

REFERENCIA RÁPIDA DE EXPRESS.JS

Rutas, middleware, solicitudes, respuestas, patrones

Configuración

Crear e iniciar servidor

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

Middleware integrado

```
app.use(express.json());
app.use(express.urlencoded({ extended: true })); // datos de formulario
app.use(express.static("public")); // servir archivos estáticos
```

Rutas

Métodos 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));
```

Parámetros de ruta

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

Query strings

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

Middleware

A nivel de aplicación

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

A nivel de ruta

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

Orden de ejecución

app.use(fn) Se ejecuta en cada solicitud (en orden)

app.use(path, fn) Se ejecuta solo en el prefijo de ruta coincidente

next() Pasa el control al siguiente middleware

next(err) Salta al manejador de errores

Solicitud y respuesta

Objeto Request

req.params Parámetros de ruta (`/users/:id`)

req.query Query string (`?key=val`)

req.body Cuerpo de la solicitud parseado (necesita parser)

req.headers Objeto de cabeceras de solicitud

req.method Método HTTP (GET, POST, ...)

req.path Ruta de la URL

req.cookies Cookies (necesita cookie-parser)

Objeto Response

res.json(obj) Enviar respuesta JSON

res.send(body) Enviar cadena/Buffer/objeto

res.status(code) Establecer estado HTTP (encadenable)

res.redirect(url) Redirección 302 (o pasar estado)

res.sendFile(path) Enviar un archivo como respuesta

res.sendStatus(code) Enviar estado con texto por defecto

res.set(header, val) Establecer cabecera de respuesta

Manejo de errores

Middleware de errores

```
// Debe tener 4 parámetros - Express lo reconoce como manejador de errores
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ error: err.message });
});
```

Definir manejadores de errores después de todos los `app.use()` y rutas

Errores async

```
// Envolver manejadores async para capturar rechazos
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);
}));
```

Archivos estáticos

Servir directorio estático

```
app.use(express.static("public"));
// sirve public/style.css en /style.css

// Con prefijo de ruta virtual
app.use("/assets", express.static("public"));
// sirve public/style.css en /assets/style.css
```

Opciones

dotfiles `ignore` | `allow` | `deny``

maxAge Cache-Control max-age en ms

index Nombre del archivo índice (por defecto: `index.html`)

fallthrough Pasar al siguiente middleware en 404

Plantillas

Configurar motor de vistas

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

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

Motores comunes

ejs Plantillas JS embebidas (`<%= val %>`)

pug Basado en indentación (antes Jade)

handlebars Estilo Mustache (`{{val}}`)

Router

Rutas modulares

```
// 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;
```

Montar Router

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

Métodos del Router

router.get/post/put/delete Manejadores de métodos HTTP

router.use(fn) Middleware a nivel de router

router.param(name, fn) Pre-procesar parámetro de ruta

router.route(path) Encadenar métodos en una ruta

Patrones de autenticación

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();
};
```

Rutas protegidas

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

Patrones comunes

CORS

```
const cors = require("cors");
app.use(cors()); // permitir todos los orígenes
app.use(cors({ origin: "https://example.com" })); // restringir
```

Entorno y configuración

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

```
// Acceder a env en rutas
if (app.get("env") === "production" ) {
  app.use(helmet());
}
```

Paquetes npm útiles

cors Compartir recursos entre orígenes

helmet Cabeceras de seguridad

morgan Logger de solicitudes HTTP

cookie-parser Parsear cabecera Cookie

dotenv Cargar `env` en `process.env`

multer Datos de formulario multipart (carga de archivos)