Mastering Express Routing

A comprehensive guide to building robust web applications with Express.js.

Core HTTP Methods

Express.js leverages standard HTTP methods to define the fundamental operations for your API endpoints. Each method corresponds to a specific action on a resource.

GET

Retrieve data from the server.

```
app.get('/users', (req, res) => res.send('User List'));
```

POST

Send new data to the server to create a resource.

```
app.post('/users', (req, res) => res.send('User Created'));
```

PUT

Update an existing resource, typically replacing it entirely.

```
app.put('/users/:id', (req, res) => res.send('User Updated'));
```

DELETE

Remove a resource from the server.

```
app.delete('/users/:id', (req, res) => res.send('User Deleted'));
```

PATCH

Apply partial modifications to an existing resource.

```
app.patch('/users/:id', (req, res) => res.send('User Partially Updated'));
```

Dynamic Routing with Parameters & Wildcards

Capture dynamic values from URLs and match flexible patterns using route parameters and wildcards.

Route Parameters

Parameters like **:id** allow you to extract specific values from the URL path. This is crucial for routes targeting individual resources.

```
app.get('/users/:id', (req, res) => {
  res.send(`User ID is ${req.params.id}`);
});
```

Wildcards

Use the asterisk (*) to match any sequence of characters, enabling flexible route definitions for varying path segments.

```
app.get('/file/*', (req, res) => {
 res.send('Wildcard matched');
});
```

Query Strings & Request Body Parsing

Access data sent via URL query strings and efficiently parse request bodies for data submission.

Query Strings

Retrieve data from the URL's query component using **req.query**. Ideal for search parameters or optional data.

```
// Example URL: /search?name=ram
app.get('/search', (req, res) => {
  res.send(`Searching for ${req.query.name}`);
});
```

Parsing Request Bodies

Process incoming request bodies (e.g., JSON or URL-encoded data) with built-in Express middleware like **express.json()** and **express.urlencoded()**.

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

Route Handlers & Middleware Chains

Leverage multiple callback functions within a single route, creating a powerful chain of operations, similar to middleware.

ƒ(x)

Defining Callbacks

Each function in the chain receives the **req**, **res**, and **next** objects, allowing sequential processing.

Logging Example

```
const log = (req, res, next) => {
  console.log('Logged');
  next(); // Pass control to the next handler
};
```

Applying to a Route

```
app.get('/profile', log, (req, res) => {
 res.send('Profile Page');
});
```

Understanding Middleware Architecture

Middleware functions are the backbone of Express.js, executing crucial tasks during the request/response cycle.

1

Execution

They run sequentially during the HTTP request/response process.

2

Modification

Can modify request (**req**) and response (**res**) objects.

3

Control Flow

Can end the cycle by sending a response or pass control to the next middleware via **next()**.

Request & Response Handling

The **req** (request) and **res** (response) objects are central to interacting with incoming requests and sending back replies.

Request Object (req)

- **req.body**: Parsed body data (requires body-parser middleware).
- **req.params**: URL parameters captured from route.
- **req.query**: Query string parameters.
- **req.headers**: Request headers sent by the client.

Response Object (res)

- **res.send()**: Sends various types of responses (text, HTML).
- **res.json()**: Sends a JSON response.
- **res.status()**: Sets the HTTP status code.
- **res.redirect()**: Redirects the client to another URL.

Content Negotiation: Express can automatically select response types based on client **Accept** headers using **res.format()**.