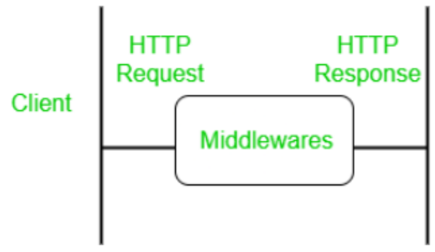

Middleware working

Middleware functions are the building blocks of any web server, especially in frameworks like ExpressJS. It plays a vital role in the request-response cycle.

Syntax

```
app.use((req, res, next) => {
  console.log('Middleware executed');
  next();
});
```

- **(req, res, next) => {}:** This is the middleware function where you can perform actions on the request and response objects before the final handler is executed.
- **next():** This function is called to pass control to the next middleware in the stack if the current one doesn't end the request-response cycle.

Types of Middleware

ExpressJS offers different types of middleware and you should choose the middleware based on functionality required.

- **Application-level middleware:** Bound to the entire application using app.use() or app.METHOD() and executes for all routes.
- **Router-level middleware:** Associated with specific routes using router.use() or router.METHOD() and executes for routes defined within that router.
- **Error-handling middleware:** Handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
- **Built-in middleware:** Provided by Express (e.g., express.static, ExpressJSon, etc.).
- **Third-party middleware**: Developed by external packages (e.g., body-parser, morgan, etc.).

Steps to Implement Middleware in Express

Step 1: Initialize the Node.js Project

```
npm init -y
```

Step 2: Install the required dependencies.

```
npm install express
```

Step 3: Set Up the Express Application

```
// Filename: index.js
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('<div><h2>Welcome to Gowtham</h2><h5>working Middleware</h5></div>');
});

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

Step 4: Start the Application:

```
node index.js
```

Output:

When you navigate to http://localhost:3000/, you will see: