Koa.js Cheatsheet

A comprehensive cheat sheet for Koa.js, covering essential concepts, middleware, routing, and common functionalities.

Koa.js Fundamentals

Basic Setup

Import Koa:

```
const Koa = require('koa');
const app = new Koa();
```

Starting the server:

```
app.listen(3000);
console.log('App listening on port 3000');
```

Basic middleware:

```
app.use(async ctx => {
  ctx.body = 'Hello Koa';
});
```

Context Object

`ctx.request`	Koa Request object (similar to Node's `req` but with added functionality).
`ctx.response`	Koa Response object (similar to Node's `res` but with added functionality).
`ctx.body`	Shorthand for setting response body (`ctx.response.body`).
`ctx.status`	Shorthand for setting response status (`ctx.response.status`).
`ctx.params`	Object containing route parameters.

Middleware

Middleware is a function that will be invoked on every request.

```
app.use(async (ctx, next) => {
  console.log('Middleware 1');
  await next();
  console.log('Middleware 1 after');
});
```

Middleware execution order is determined by the order in which they are added to the app.

Use `await next()` to pass control to the next middleware in the chain.

Routing in Koa

Basic Routing

Using `koa-router`:

```
const Router = require('koa-router');
const router = new Router();

router.get('/', async ctx => {
  ctx.body = 'Hello World!';
});

app.use(router.routes()).use(router.allowedMethods());
```

Route parameters:

```
router.get('/users/:id', async ctx => {
  ctx.body = `User ID: ${ctx.params.id}`;
});
```

HTTP Methods

`router.get(path, middleware)`	Handles GET requests.
`router.post(path, middleware)`	Handles POST requests.
`router.put(path, middleware)`	Handles PUT requests.
`router.delete(path, middleware)`	Handles DELETE requests.
`router.patch(path, middleware)`	Handles PATCH requests.

Route Middleware

Applying middleware to specific routes:

```
const authMiddleware = async (ctx, next) => {
  // Authentication logic
  await next();
};

router.get('/protected', authMiddleware, async ctx => {
  ctx.body = 'Protected resource';
});
```

Working with Request & Response

Request Body

Parsing request body using `koa-bodyparser`:

```
const bodyParser = require('koa-bodyparser');
app.use(bodyParser());

router.post('/submit', async ctx => {
  const data = ctx.request.body;
  console.log('Received data:', data);
  ctx.body = 'Data received';
});
```

Ensure `koa-bodyparser` is installed: `npm install koa-bodyparser`

Response Headers

`ctx.set(field, value)`	Sets a response header field.
`ctx.get(field)`	Gets a request header field.
`ctx.type = value`	Sets the Content-Type response header.
`ctx.attachment(filename)`	Sets the Content-Disposition to 'attachment' to signal the client to download the file.

Cookies

Setting cookies:

```
ctx.cookies.set('myCookie', 'cookieValue', {signed: true});
```

Getting cookies:

```
const cookieValue = ctx.cookies.get('myCookie', {signed: true});
```

Signed cookies require setting a `keys` property on the Koa app instance:

```
app.keys = ['secret-key'];
```

Error Handling

Basic Error Handling

Using `try...catch` blocks:

```
app.use(async (ctx, next) => {
  try {
    await next();
  } catch (err) {
    ctx.status = err.status || 500;
    ctx.body = err.message;
    ctx.app.emit('error', err, ctx);
  }
});
```

Emitting errors for centralized logging:

```
app.on('error', (err, ctx) => {
  console.error('Server error', err, ctx)
});
```

Custom Error Pages

`ctx.throw(status, message)`	Throws an error with the specified status and message. Example: `ctx.throw(404, 'Not Found')`
`ctx.status = 404; ctx.body = 'Page Not Found';`	Alternative way to set status and body for custom error pages.

Error Middleware

Example error-handling middleware:

```
app.use(async (ctx, next) => {
  try {
    await next();
    if (ctx.status === 404) {
      ctx.throw(404, 'Page Not Found');
    }
  } catch (err) {
    ctx.status = err.status || 500;
    ctx.body = `Error ${ctx.status}: ${err.message}`;
    ctx.app.emit('error', err, ctx);
  }
});
```