MIDDLEWARE

What Is Middleware?

A request handler with access to the application's request-response cycle is known as middleware. It's a function that holds the request object, the response object, and the middleware function. Middleware can also send the response to the server before the request.

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.

Type of middleware:

1. **Application-level middleware.**

   Application-level middleware is executed for every request to the application. It is defined using app.use().

2. **Router-level middleware.**

   Router-level middleware is similar to application-level middleware but is bound to a specific route. It is used with express.Router().

3. **Build-in middleware.**

   Express comes with built-in middleware that can be used without installing additional packages.

4. **Error-handling middleware.**

Error-handling middleware is used to handle errors that occur during the request-response cycle.

5. **Third-party middleware.**

Third-party middleware are external modules that can be added to an Express application for additional functionality.

Examples:

*** login middleware:**

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

// Application-level middleware
app.use((req, res, next) => {
  console.log(Request received at ${new Date()});
  next(); // Pass control to the next middleware or route handler
});

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

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

2.Router-Level Middleware:

```
const express = require('express');
const app = express();
const router = express.Router();

// Router-level middleware
router.use((req, res, next) => {
  // Check authentication
  if (req.isAuthenticated()) {
    return next();
  }
  res.redirect('/login');
});

router.get('/dashboard', (req, res) => {
  res.send('Welcome to the dashboard!');
});
```

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

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

3.Error-Handling Middleware:

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

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

app.get('/', (req, res, next) => {
   // Simulate an error
   next(new Error('An error occurred'));
});

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

4.**Built-in Middleware:**

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

// Built-in middleware
app.use(express.static('public'));

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

5. Third-Party Middleware:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

// Third-party middleware
app.use(bodyParser.json()); // Parse incoming JSON requests

app.post('/api/data', (req, res) => {
const data = req.body;
 res.json(data);
```

```
    });

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