Third Party Modules

In Node.js, third-party modules refer to modules that are not built into the Node.js runtime itself but are developed and maintained by the community or third-party developers.
These modules can be easily integrated into Node.js applications using package managers such as npm (Node Package Manager) or yarn.

How to use third-party modules in Node.js

- **Install the Module:** Use npm or yarn to install the desired module.
  For example:
  ```
  npm install <module-name>
  or
  yarn add <module-name>
  ```

- **Require the Module:** After installing, require the module in your Node.js application to start using its functionality.
  For example:
  ```
  const module = require('<module-name>');
  ```

- **Use the Module:** Once required, you can use the functions, classes, or other features provided by the module in your application code.

- **Manage Dependencies:** If the module you're using has its own dependencies, npm or yarn will automatically install them for you. You can view and manage dependencies in the package.json file of your project.

- **Update and Maintain:** Regularly update the modules in your project to benefit from bug fixes, security patches, and new features. You can use commands like npm update or yarn upgrade to update modules to their latest versions.

Popular third-party modules in the Node.js ecosystem cover a wide range of functionalities including web frameworks (Express.js), database connectors (Mongoose for MongoDB), utility libraries (Lodash), testing frameworks (Mocha, Jest), and much more.

Example : using axios for making an HTTP GET request in Node.js

First, run the following command to install the axios module

```
npm install axios
```

App.js

```
// Import the axios module
const axios = require('axios');

// Define the URL of the API you want to make a GET request to
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

// Function to fetch data from the API using axios
async function fetchData() {
  try {
    // Make an HTTP GET request using axios
    const response = await axios.get(apiUrl);
    // Log the response data to the console
    console.log('Response:', response.data);
  } catch (error) {
    // If there's an error with the request, log the error
    console.error('Error:', error.message);
  }
}

// Call the fetchData function to initiate the HTTP request
fetchData();
```

Now you can run your Node.js program by executing the following command:

```
Node App.js
```

Routing

- Routing in Node.js refers to the process of determining how an application responds to client requests to specific endpoints or URLs.
- In web applications, routes are used to map URLs to server-side code, allowing you to define the behavior and response for different HTTP methods (such as GET, POST, PUT, DELETE, etc.) and URL patterns.
- Routing functionality is typically provided by web frameworks like Express.js, although you can implement basic routing logic without a framework using Node.js's built-in HTTP module.
- In Node.js, handling routes typically involves using a web framework such as Express.js. Express.js is one of the most popular web frameworks for Node.js, known for its simplicity and flexibility. It provides a robust set of features for building web applications and APIs, including routing.

Key concepts related to routing in Node.js

- **Route Definition:** Routes are defined using HTTP methods (GET, POST, PUT, DELETE, etc.) and URL patterns. In Express.js, you define routes using methods like app.get(), app.post(), app.put(), app.delete(), etc.
- **URL Parameters:** Routes can contain dynamic segments, which are denoted by a colon (:) followed by a parameter name. These parameters can be accessed in your route handler functions using req.params.
- **Query Parameters:** Routes can also include query parameters, which are appended to the URL after a question mark (?). Query parameters can be accessed in your route handler functions using req.query.
- **Route Middleware:** Middleware functions can be used to execute code before or after route handler functions. Middleware can be applied to specific routes, all routes, or a subset of routes using app.use() or by passing middleware functions directly to route definitions.
- **Error Handling:** Routes can include error-handling middleware functions to catch errors that occur during request processing. Error-handling middleware is defined using a special signature with four arguments (err, req, res, next).
- **Route Order:** Route order matters in Express.js. Routes are evaluated in the order they are defined, so more specific routes (e.g., routes with URL parameters) should be defined before more general routes to avoid route conflicts.

Example : handle routes in Node.js using Express.js

First, you need to install Express.js using the following command

```
npm install express
```

App.js

```
// Import required modules
const express = require('express');

// Create an Express application
const app = express();

// Define routes
// GET request to the root URL '/'
app.get('/', (req, res) => {
  res.send('Hello, Students!');
});

// GET request to the '/about' route
app.get('/about', (req, res) => {
  res.send('Learn2Earn Labs  is the best training institute for Programming, Development & Business Management related training programs');
});

// POST request to the '/details' route
app.post('/details', (req, res) => {
  res.send('Contact on +91-9548868337 for more details and registration');
});

// Start the server
const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Example : Dynamic route with URL parameters

App.js

```
// Import required modules
const express = require('express');

// Create an Express application
const app = express();

// Middleware to log request method and URL
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});

// Define routes
// GET request to the root URL '/'
app.get('/', (req, res) => {
  res.send('Welcome to Learn2Earn Labs!');
});

// GET request to the '/about' route
app.get('/about', (req, res) => {
  res.send('Learn2Earn Labs is the best training institute for Programming, Development & Business Management related training programs');
});

// Dynamic route with URL parameters
app.get('/students/:id', (req, res) => {
  const studentId = req.params.id;
  res.send(`Student ID: ${studentId}`);
});

// POST request to the '/details' route
app.post('/details', (req, res) => {
  res.send('Contact on +91-9548868337 for more details and registration');
});
```

```
// Handling 404 errors (Page Not Found)
app.use((req, res) => {
  res.status(404).send('Page Not Found');
});

// Start the server
const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Example : Route with query parameters

App.js

```
const express = require('express');
const app = express();
// Define routes
// GET request to the root URL '/'
app.get('/', (req, res) => {
   res.send('Welcome to Learn2Earn Labs!');
 });
 // GET request to the '/about' route
 app.get('/about', (req, res) => {
   res.send('Learn2Earn Labs  is the best training institute for Programming, Development & Business Management related training programs'); } );
// Define route with query parameters
//pass /search?query=full stack web development
app.get('/search', (req, res) => {
  const { query } = req.query;
  res.send(`Search query: ${query}`);
});
// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Some issues in the flow!');
});
// Catch-all route for 404 errors
app.use((req, res) => {
  res.status(404).send('Page Not Found');
});
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

This is just a basic overview of routing in Node.js using Express.js. As your application grows, you may need to implement more complex routing logic, authentication, authorization, and other features to handle various use cases. Express.js provides a rich set of features and middleware to help you build robust web applications and APIs.