Node.js with Express Cheatsheet

1 Setting Up a Basic Server

Initialize an Express server and listen for connections.

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

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

2 Routing

Define routes for HTTP methods (GET, POST, PUT, DELETE).

```
// GET request
app.get('/users', (req, res) => {
  res.json({ users: [{ id: 1, name: 'Alice' }] });
});

// POST request
app.post('/users', (req, res) => {
  const user = req.body;
  res.status(201).json({ message: 'User created', user });
});

// PUT request
app.put('/users/:id', (req, res) => {
  const id = req.params.id;
  res.json({ message: `User ${id} updated` });
});

// DELETE request
app.delete('/users/:id', (req, res) => {
  const id = req.params.id;
  res.json({ message: `User ${id} deleted` });
});
```

3 Middleware Functions

Middleware processes requests before reaching route handlers.

```
// Built-in middleware for JSON parsing
app.use(express.json());

// Custom middleware
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next(); // Pass control to next middleware
});

```

```
// Route-specific middleware
const auth = (req, res, next) => {
  if (req.headers.authorization) {
    next();
  } else {
    res.status(401).json({ error: 'Unauthorized' });
  }
};
app.get('/protected', auth, (req, res) => {
  res.json({ message: 'Protected data' });
});
```

4 Serving Static Files

Serve static assets like HTML, CSS, or images.

```
app.use(express.static('public')); // Serves files from 'public' folder
// Access at http://localhost:3000/image.jpg
```

5 Handling Request/Response Objects

Access and manipulate request and response data.

```
app.get('/example', (req, res) => {
  // Request data
  const query = req.query; // e.g., ?name=Alice
  const params = req.params; // e.g., /example/:id
  const body = req.body; // POST/PUT data

  // Response methods
  res.status(200); // Set status code
  res.json({ message: 'Success' }); // Send JSON
  res.send('Plain text'); // Send text
  res.redirect('/new-route'); // Redirect
});
```

6 Error Handling

Handle errors with dedicated middleware.

```
// Error-handling middleware (must have 4 parameters)
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});

// Trigger error
app.get('/error', (req, res, next) => {
  next(new Error('Test error'));
});
```

7 Working with JSON and URL Parameters

Handle JSON payloads and dynamic URL parameters.

```
// URL Parameters
app.get('/users/:id', (req, res) => {
  const id = req.params.id; // e.g., /users/123
  res.json({ userId: id });
```

```
});

// JSON Request Body
app.post('/data', express.json(), (req, res) => {
  const data = req.body; // e.g., { name: "Bob" }
  res.json({ received: data });
});
```

8 Best Practices

- Use `express.json()` to parse JSON request bodies.
- Organize routes into separate files for large apps.
- Use environment variables (e.g., via `dotenv`) for configuration.
- Always include error-handling middleware.
- Validate input data to prevent security issues.
- Use `async/await` with try-catch for async routes.