Riferimento Rapido Express.js

Routing, middleware, richieste, risposte, pattern

Configurazione

Crea e Avvia il Server

```
const express = require("express");
const app = express();
app.listen(3000, () => console.log("Running on :3000"));
```

Middleware Integrati

```
app.use(express.json());          // parse JSON body
app.use(express.urlencoded({ extended: true })); // dati form
app.use(express.static("public")); // file statici
```

Routing

Metodi HTTP

```
app.get("/users", (req, res) => res.json(users));
app.post("/users", (req, res) => res.status(201).json(req.body));
app.put("/users/:id", (req, res) => res.json(updated));
app.delete("/users/:id", (req, res) => res.sendStatus(204));
```

Parametri di Route

```
app.get("/users/:id", (req, res) => {
  const { id } = req.params;
  res.json({ id });
});
```

Query String

```
// GET /search?q=express&page=2
app.get("/search", (req, res) => {
  const { q, page } = req.query;
  res.json({ q, page });
});
```

Middleware

Livello Applicazione

```
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});
```

Livello Route

```
const auth = (req, res, next) => {
  if (!req.headers.authorization) return res.sendStatus(401);
  next();
};
app.get("/secret", auth, (req, res) => res.json({ ok: true }));
```

Ordine di Esecuzione

`app.use(fn)`	Eseguito per ogni richiesta (in ordine)
`app.use(path, fn)`	Eseguito solo su percorso corrispondente
`next()`	Passa il controllo al middleware successivo
`next(err)`	Salta al gestore errori

Richiesta e Risposta

Oggetto Request

`req.params`	Parametri route (**`/users/:id`**)
`req.query`	Query string (**`?key=val`**)
`req.body`	Body della richiesta parsato (richiede parser)
`req.headers`	Oggetto header richiesta
`req.method`	Metodo HTTP (GET, POST, ...)
`req.path`	Pathname dell'URL
`req.cookies`	Cookie (richiede cookie-parser)

Oggetto Response

`res.json(obj)`	Invia risposta JSON
`res.send(body)`	Invia stringa/Buffer/oggetto
`res.status(code)`	Imposta stato HTTP (concatenabile)
`res.redirect(url)`	Redirect 302 (o passa stato)
`res.sendFile(path)`	Invia un file come risposta
`res.sendStatus(code)`	Invia stato con testo predefinito
`res.set(header, val)`	Imposta header risposta

Gestione Errori

Middleware di Errore

```
// Deve avere 4 parametri — Express lo riconosce come gestore errori
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ error: err.message });
});
```

Definisci i gestori errori dopo tutti gli altri app.use() e route

Errori Async

```
// Avvolgi gli handler async per catturare i rejection
const wrap = (fn) => (req, res, next) =>
  Promise.resolve(fn(req, res, next)).catch(next);

app.get("/data", wrap(async (req, res) => {
  const data = await fetchData();
  res.json(data);
}));
```

File Statici

Serve Directory Statica

```
app.use(express.static("public"));
// serve public/style.css su /style.css

// Con prefisso percorso virtuale
app.use("/assets", express.static("public"));
// serve public/style.css su /assets/style.css
```

Opzioni

`dotfiles`	**`'ignore'`** | **`'allow'`** | **`'deny'`**
`maxAge`	Cache-Control max-age in ms
`index`	Nome file indice (default: **`index.html`**)
`fallthrough`	Passa al middleware successivo in caso di 404

Template

Configurazione View Engine

```
app.set("view engine", "ejs");
app.set("views", "./views");

app.get("/", (req, res) => {
  res.render("index", { title: "Home", items: [1, 2, 3] });
});
```

Engine Comuni

`ejs`	Template JS embedded (**`<%= val %>`**)
`pug`	Basato su indentazione (ex Jade)
`handlebars`	Stile Mustache (**`{{val}}`**)

Router

Route Modulari

```
// routes/users.js
const router = require("express").Router();
router.get("/", (req, res) => res.json(users));
router.get("/:id", (req, res) => res.json(user));
module.exports = router;
```

Montare il Router

```
const usersRouter = require("./routes/users");
app.use("/api/users", usersRouter);
// GET /api/users -> router "/"
// GET /api/users/5 -> router "/:id"
```

Metodi Router

`router.get/post/put/delete`	Handler metodi HTTP
`router.use(fn)`	Middleware a livello router
`router.param(name, fn)`	Pre-elabora parametro route
`router.route(path)`	Concatena metodi su un percorso

Pattern di Autenticazione

Middleware JWT

```
const jwt = require("jsonwebtoken");
const auth = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.sendStatus(401);
  req.user = jwt.verify(token, process.env.SECRET);
  next();
};
```

Route Protette

```
app.get("/profile", auth, (req, res) => {
  res.json({ user: req.user });
});
app.use("/api/admin", auth, adminRouter);
```

Pattern Comuni

CORS

```
const cors = require("cors");
app.use(cors());                          // consenti tutte le
origini
app.use(cors({ origin: "https://example.com" })); // limita
```

Ambiente e Configurazione

```
const port = process.env.PORT || 3000;
app.listen(port);

// Accedi env nelle route
if (app.get("env") === "production") {
  app.use(helmet());
}
```

Pacchetti npm Utili

`cors`	Condivisione risorse cross-origin
`helmet`	Header di sicurezza
`morgan`	Logger richieste HTTP
`cookie-parser`	Parse header Cookie
`dotenv`	Carica **`.env`** in **`process.env`**
`multer`	Dati form multipart (upload file)