Referência Rápida Express.js

Roteamento, middleware, requisições, respostas, padrões

Configuração

Criar e Iniciar Servidor

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

Middleware Embutido

```
app.use(express.json());          // parse JSON bodies
app.use(express.urlencoded({ extended: true })); // form data
app.use(express.static("public")); // serve static files
```

Roteamento

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));
```

Parâmetros de Rota

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

Query Strings

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

Middleware

Nível de Aplicação

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

Nível de Rota

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

Ordem de Execução

`app.use(fn)`	Executa em toda requisição (em ordem)
`app.use(path, fn)`	Executa apenas em prefixo de caminho correspondente
`next()`	Passa controle para o próximo middleware
`next(err)`	Pula para o tratador de erros

Requisição e Resposta

Objeto Request

`req.params`	Parâmetros de rota (**`/users/:id`**)
`req.query`	Query string (**`?key=val`**)
`req.body`	Corpo da requisição parseado (precisa de parser)
`req.headers`	Objeto de headers da requisição
`req.method`	Método HTTP (GET, POST, ...)
`req.path`	Pathname da URL
`req.cookies`	Cookies (precisa de cookie-parser)

Objeto Response

`res.json(obj)`	Enviar resposta JSON
`res.send(body)`	Enviar string/Buffer/objeto
`res.status(code)`	Definir status HTTP (encadeável)
`res.redirect(url)`	Redirecionamento 302 (ou passar status)
`res.sendFile(path)`	Enviar um arquivo como resposta
`res.sendStatus(code)`	Enviar status com texto padrão
`res.set(header, val)`	Definir header de resposta

Tratamento de Erros

Middleware de Erro

```
// Must have 4 parameters — Express recognizes it as error handler
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ error: err.message });
});
```

Defina tratadores de erro após todos os outros app.use() e rotas

Erros Assíncronos

```
// Wrap async route handlers to catch rejections
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);
}));
```

Arquivos Estáticos

Servir Diretório Estático

```
app.use(express.static("public"));
// serves public/style.css at /style.css

// With virtual path prefix
app.use("/assets", express.static("public"));
// serves public/style.css at /assets/style.css
```

Opções

`dotfiles`	**`'ignore'`** | **`'allow'`** | **`'deny'`**
`maxAge`	Cache-Control max-age em ms
`index`	Nome do arquivo de índice (padrão: **`index.html`**)
`fallthrough`	Passar para próximo middleware em 404

Templates

Configuração do View Engine

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

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

Engines Comuns

`ejs`	Templates JS embutido (**`<%= val %>`**)
`pug`	Baseado em indentação (antigo Jade)
`handlebars`	Estilo Mustache (**`{{val}}`**)

Router

Rotas 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 -> router's "/"
// GET /api/users/5 -> router's "/:id"
```

Métodos do Router

`router.get/post/put/delete`	Handlers de método HTTP
`router.use(fn)`	Middleware de nível de router
`router.param(name, fn)`	Pré-processar parâmetro de rota
`router.route(path)`	Encadear métodos em um caminho

Padrões de Autenticação

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();
};
```

Rotas Protegidas

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

Padrões Comuns

CORS

```
const cors = require("cors");
app.use(cors());                          // allow all origins
app.use(cors({ origin: "https://example.com" })); // restrict
```

Ambiente e Configuração

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

// Access env in routes
if (app.get("env") === "production") {
  app.use(helmet());
}
```

Pacotes npm Úteis

`cors`	Compartilhamento de recursos de origem cruzada
`helmet`	Headers de segurança
`morgan`	Logger de requisições HTTP
`cookie-parser`	Parsear header Cookie
`dotenv`	Carregar **`.env`** no **`process.env`**
`multer`	Dados de formulário multipart (upload de arquivos)