

Basic Server Setup

Express is a minimal, fast, and unopinionated web framework for Node.js.

```
import express from 'express';
const app = express();
const PORT = process.env.PORT || 3000;
```

```
// Built-in middleware
app.use(express.json()); // Parse JSON body
app.use(express.urlencoded({ extended: true }));
app.use(express.static('public')); // Serve static assets
```

```
app.get('/', (req, res) => {
  res.send('Hello, Express!');
});
```

```
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Routing & HTTP Methods

Basic Route Handlers

- `app.get('/path', (req, res) => { ... })`
- `app.post('/path', (req, res) => { ... })`
- `app.put('/path', (req, res) => { ... })`
- `app.patch('/path', (req, res) => { ... })`
- `app.delete('/path', (req, res) => { ... })`
- `app.all('/path', (req, res) => { ... })` (Matches all HTTP verbs)

Modular Router (`express.Router`)

```
// routes/users.js
import { Router } from 'express';
const router = Router();
```

```
router.get('/', (req, res) =>
  res.json({ users: [] }));
router.get('/:id', (req, res) => {
  const { id } = req.params;
  res.json({ id, name: 'Alice' });
});
```

```
export default router;
```

```
// app.js
import userRouter from './routes/users.js';
app.use('/api/users', userRouter);
```

Request & Response Objects

Request (`req`)

- `req.params`: Path parameters (`/users/:id` -> `req.params.id`)
- `req.query`: Query string parameters (`?page=2` -> `req.query.page`)
- `req.body`: Parsed request body payload
- `req.headers`: Headers object
- `req.cookies`: Parsed cookies (via `cookie-parser`)
- `req.ip` / `req.path` / `req.method`: Client metadata

Response (`res`)

- `res.status(200)`: Sets HTTP status code
- `res.json({ success: true })`: Sends JSON response
- `res.send('Text')`: Sends string or Buffer
- `res.sendStatus(404)`: Sends status code with default message
- `res.redirect('/new-path')`: Performs HTTP redirect
- `res.sendFile(path.resolve('file.pdf'))`: Transmits file
- `res.setHeader('X-Custom', 'val')`: Sets header property

Middleware Architecture

Functions with access to request, response, and next handler.

Application & Route Middleware

```
// Logging middleware
const logger = (req, res, next) => {
  console.log(`${req.method} ${req.url} - ${Date.now()}`);
  next(); // Pass control to next middleware
};
```

```
};
app.use(logger);
```

```
// Route-specific middleware
const requireAuth = (req, res, next) => {
  if (!req.headers.authorization) {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  next();
};
app.get('/api/secret', requireAuth, (req, res) => {
  res.json({ secret: 'data' });
});
```

Error-Handling Middleware (4 arguments)

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({
    error: err.message || 'Internal Server Error',
  });
});
```

Essential Third-Party Middleware

- **cors**: Cross-Origin Resource Sharing


```
import cors from 'cors';
app.use(cors({ origin: 'http://localhost:3000' }));
```
- **helmet**: Security headers


```
import helmet from 'helmet';
app.use(helmet());
```
- **morgan**: HTTP request logging


```
import morgan from 'morgan';
app.use(morgan('dev'));
```
- **cookie-parser**: Cookie header parsing


```
import cookieParser from 'cookie-parser';
app.use(cookieParser('secret-key'));
```