Référence rapide Express.js

Routage, middleware, requêtes, réponses, patterns

Configuration

Créer et démarrer le serveur

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

Middleware intégré

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

Routage

Méthodes 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));
```

Paramètres de route

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

Chaînes de requête

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

Middleware

Niveau application

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

Niveau 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 }));
```

Ordre d'exécution

`app.use(fn)`	S'exécute sur chaque requête (dans l'ordre)
`app.use(path, fn)`	S'exécute uniquement pour le préfixe de chemin correspondant
`next()`	Passe le contrôle au middleware suivant
`next(err)`	Passe au gestionnaire d'erreurs

Requête et réponse

Objet requête

`req.params`	Paramètres de route (**`/users/:id`**)
`req.query`	Chaîne de requête (**`?key=val`**)
`req.body`	Corps de requête analysé (nécessite un parser)
`req.headers`	Objet des en-têtes de requête
`req.method`	Méthode HTTP (GET, POST, ...)
`req.path`	Chemin de l'URL
`req.cookies`	Cookies (nécessite cookie-parser)

Objet réponse

`res.json(obj)`	Envoyer une réponse JSON
`res.send(body)`	Envoyer une chaîne / Buffer / objet
`res.status(code)`	Définir le statut HTTP (chaînable)
`res.redirect(url)`	Redirection 302 (ou passer le statut)
`res.sendFile(path)`	Envoyer un fichier comme réponse
`res.sendStatus(code)`	Envoyer le statut avec le texte par défaut
`res.set(header, val)`	Définir un en-tête de réponse

Gestion des erreurs

Middleware d'erreur

```
// 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 });
});
```

Définir les gestionnaires d'erreurs après tous les autres app.use() et routes

Erreurs asynchrones

```
// 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);
}));
```

Fichiers statiques

Servir un répertoire statique

```
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
```

Options

`dotfiles`	**`'ignore'`** | **`'allow'`** | **`'deny'`**
`maxAge`	Âge max de Cache-Control en ms
`index`	Nom du fichier index (défaut : **`index.html`**)
`fallthrough`	Passer au middleware suivant en cas de 404

Templates

Configuration du moteur de vues

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

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

Moteurs courants

`ejs`	Templates JS intégrés (**`<%= val %>`**)
`pug`	Basé sur l'indentation (anciennement Jade)
`handlebars`	Style Mustache (**`{{val}}`**)

Router

Routes modulaires

```
// 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;
```

Monter un router

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

Méthodes du router

`router.get/post/put/delete`	Gestionnaires de méthodes HTTP
`router.use(fn)`	Middleware au niveau du router
`router.param(name, fn)`	Pré-traitement des paramètres de route
`router.route(path)`	Chaîner les méthodes sur un seul chemin

Patterns d'authentification

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();
};
```

Routes protégées

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

Patterns courants

CORS

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

Environnement et configuration

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

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

Paquets npm utiles

`cors`	Partage de ressources cross-origin
`helmet`	En-têtes de sécurité
`morgan`	Logger de requêtes HTTP
`cookie-parser`	Analyser l'en-tête Cookie
`dotenv`	Charger **`.env`** dans **`process.env`**
`multer`	Données de formulaire multipart (téléversement de fichiers)