Express.js Routing Lab Manual

Introduction to Express.js

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It facilitates the rapid development of Node based Web applications.

Key Features of Express.js:

- Robust routing
- HTTP helpers (redirection, caching, etc)

Getting Started with Express.js

Installation

To use Express.js, you first need to have Node.js installed on your system. Then, you can install Express using npm (Node Package Manager) using the command:

```
npm install express
```

Basic Express.js Application Structure

Here's a simple Express.js application:

```
const express = require('express');
const app = express();
const port = 3000;

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

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

This creates a server that listens on port 3000 and responds with "Hello World!" when you access the root URL.

Routing in Express.js

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, etc.).

Basic Route Structure

```
app.METHOD(PATH, HANDLER)
```

- `app` is an instance of express.
- `METHOD` is an HTTP request method, in lowercase.
- `PATH` is a path on the server.
- `HANDLER` is the function executed when the route is matched.

Route Methods

Express supports methods that correspond to all HTTP request methods: get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, and connect. The most common ones, however, are get, post, put and delete.

Route Paths

Route paths, in combination with a request method, define the endpoints at which requests can be made. Route paths can be strings, string patterns, or regular expressions.

Examples:

```
// This route path will match requests to the root route, /
app.get('/', (req, res) => {
  res.send('Root');
});

// This route path will match requests to /about
app.get('/about', (req, res) => {
  res.send('About');
});

// This route path will match acd and abcd
app.get('/ab?cd', (req, res) => {
  res.send('ab?cd');
});
```

Route Parameters

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the `req.params` object, with the name of the route parameter specified in the path as their respective keys.

```
app.get('/users/:userId/books/:bookId', (req, res) => {
  res.send(req.params);
});
```

Response Methods

The methods on the response object (`res`) in the examples above can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

Some common response methods:

- `res.json()`: Sends a JSON response
- `res.send()`: Sends a response of various types
- `res.sendFile()`: Sends a file as an octet stream
- `res.render()`: Renders a view template

Error Handling

In Express, error handling is done by writing middleware functions that take four arguments instead of three (err, req, res, next):

Example:

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

Helpful Resources

1. Express.js Official Documentation
2. MDN Web Docs - Express/Node introduction
3. Express.js Routing Guide