
- Solo i cliente possono accedere al carrello
- Gli artigiani accedono alla dashboard per la gestione delle vendite

Cliente

Visualizza dati personali /cliente/nome , /cliente/cognome , /cliente/data_nascita
Recupera da tre endpoint separati nome, cognome e data di nascita del cliente, quindi li mostra nella dashboard personale.

```
async function caricaDatiCliente(email) {

  try {
    const [nomeRes, cognomeRes, dataNascitaRes] = await Promise.all([
      fetch(`/cliente/nome?id=${email}`).then(res => res.json()),
      fetch(`/cliente/cognome?id=${email}`).then(res => res.json()),
      fetch(`/cliente/data_nascita?id=${email}`).then(res => res.json())
    ]);

    const nome = nomeRes.nome;
    const cognome = cognomeRes.cognome;
    const formatoData = new
Date(dataNascitaRes.data_nascita).toLocaleDateString("it-IT");
```

Mostra storico ordini /ordine/cliente
Ottiene l'elenco di ordini effettuati dal cliente. Se l'elenco è vuoto, viene mostrato un messaggio informativo. Ogni ordine viene mostrato come card interattiva.

```
  try {
    const response = await fetch(`/ordine/cliente?cliente=${email}`);
    const ordini = await response.json();
    orderList.innerHTML = "";

    const uniqueOrdini = Array.from(new Set(ordini));
    const ordersContainer = document.createElement("div");
    ordersContainer.classList.add("orders-container");

    if (!Array.isArray(uniqueOrdini) || uniqueOrdini.length === 0) {
```

```
      orderList.innerHTML = `<div class="no-orders">
      Nessun ordine trovato!<br><br>
        Quando effettuerai un acquisto, lo troverai qui!
      </div>`;
      return;
    }
```

Dettagli contenuti ordine /ordine/prodotti
Per ogni ordine, recupera e mostra i prodotti acquistati con le relative quantità. Ogni riga mostra il nome del prodotto e il numero di unità

```
 const prodottiRes = await fetch(`/ordine/prodotti?cliente=${email}&ordine=${ordineId}`);
 const prodotti = await prodottiRes.json();

 const ul = document.createElement("ul");
 let prodottiValidi = 0;

 if (!Array.isArray(prodotti)) {
   console.error("Prodotti non validi o errore dal backend:", prodotti);
   continue;
 }
```

Visualizza statistiche /stat/oridini , /stat/spesa , /stat/quantita
Mostra il totale degli ordini effettuati, dei prodotti acquistati e l'importo totale speso. Le informazioni sono raccolte da tre endpoint dedicati.

```
async function caricaStatisticheOrdini(email) {
  const statistiche = document.getElementById("order-stats");

  if (!statistiche) return;

  try {
    const [ordini, prodotti, spesa] = await Promise.all([
      fetch(`/stat/ordini?cliente=${email}`).then(res => res.json()),
      fetch(`/stat/quantita?cliente=${email}`).then(res => res.json()),
      fetch(`/stat/spesa?cliente=${email}`).then(res => res.json())
    ]);
```

Artigiano

Caricamento dati aziendali /artigiano/info
Recupera nome azienda, email, telefono e partita IVA dell'artigiano. Queste informazioni sono poi mostrate nella sezione "Dati utente" della dashboard.

```
async function caricaDatiVenditore(email) {
  let ivaVenditore = null
```

```
  try {
    const info = await fetch(`/artigiano/info?email=${email}`).then(res => res.json());
    ivaVenditore = info.iva;
    console.log("IVA Venditore:", ivaVenditore);
    sessionStorage.setItem("ivaVenditore", info.iva);
    console.log("IVA Venditore salvata in sessionStorage:",
sessionStorage.getItem("ivaVenditore"));
    const venditoreSection = document.getElementById("user-data");

    venditoreSection.innerHTML = `
      <p><strong>Nome azienda:</strong> ${info.nomeazienda}</p>
      <p><strong>Partita IVA:</strong> ${info.iva}</p>
      <p><strong>Email:</strong> ${info.email}</p>
      <p><strong>Telefono:</strong> ${info.numerotel}</p>
    `;
```

Visualizzazione dei prodotti venduti /prodotti/idVenditore

Recupera tutti i prodotti associati alla partita IVA dell'artigiano e li mostra come lista nella dashboard venditore.

```
async function caricaProdottiVenditore(email) {
  try {
    const ivaData = await fetch(`/artigiano/iva-by-email?email=${email}`).then(res =>
res.json());
    const ivaVenditore = ivaData.iva;
    console.log("IVA Venditore:", ivaVenditore);

    const prodotti = await
fetch(`/prodotti/idvenditore?idvenditore=${ivaVenditore}`).then(res => res.json());
    console.log("Prodotti trovati:", prodotti);

    const lista = document.getElementById("lista-prodotti");
    lista.innerHTML = "";

    if (prodotti.length === 0) {
      lista.innerHTML = "<li>Nessun prodotto trovato</li>";
      return;
    }
```

Storico ordini ricevuti /ordine/venditore

Ottiene gli ordini ricevuti da clienti per i prodotti dell'artigiano. Mostra ogni ordine in una card, con lista prodotti, quantità e totale. Offre anche la possibilità di eliminare un ordine.

```
async function caricaStoricoOrdiniArtigiano(email) {
  const orderList = document.getElementById("order-history-artigiano");
  if (!orderList) return;

  try {
    const ivaRes = await fetch(`/artigiano/iva-by-email?email=${email}`);
```

```
  const ivaData = await ivaRes.json();
  const ivaVenditore = ivaData.iva;

  // Recupera tutti gli ordini dove il venditore è questo artigiano
  const response = await fetch(`/ordine/venditore?ivavenditore=${ivaVenditore}`);
  const ordini = await response.json();
  console.log("Ordini trovati per artigiano:", ordini);
  orderList.innerHTML = "";

  if (!Array.isArray(ordini) || ordini.length === 0) {
    orderList.innerHTML = `<div class="no-orders">
      Nessun ordine trovato!<br><br>
      Quando riceverai un ordine, lo troverai qui!
    </div>`;
    return;
  }
```

Aggiunta, modifica e rimozione prodotto /prodotto/aggiungi , /modifica , /elimina

Gestisce il form per l'aggiunta di un nuovo prodotto. Valida i campi, costruisce un oggetto FormData e invia i dati, inclusa l'immagine, al backend via POST

```
const formData = new FormData();
formData.append("nome", nome);
formData.append("immagine", immagine.files[0]);
await fetch("/prodotto/aggiungi", { method: "POST", body: formData });
```

Permette di modificare nome, prezzo, disponibilità, descrizione, categoria e immagine di un prodotto esistente. I dati vengono inviati tramite richiesta PUT.

```
const formData = new FormData();
formData.append("id", id);
formData.append("nome", nome);
await fetch("/prodotto/modifica", { method: "PUT", body: formData });
```

Dopo una conferma da parte dell'utente, invia una richiesta DELETE al backend per eliminare il prodotto selezionato.

```
await fetch("/prodotto/elimina", {
  method: "DELETE",
  body: JSON.stringify({ id }),
  headers: { "Content-Type": "application/json" }
});
```

Gestione popup CRUD prodotti
Tutti i form del popup form-aggiungi, form-modifca, form-elimina sono gestiti dinamicamente e mostrati/nascosti in base al bottone premuto