MIDDLEWEAR

What Is Middleware?

middleware functions are functions that have access to the request object (`req`), the response object (`res`), and the next middleware function in the application's request-response cycle. Middleware functions can perform various tasks, modify the request and response objects, end the request-response cycle, and call the next middleware function in the stack.

Functions of Middleware:

1. Execution of Code:Middleware functions can execute code to perform specific tasks.

2. **Modify Request-Response Objects: Middleware can modify the request and response objects, adding or altering properties.**

3. End the Request-Response Cycle:Middleware can end the request-response cycle by sending a response to the client.

4. **Call the Next Middleware:Middleware functions use the `next` function to pass control to the next middleware in the stack.

`app.use()` in an Express app:

The `app.use()` function in Express is used to mount middleware functions in the application's request-processing pipeline. It is a way to bind middleware to a specific path or use it globally for all routes. Middleware functions added with `app.use()` are executed in the order they are added.

Example:

- **Application-level Middleware Example:**

```
app.use((req, res, next) => {
  console.log('This middleware runs for every request');
  next();
});
```

Types of Middlewares in Express:

1. Application-level Middleware:

Definition: Application-level middleware is a middleware that is bound to the entire Express application. It is executed for every incoming request.

- **Applied to the entire application.**
- **Examples: Logging, authentication.**

2. Router-level Middleware:

Definition: Router-level middleware is applied to specific routes or a group of routes defined using `express.Router()`. It is executed only when a request matches the specified route(s).

- **Applied to specific routes or a group of routes.**
- **Examples: Authentication for a specific route.**

3. Error-handling Middleware:

Definition: Error-handling middleware is responsible for handling errors that occur during the request-response cycle. It takes four parameters, including the `err` parameter representing the error.

- **Specialized middleware for handling errors.**
- **Examples: Custom error logging, error responses.**

4. Built-in Middleware:

Definition: Built-in middleware comes pre-packaged with Express. These are functionalities that can be used out of the box without the need for additional installation.

- **Included with Express without additional installation.**
- **Examples: `express.static` for serving static files.**

5. Third-party Middleware:

Definition: Third-party middleware are external modules that can be added to an Express application to extend its functionality. They are not built into Express but can be easily integrated.

- External modules added to extend functionality.
- Examples: `body-parser` for parsing request bodies.

Examples:

- Application-level Middleware Example:

```
app.use((req, res, next) => {
  console.log('This middleware runs for every request');
  next();
});
```

- Router-level Middleware Example:

```
const router = express.Router();

router.use((req, res, next) => {
  console.log('This middleware runs for routes defined in this router');
  next();
});

router.get('/example', (req, res) => {
  res.send('Example route');
});

app.use('/api', router);
```

- Error-handling Middleware Example:

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

- **Built-in Middleware Example:**

```
app.use(express.static('public'));
```

- **Third-party Middleware Example:**

```
const bodyParser = require('body-parser');
app.use(bodyParser.json());
```