Cos’è un middleware in Express

Un **middleware** è semplicemente una **funzione** che riceve tre parametri principali:

`(req, res, next)`

- `req` → rappresenta la richiesta del client (dati, header, ecc.)
- `res` → è la risposta che invieremo al client
- `next()` → serve per dire ad Express: “Ho finito, passa al middleware successivo”

Se un middleware **non chiama `next()`**, la catena si ferma lì.

Flusso della richiesta

Immagina che una richiesta HTTP sia un “pacco” che attraversa una catena di montaggio:

1. Arriva a Express
2. Passa da `express.json()` (che lo trasforma in oggetto JS)
3. Passa da `cors()` (che gestisce le origini)
4. Passa da **middleware personalizzati** (es. logger, validate, auth)
5. Arriva infine alla **rotta** (`router.post(...)`)
6. Se succede un errore, finisce nel **middleware di errore**

1 Esempio: Middleware di logging (`logger.js`)

Questo middleware stampa ogni richiesta che arriva al server.

logger.js

```
// logger.js — Middleware per loggare le richieste

// Esporta una funzione middleware
module.exports = function logger(req, res, next) {
  const now = new Date().toLocaleTimeString(); // orario leggibile
  const method = req.method;                   // es: GET, POST, DELETE
  const url = req.originalUrl;                 // es: /api/users
  console.log(`[${now}] ${method} ${url}`);

  // Importante! Passa al middleware successivo
  next();
};
```

Come usarlo nel server

```
const express = require('express');
const logger = require('./middleware/logger'); // importa il middleware

const app = express();

app.use(logger); // usa il middleware su TUTTE le rotte
```

```
app.get('/', (req, res) => {
  res.send('Home Page');
});

app.listen(3000, () => console.log('Server avviato su http://localhost:3000'));
```

Cosa vedrai in console

```
[14:35:02] GET /
[14:35:10] POST /api/notes
```

2 Esempio: Middleware di validazione (`validate.js`)

Serve per **bloccare richieste incomplete** (es. quando mancano campi obbligatori nel `req.body`).

validate.js

```
// validate.js — Middleware per controllare i campi richiesti nel body

// Prende come parametro un array di campi obbligatori
function validate(requiredFields) {
  return (req, res, next) => {
    const missing = [];

    // Controlla se i campi mancano nel corpo della richiesta
    requiredFields.forEach((field) => {
      if (!req.body[field]) {
        missing.push(field);
      }
    });

    // Se mancano campi → risposta 400 Bad Request
    if (missing.length > 0) {
      return res.status(400).json({
        error: 'INVALID_INPUT',
        message: `Campi mancanti: ${missing.join(', ')}`
      });
    }

    // Tutto ok → passa alla rotta successiva
    next();
  };
}

module.exports = validate;
```

Come usarlo in una rotta

```
const express = require('express');
const validate = require('./middleware/validate');
const app = express();

app.use(express.json());

// Applichiamo il middleware solo su questa rotta
app.post('/api/register', validate(['email', 'password']), (req, res) => {
  res.json({ message: 'Utente registrato correttamente' });
});
```

```
app.listen(3000, () => console.log('Server avviato su http://localhost:3000'));
```

Esempi pratici:

Richiesta corretta

```
POST /api/register
{
  "email": "test@example.com",
  "password": "123456"
}
```

Risposta:

```
{ "message": "Utente registrato correttamente" }
```

Richiesta errata

```
POST /api/register
{
  "email": "test@example.com"
}
```

Risposta:

```
{
  "error": "INVALID_INPUT",
  "message": "Campi mancanti: password"
}
```

3 Come combinare più middleware insieme

Puoi usarne più di uno sulla stessa rotta, in sequenza:

```
app.post('/api/notes', logger, validate(['title', 'content']), (req, res) => {
  res.json({ message: 'Nota creata correttamente' });
});
```

Ordine di esecuzione:

1. `logger` stampa la richiesta
2. `validate` controlla i campi
3. Solo se tutto è ok → la rotta esegue il codice principale

Obiettivo didattico

> Il backend è come una **catena di montaggio**: ogni middleware fa un piccolo controllo o modifica i dati, poi passa il "pacchetto" al successivo. Se uno di questi trova un errore, ferma tutto e risponde subito al client.