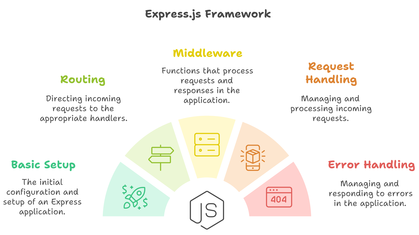
Introduction to Express.js Framework

This document provides a comprehensive overview of the Express.js framework, a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. We will cover the essentials of setting up a basic Express application, routing, middleware, request and response handling, and error handling. By the end of this document, you will have a solid foundation to start building applications using Express.js.



Introduction to Express.js

Express.js is a web application framework for Node.js designed for building web applications and APIs. It simplifies the process of creating server-side applications by providing a set of tools and features that streamline the development process. With its unopinionated nature, Express allows developers to structure their applications in a way that best suits their needs.

Setting up a Basic Express Application

To get started with Express.js, you need to have Node.js installed on your machine. Once you have Node.js set up, you can create a new Express application by following these steps:

1. **Initialize a new Node.js project**:

```
mkdir my-express-app
cd my-express-app
npm init -y
```

2. **Install Express**:

```
npm install express
```

3. **Create a basic server**:
   Create a file named **app.js** and add the following code:

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

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

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

4. **Run the application**:

```
node app.js
```

Now, you can open your browser and navigate to **http://localhost:3000** to see your basic Express application in action.

Routing

Routing in Express.js allows you to define different endpoints for your application. You can handle various HTTP methods (GET, POST, PUT, DELETE) and specify the corresponding actions. Here's an example of how to set up routing:

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

app.post('/submit', (req, res) => {
    res.send('Form Submitted');
});
```

You can define routes for different paths and methods, making it easy to manage the flow of your application.

Middleware

Middleware functions are functions that have access to the request and response objects and can modify them or end the request-response cycle. They are used for tasks such as logging, authentication, and error handling. Here's how to use middleware in Express:

```
app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next(); // Pass control to the next middleware
});
```

You can also use built-in middleware like **express.json()** to parse incoming JSON requests:

```
app.use(express.json());
```

Request and Response Handling

Express.js makes it easy to handle incoming requests and send responses. You can access request parameters, query strings, and body data. Here's an example of handling a POST request with JSON data:

```
app.post('/data', (req, res) => {
    const data = req.body;
    res.json({ message: 'Data received', data });
});
```

This allows you to create dynamic and interactive applications that respond to user input.

Error Handling

Error handling is crucial in any application. Express provides a simple way to handle errors using middleware. You can define an error-handling middleware function as follows:

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

This middleware will catch any errors that occur in your application and provide a response to the client.

Conclusion

Express.js is a powerful framework that simplifies the process of building web applications and APIs. By understanding the basics of setting up an application, routing, middleware, request and response handling, and error handling, you can create robust and efficient applications. With this foundation, you can explore more advanced features and build complex applications tailored to your needs.