Endpoint: `/api/login`

Metodo: `POST`

Descrizione: Autentica un utente esistente.

Parametri della richiesta (JSON Body): - `username` (stringa, obbligatorio): Il nome utente dell'utente. - `password` (stringa, obbligatorio): La password dell'utente.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "id": "<user_id>",
    "username": "<username>",
    "email": "<email>",
    "telefono": "<telefono>",
    "indirizzo": "<indirizzo>",
    "role": "<ruolo>"
  }
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "Username e password sono obbligatori" }` (Se `username` o `password` sono mancanti) - **Status:** `401 Unauthorized` - **Corpo:** `json { "success": false, "message": "Utente non trovato" }` (Se l'username non esiste) - **Corpo:** `json { "success": false, "message": "Password errata" }` (Se la password non corrisponde) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/login", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    username: "testuser",
    password: "password123",
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/register`

Metodo: `POST`

Descrizione: Registra un nuovo utente nel sistema.

Parametri della richiesta (JSON Body): - `username` (stringa, obbligatorio): Il nome utente desiderato. - `password` (stringa, obbligatorio): La password per il nuovo utente. - `email` (stringa, obbligatorio): L'indirizzo email dell'utente. - `telefono` (stringa, opzionale): Il numero di telefono dell'utente. - `ruolo` (stringa, opzionale): Il ruolo dell'utente (es. 'cliente', 'venditore'). - `indirizzo` (stringa, opzionale): L'indirizzo fisico dell'utente.

Risposte di successo: - **Status:** `201 Created` - **Corpo:**

```
{
  "success": true,
  "data": {
    "username": "<username>",
    "email": "<email>",
    "telefono": "<telefono>",
    "indirizzo": "<indirizzo>",
    "role": "<ruolo>"
  }
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "Username e password sono obbligatori" }` (Se `username`, `password` o `email` sono mancanti) - **Status:** `409 Conflict` - **Corpo:** `json { "success": false, "message": "Un utente con questo username è già esistente" }` (Se l'username è già in uso) - **Status:** `500 Internal Server Error` - **Corpo:** `json {`

`"success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/register", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    username: "newuser",
    password: "securepass",
    email: "newuser@example.com",
    telefono: "1234567890",
    ruolo: "cliente",
    indirizzo: "Via Roma 1, Milano",
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/user/:username`

Metodo: `GET`

Descrizione: Recupera i dettagli di un utente specifico tramite il suo username.

Parametri della richiesta (URL Parameter): - `:username` (stringa, obbligatorio): L'username dell'utente da cercare.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "id": "<user_id>",
    "username": "<username>",
    "email": "<email>",
    "telefono": "<telefono>",
    "indirizzo": "<indirizzo>",
    "role": "<ruolo>"
  }
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "Sono presenti caratteri speciali" }` (Se l'username contiene

caratteri speciali non validi) - **Status:** `401 Unauthorized` - **Corpo:** `json { "success": false, "message": "Utente non trovato" }` (Se l'utente con l'username specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/user/testuser")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/user/update`

Metodo: `POST`

Descrizione: Aggiorna i dettagli di un utente esistente.

Parametri della richiesta (JSON Body): - `id` (numero, obbligatorio): L'ID dell'utente da aggiornare. - `username` (stringa, obbligatorio): Il nuovo nome utente. - `password` (stringa, obbligatorio): La nuova password (verrà hashata). - `telefono` (stringa, obbligatorio): Il nuovo numero di telefono. - `indirizzo` (stringa, obbligatorio): Il nuovo indirizzo.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "id": "<user_id>",
    "username": "<username>",
    "email": "<email>",
    "telefono": "<telefono>",
    "indirizzo": "<indirizzo>",
    "role": "<ruolo>"
  }
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Utente non trovato" }` (Se l'utente con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/user/update", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    id: 1,
    username: "updateduser",
    password: "newpassword",
    telefono: "0987654321",
    indirizzo: "Via Verdi 10, Roma",
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/:id/image`

Metodo: `GET`

Descrizione: Recupera le immagini associate a un prodotto specifico.

Parametri della richiesta (URL Parameter): - `:id` (numero, obbligatorio): L'ID del prodotto di cui recuperare le immagini.

Parametri della query (Opzionali): - `max` (numero, opzionale): Il numero massimo di immagini da restituire. Se non specificato o 0, restituisce tutte le immagini.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "id": "<image_id_1>",
      "image": "<image_url_1>"
    },
    {
      "id": "<image_id_2>",
      "image": "<image_url_2>"
    }
  ]
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Prodotto non trovato" }` (Se il prodotto con l'ID specificato non esiste o non ha immagini) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
// Recupera tutte le immagini per il prodotto con ID 123
fetch("/api/product/123/image")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));

// Recupera al massimo 5 immagini per il prodotto con ID 123
fetch("/api/product/123/image?max=5")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/:id/image/add`

Metodo: `POST`

Descrizione: Aggiunge una o più immagini a un prodotto specifico.

Parametri della richiesta (URL Parameter): - `:id` (numero, obbligatorio): L'ID del prodotto a cui aggiungere le immagini.

Parametri della richiesta (JSON Body): - `images` (array di stringhe, obbligatorio): Un array di URL o percorsi delle immagini da aggiungere.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "id": "<image_id_1>",
      "idProd": "<product_id>",
      "img": "<image_url_1>"
    },
    {
      "id": "<image_id_2>",
      "idProd": "<product_id>",
      "img": "<image_url_2>"
    }
  ]
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "Lista di immagini non fornita o vuota nel corpo della richiesta." }` (Se l'array `images` è mancante, vuoto o non è un array) - **Corpo:** `json { "success": false, "message": "Nessuna immagine valida fornita." }` (Se nessuna immagine valida è stata fornita nel corpo della richiesta) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server: nessuna riga inserita." }` (Se l'inserimento delle immagini nel database fallisce) - **Corpo:** `json { "success": false, "message": "Errore interno del server." }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/product/123/image/add", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    images: [
      "http://example.com/image1.jpg",
      "http://example.com/image2.png",
    ],
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/image/:id/delete`

Metodo: `DELETE`

Descrizione: Elimina un'immagine specifica dal database.

Parametri della richiesta (URL Parameter): - `:id` (numero, obbligatorio): L'ID dell'immagine da eliminare.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Immagine non trovata" }` (Se l'immagine con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/product/image/123/delete", {
  method: "DELETE",
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/products`

Metodo: `GET`

Descrizione: Recupera un elenco di prodotti, con la possibilità di applicare filtri.

Parametri della query (Opzionali): - `categoria` (stringa o array di stringhe): Filtra i prodotti per categoria. Può essere una singola categoria o un elenco di categorie separate da virgole. - `prezzo_min` (numero): Filtra i prodotti con un prezzo minimo specificato. - `prezzo_max` (numero): Filtra i prodotti con un prezzo massimo specificato. - `disponibilita` (numero): Filtra i prodotti con una disponibilità minima specificata. - `limit` (numero): Limita il numero di prodotti restituiti.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "prodotto": {
        "id": "<product_id>",
        "nome": "<product_name>",
        "prezzo": "<product_price>",
        "disponibilita": "<product_availability>",
        "categoria": "<product_category>",
        "descrizione": "<product_description>",
        "artigiano_id": "<seller_id>"
      },
      "immagini": [
        {
          "id": "<image_id_1>",
          "image": "<image_url_1>"
        }
      ]
    }
  ]
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Nessun prodotto trovato" }` (Se nessun prodotto corrisponde ai filtri specificati) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
// Recupera tutti i prodotti
fetch("/api/products")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));

// Recupera prodotti con categoria 'elettronica' e prezzo massimo 100
fetch("/api/products?categoria=elettronica&prezzo_max=100")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/products/:id`

Metodo: `GET`

Descrizione: Recupera i dettagli di un singolo prodotto tramite il suo ID.

Parametri della richiesta (URL Parameter): - `:id` (numero, obbligatorio): L'ID del prodotto da recuperare.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "id": "<product_id>",
    "nome": "<product_name>",
    "categoria": "<product_category>",
    "descrizione": "<product_description>",
    "prezzo": "<product_price>",
    "disponibilita": "<product_availability>",
    "artigiano_id": "<seller_id>",
    "immagini": [
      {
        "id": "<image_id_1>",
        "image": "<image_url_1>"
      }
    ]
  }
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "ID prodotto non valido" }` (Se l'ID del prodotto non è un numero valido o è <= 0) - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Prodotto non trovato" }` (Se il prodotto con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/products/123")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/products/seller/:id`

Metodo: `GET`

Descrizione: Recupera tutti i prodotti associati a un venditore specifico tramite il suo ID.

Parametri della richiesta (URL Parameter): - `:id` (numero, obbligatorio): L'ID del venditore di cui recuperare i prodotti.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "prodotto": {
        "id": "<product_id>",
        "nome": "<product_name>",
        "prezzo": "<product_price>",
        "disponibilita": "<product_availability>",
        "categoria": "<product_category>",
        "descrizione": "<product_description>",
        "artigiano_id": "<seller_id>"
      },
      "immagini": [
        {
          "id": "<image_id_1>",
          "image": "<image_url_1>"
        }
      ]
    }
  ]
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "ID venditore non valido" }` (Se l'ID del venditore non è un numero valido o è <= 0) - **Status:** `404 Not Found` - **Corpo:** `json { "succe (Content truncated due to size limit. Use line ranges to read in chunks) (Se nessun prodotto è associato al venditore specificato) - **Status:** `500 Internal Server Error` - **Corpo:** json { "success": false, "message": "Errore interno del server" } ``` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/products/seller/456")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/search`

Metodo: `GET`

Descrizione: Cerca prodotti in base a un termine di ricerca nel nome o nella descrizione.

Parametri della query (Obbligatori): - `q` (stringa): Il termine di ricerca.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "prodotto": {
        "id": "<product_id>",
        "nome": "<product_name>",
        "prezzo": "<product_price>",
        "disponibilita": "<product_availability>",
        "categoria": "<product_category>",
        "descrizione": "<product_description>",
        "artigiano_id": "<seller_id>"
      },
      "immagini": [
        {
          "id": "<image_id_1>",
          "image": "<image_url_1>"
        }
      ]
    }
  ]
}
```

Risposte di errore: - **Status:** `400 Bad Request` - **Corpo:** `json { "success": false, "message": "Il termine di ricerca è obbligatorio" }` (Se il termine di ricerca `q` è mancante o vuoto) - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Nessun prodotto trovato" }` (Se nessun prodotto corrisponde al termine di ricerca) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/search?q=scarpe")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/add`

Metodo: `POST`

Descrizione: Aggiunge un nuovo prodotto al database.

Parametri della richiesta (JSON Body): - `nome` (stringa, obbligatorio): Il nome del prodotto. - `categoria` (stringa, obbligatorio): La categoria del prodotto. - `descrizione` (stringa, obbligatorio): La descrizione del prodotto. - `prezzo` (numero, obbligatorio): Il prezzo del prodotto. - `disponibilita` (numero, obbligatorio): La quantità disponibile del prodotto. - `idVenditore` (numero, obbligatorio): L'ID del venditore che aggiunge il prodotto.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "product": {
    "id": "<product_id>",
    "nome": "<product_name>",
    "categoria": "<product_category>",
    "descrizione": "<product_description>",
    "prezzo": "<product_price>",
    "disponibilita": "<product_availability>",
    "artigiano_id": "<seller_id>"
  }
}
```

Risposte di errore: - **Status:** `409 Conflict` - **Corpo:** `json { "success": false, "message": "product_already_exists" }` (Se un prodotto con lo stesso nome e venditore esiste già) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "server_error" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/product/add", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    nome: "Nuovo Prodotto",
    categoria: "Elettronica",
    descrizione: "Descrizione del nuovo prodotto.",
    prezzo: 99.99,
    disponibilita: 100,
    idVenditore: 1
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/delete`

Metodo: `DELETE`

Descrizione: Elimina un prodotto dal database tramite il suo ID.

Parametri della richiesta (JSON Body): - `id` (numero, obbligatorio): L'ID del prodotto da eliminare.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Prodotto non trovato" }` (Se il prodotto con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/product/delete", {
  method: "DELETE",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    id: 123,
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/product/update`

Metodo: `POST`

Descrizione: Aggiorna i dettagli di un prodotto esistente.

Parametri della richiesta (JSON Body): - `id` (numero, obbligatorio): L'ID del prodotto da aggiornare. - `descrizione` (stringa, opzionale): La nuova descrizione del prodotto. - `prezzo` (numero, opzionale): Il nuovo prezzo del prodotto. - `disponibilita` (numero, opzionale): La nuova quantità disponibile del prodotto.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "id": "<product_id>",
    "nome": "<product_name>",
    "categoria": "<product_category>",
    "descrizione": "<new_description>",
    "prezzo": "<new_price>",
    "disponibilita": "<new_availability>",
    "artigiano_id": "<seller_id>"
  }
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Prodotto non trovato" }` (Se il prodotto con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/product/update", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    id: 123,
    descrizione: "Nuova descrizione del prodotto.",
    prezzo: 120.50,
    disponibilita: 50,
  }),
})
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/orders`

Metodo: `GET`

Descrizione: Recupera gli ordini in base all'ID dell'utente e al suo ruolo.

Parametri della query (Obbligatori): - `id` (numero): L'ID dell'utente (cliente o venditore). - `ruolo` (stringa): Il ruolo dell'utente (es. 'cliente', 'venditore').

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": [
    {
      "id": "<order_id>",
      "cliente_id": "<customer_id>",
      "venditore_id": "<seller_id>",
      "id_prodotto": "<product_id>",
      "quantita": "<quantity>",
      "id_pagamento": "<payment_id>",
      "data_creazione": "<creation_timestamp>"
    }
  ]
}
```

Note: Il campo `data_creazione` viene generato automaticamente dal database al momento della creazione dell'ordine.

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "Ordini non trovati" }` (Se nessun ordine è stato trovato per l'ID e il

ruolo specificati) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore interno del server" }` (Per errori generici del server)

Esempio di Fetch:

```
// Recupera gli ordini per un cliente con ID 1
fetch("/api/orders?id=1&ruolo=cliente")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));

// Recupera gli ordini per un venditore con ID 2
fetch("/api/orders?id=2&ruolo=venditore")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/checkout`

Metodo: `POST`

Descrizione: Crea una sessione di checkout per l'elaborazione del pagamento.

Parametri della richiesta (JSON Body): - `carrello` (array di oggetti, obbligatorio): Un array di oggetti, dove ogni oggetto rappresenta un articolo nel carrello con `prodotto_id` e `quantita`. - `cliente_id` (numero, obbligatorio): L'ID del cliente che effettua l'acquisto.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "url": "<stripe_checkout_url>"
  }
}
```

Risposte di errore: - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "message": "Errore nel checkout" }` (Per errori generici del server o problemi durante la creazione della sessione di checkout)

Esempio di Fetch:

```
fetch("/api/checkout", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    carrello: [
      { prodotto_id: 1, quantita: 2 },
      { prodotto_id: 3, quantita: 1 },
    ],
    cliente_id: 123,
  }),
})
  .then((response) => response.json())
  .then((data) => {
    if (data.success) {
      window.location.href = data.data.url; // Reindirizza l'utente alla pagina
di checkout di Stripe
    } else {
      console.error("Errore nel checkout:", data.message);
    }
  })
  .catch((error) => console.error("Errore:", error));
```

Endpoint: `/api/checkout/status/:sessionId`

Metodo: `GET`

Descrizione: Verifica lo stato di una sessione di checkout Stripe.

Parametri della richiesta (URL Parameter): - `:sessionId` (stringa, obbligatorio): L'ID della sessione di checkout Stripe.

Risposte di successo: - **Status:** `200 OK` - **Corpo:**

```
{
  "success": true,
  "data": {
    "session": {
      "id": "<session_id>",
      "status": "<session_status>",
      "payment_status": "<payment_status>",
      "amount_total": "<total_amount>",
      "currency": "<currency>",
      "customer_email": "<customer_email>",
      "metadata": {}
    },
    "payment": {
      "id": "<payment_intent_id>",
      "status": "<payment_intent_status>",
      "amount": "<payment_amount>",
      "created": "<creation_timestamp>"
    }
  }
}
```

Risposte di errore: - **Status:** `404 Not Found` - **Corpo:** `json { "success": false, "message": "La sessione di checkount non è stata trovata" }` (Se la sessione con l'ID specificato non esiste) - **Status:** `500 Internal Server Error` - **Corpo:** `json { "success": false, "error": "<error_message>", "message": "Errore nel recupero dello stato della sessione" }` (Per errori generici del server)

Esempio di Fetch:

```
fetch("/api/checkout/status/cs_test_12345")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Errore:", error));
```