Express.js Cheat Sheet in Simple Terms

What is Express.js?

Definition and Purpose: Express.js is a web application framework for Node.js, designed for building web applications and APIs. It's known for being minimalist, flexible, and providing robust routing capabilities.

Key Features:

- **Minimalist**: Express is lightweight and doesn't dictate how you should structure your application.
- **Flexible**: It has a robust set of features that can be extended with additional modules.
- **Robust Routing:** Allows you to manage different HTTP routes via a simple and intuitive API.

Setting up Express.js

Installing Node.js and npm: Install Node.js from its official website, which includes npm (node package manager).

Installing Express.js via npm: Example: Run npm install express in your project directory to add Express.js.

Creating a Basic Express.js Server:

Example :

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

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

app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

Basic Express Application Structure

app.js File Overview: This is the entry point of most Express applications.

Explanation of require, app**, and** listen **methods:**

- require('express') imports the Express module.
- app.listen(port, callback) starts the server on a specified port.

Creating Routes

Defining Simple Routes (GET, POST, PUT, DELETE:

Example : Define a route to respond to GET requests.

```
app.get('/about', (req, res) => {
  res.send('About Page');
});
app.get('/about', (req, res) => {
  res.send('About Page');
});
```

Example : Access dynamic parameters and query strings.

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

Middleware in Express.js

What is Middleware?: Functions that have access to the request object, response object, and the next middleware function.

Types of Middleware:

- **Application-level Middleware :** Applied on the app level using app.use() and app.METHOD().
- **Router-level Middleware :** Similar to application-level but bound to an instance of express.Router().
- **Error-handling Middleware :** Handles errors, defined with four arguments instead of three.

How to Use next() in Middleware:

Example : Using next() to pass control to the next middleware function.

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

Serving Static Files

Using express.static():

Example : Serve all static files from a public folder.

Handling HTTP Requests

Accessing Request Data:

Example : Access query parameters and URL parameters.

```
app.post('/login', (req, res) => {
const username = req.query.username; // or
req.body.username
res.send(`Username: ${username}`);
});
```

Accessing Request Data:

Example : Send different types of responses.

```
app.get('/data', (req, res) => {
  res.status(200).json({ message: "Data received" });
});
```

Handling Forms in Express

Parsing Form Data:

Example : Using middleware to parse URL-encoded data.

```
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
```

Error Handling

Creating Basic Error-handling Middleware:

Example : Custom error handling.

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

Express Router

- **What is express.Router()? : A modular way to handle routes.**
- **Structuring Your App with Routers:**

Example : Creating modular routes with express.Router().

```
const router = express.Router();
router.get('/about', (req, res) => {
  res.send('About this page');
});
app.use('/', router);
```

Template Engines

Using a Template Engine like Pug or EJS:

Example : Render HTML with dynamic data using EJS.

```
app.set('view engine', 'ejs');
app.get('/profile', (req, res) => {
  res.render('profile', { name: 'John Doe' });
});
```

Environment Variables

Using dotenv to Manage Environment Variables:

Example : Set and access environment variables.

```
require('dotenv').config();
const port = process.env.PORT || 8000;
```

Connecting to a Database

Using Express.js with Databases:

Example : Connecting to MongoDB with Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/mydatabase', {
useNewUrlParser: true, useUnifiedTopology: true });
```

Deploying an Express App

Basic Steps for Deploying:

Example : Deploy on Heroku.

```
git add .
git commit -m "Prepare for deployment"
git push heroku master
```

Express.js Best Practices

- **Folder Structure:** Organize your application with folders for controllers, routes, views.
- **Error Handling, Logging, and Security Practices:** Implement error handling strategies, use logging libraries like morgan, and secure your app with `helmetCertainly! Here's a structured cheat sheet for Express.js, breaking down key topics with simple explanations and examples.