Express.js Cheatsheet

A comprehensive cheat sheet for Express.js, covering essential concepts, middleware, routing, and common tasks for building web applications and APIs.

Core Concepts & Setup

Basic Setup

Installation:

```
npm install express --save
```

Basic App Structure:

```
const express = require('express');
const app = express();
const port = 3000;

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

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

Running the App:

```
node <your_app_file>.js
```

Middleware Basics

`app.use(middleware)`	Applies middleware to all routes. Middleware functions have access to the request object (`req`), the response object (`res`), and the `next()` middleware function in the application's request-response cycle.
`next()`	A function to pass control to the next middleware function. Crucial for chaining middleware.

Example:

```
app.use((req, res, next) => {
  console.log('Time: ', Date.now());
  next();
});
```

Request and Response Objects

`req.params`	Access route parameters (e.g., `/users/:userId`).
`req.query`	Access query string parameters (e.g., `/search?q=term`).
`req.body`	Access the request body (requires body-parsing middleware).
`res.send(body)`	Sends the HTTP response. The `body` can be a string, an object, or an array.
`res.json(json)`	Sends a JSON response.
`res.status(code)`	Sets the HTTP status code.

Routing

Basic Routing

Route Methods:

```
app.get('/', (req, res) => { ... });
// GET route
app.post('/', (req, res) => { ... });
// POST route
app.put('/:id', (req, res) => { ... });
// PUT route
app.delete('/:id', (req, res) => { ... }); // DELETE route
```

Route Parameters:

```
app.get('/users/:userId', (req, res) => {
  res.send(`User ID: ${req.params.userId}`);
});
```

Chaining Route Handlers:

```
app.route('/book')
  .get((req, res) => { ... })
  .post((req, res) => { ... })
  .put((req, res) => { ... });
```

Route Paths

`/`	Matches only the path `/`.
`/*/`	Matches any single character except `/`.
`/users/:userId?`	The `?` after userId makes the route parameter optional.
`/:username([a-zA-Z]+)`	Matches only if username contains letters.

Using `next()` in Routes

```
app.get('/example', (req, res, next) => {
  console.log('First handler');
  next();
}, (req, res) => {
  console.log('Second handler');
  res.send('Hello from example!');
});
```

This allows you to define multiple handlers for a single route, performing different operations in sequence.

Middleware

Built-in Middleware

`express.static(root, [options])`	Serves static files (e.g., images, CSS, JavaScript) from a directory. `root` specifies the root directory from which to serve static assets. `app.use(express.static('public'));`
`express.json([options])`	Parses incoming requests with JSON payloads and is based on `body-parser`. `app.use(express.json());`
`express.urlencoded([options])`	Parses incoming requests with URL-encoded payloads and is based on `body-parser`. `app.use(express.urlencoded({ extended: true }));`

Third-Party Middleware

Examples:

- `morgan`: HTTP request logger middleware.
- `cors`: Enable Cross-Origin Resource Sharing.
- `helmet`: Helps secure Express apps by setting various HTTP headers.
- `cookie-parser`: Parse Cookie header and populate `req.cookies`.

Example with Morgan:

```
npm install morgan
```

```
const morgan = require('morgan');
app.use(morgan('dev')); // 'dev' is a predefined format
```

Custom Middleware

You can create your own middleware functions to handle specific tasks.

```
const myLogger = (req, res, next) => {
  console.log('LOGGED');
  next();
};

app.use(myLogger);
```

Middleware can also be applied to specific routes:

```
app.get('/profile', myLogger, (req, res) => {
  res.send('Profile Page');
});
```

Advanced Topics

Error Handling

Express comes with a built-in error handler. To use it, you simply pass an `Error` object to `next()`:

```
app.get('/error', (req, res, next) => {
  const err = new Error('This is an error!');
  next(err);
});

app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Remember to place the error-handling middleware after all other middleware and route handlers.

Template Engines

Setting up EJS	`npm install ejs`
	`app.set('view engine', 'ejs');`
	`app.get('/template', (req, res) => {`
	`  res.render('index', { name: 'World' }); // renders views/index.ejs`
	`});`

Popular Engines	Besides EJS, other template engines include Pug (formerly Jade), Handlebars, and Mustache.

Best Practices

- **Security:** Use Helmet to secure HTTP headers, and sanitize user inputs to prevent XSS and SQL injection attacks.
- **Configuration:** Use environment variables for configuration.
- **Logging:** Use a logging library like Winston or Bunyan for structured logging.
- **API Versioning:** Version your APIs to maintain backward compatibility.