Implementing Error Handling in Node

Skills Network

Estimated Effort: 30 mins

Proper error handling makes code more maintainable and facilitates debugging. It helps developers identify and fix issues during development and maintenance phases. Express comes with a built-in error handler that takes care of any errors that might be encountered in the app. This default error-handling middleware function is added at the end of the middleware function stack.

Learning Objectives

In this lab you'll learn how to:

- Throw errors to handle unexpected behaviour.
- Handle errors that are thrown.
- Handle all errors thrown while accessing the endpoints.

Create node application

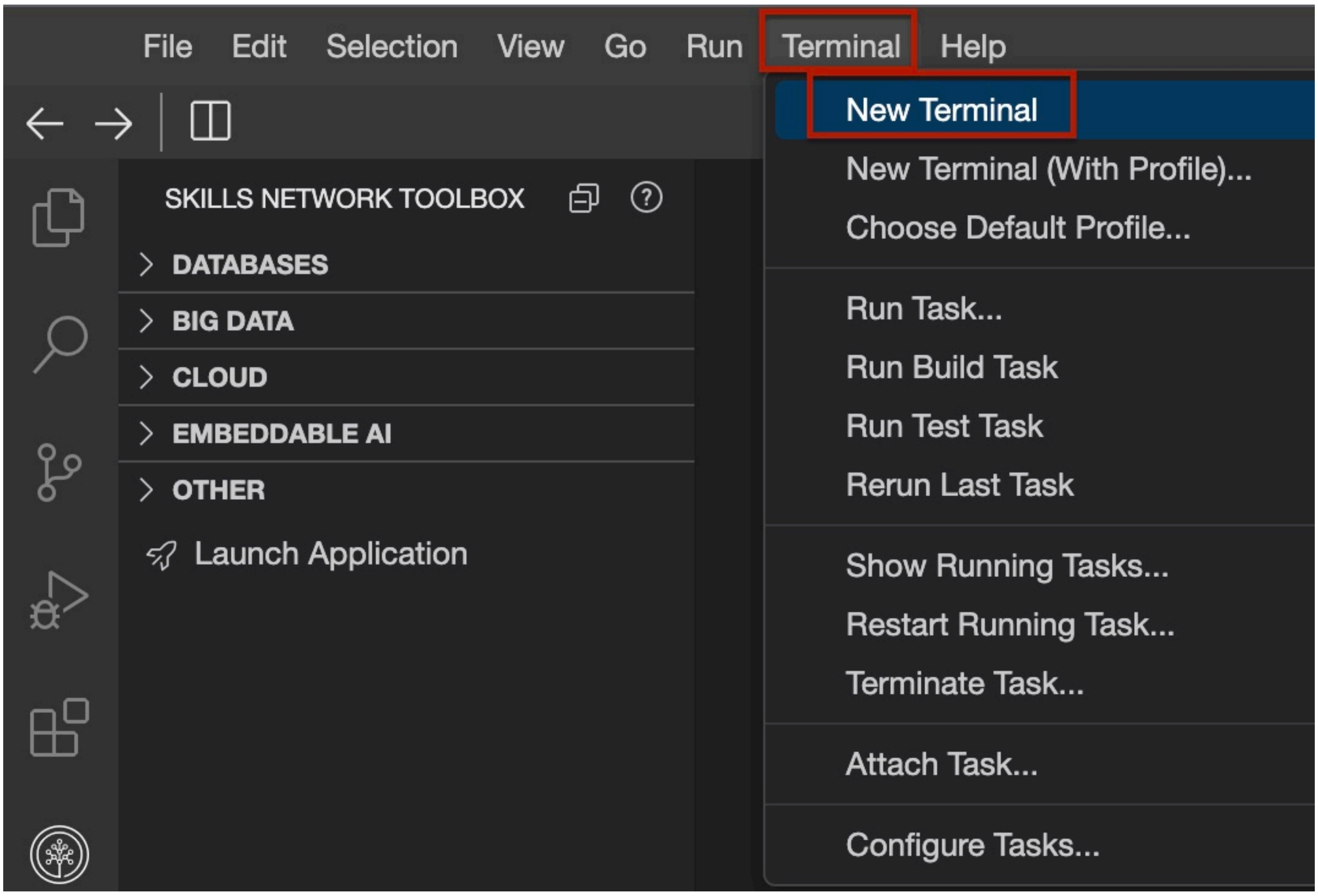
You will create an application and include steps for error handling. You have done a bit of error handling in the tasks done so far as part of this course as given below.

```
try {
  // Code that might throw an error
} catch (error) {
  // Handle the error
}
```

Copied!

These were done individually for each route. It could have been for handling erroneous input, an outcome of data validation, etc., Now you will learn to handle these globally at application level.

1. Open a new terminal.



2. Create a directory to add the code by running the following command.

```
1. mkdir serverapp
```

Copied! Executed!

3. Change to the newly created directory by running the following command.

```
1. cd serverapp
```

Copied! Executed!

4. Run the following command which initialized the directory as a node application. This will keep track of the dependencies that you will install for this application that you are going to create.

```
1. npm init
```

Copied! Executed!

Please choose appropriate values when prompted else press enter to leave the default values. The package.json will be created in the current directory `serverapp`.

> The lab environment is transient and the files and directories that you create will be deleted when you log out of the lab environment. But it is highly recommended that you follow these steps as good programming practice.

5. To provide API endpoints, we need to run a server. Express is Node.js web application framework that allows you to host services. run the following command to install `express`.

```
1. npm install express
```

Copied! Executed!

Create the application file

6. Run the following command to create a new file named `serverapp.js`.

```
1. touch serverapp.js
```

Copied!

7. Open the file `serverapp.js` in the editor and paste the following content.

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

const app = express();
const port = 3000;

// GET endpoint
app.get('/', async (req, res) => {
    res.send("This endpoint works.")
});

// GET endpoint
app.get('/squarenumber/:num', async (req, res) => {
    let x = req.params.num;
    res.json({"square":x*x});
});

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

Copied!

There are 3 endpoints in this application. The default route `/` is a normal endpoint which will return a string. The `/squarenumber/:num` endpoint will return the square if a number is passed. You should expect it to throw an error if the input is not a string.

Run the application and access endpoints

1. Run the serverapp.js from the terminal.

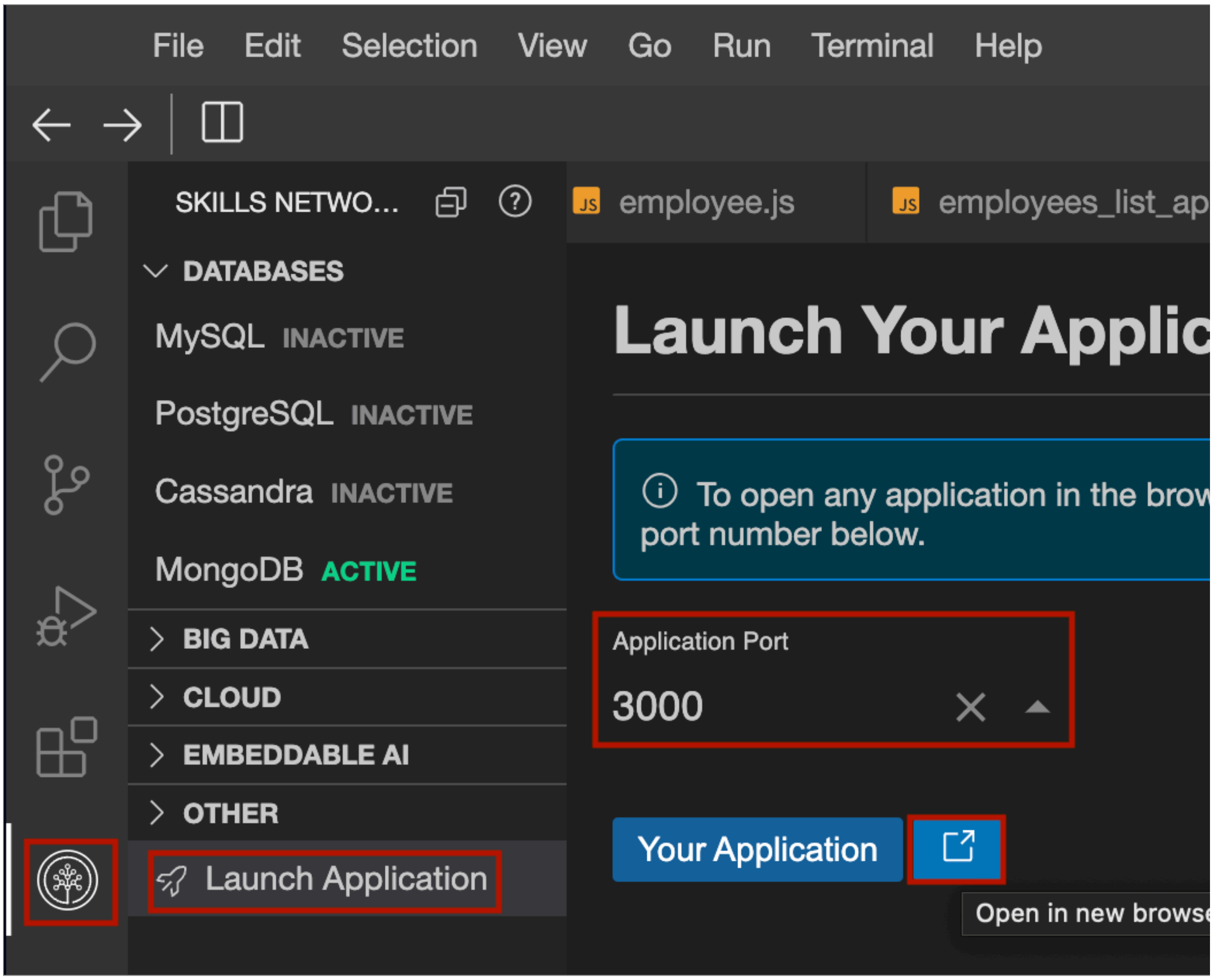
```
node serverapp.js
```

Copied! Executed!

This starts the express server on port `3000`.

2. Open the web application on port `3000` by click on the button below or through the `Launch Application` under the `SkillsNetwork tools`, as shown in the image below.

Server Application



Launch Your Applic

To open any application in the brow
port number below.

Application Port
3000

3. Append the URL in the browser with `/squarenumber/3` and see the output rendered.

4. Append the URL in the browser with `/squarenumber/three` and see the output rendered.

The output is not an error. It just returns `null`. To handle this, change the code to throw an error if the input is not a valid number.

5. Change the code for `/squarenumber/:num` api endpoint as below.

```
// GET endpoint
app.get('/squarenumber/:num', async (req, res) => {
    let x = req.params.num;
    if (isNaN(x)){
        throw Error("Input not a number")
    }

    res.json({"square":x*x});
});
```

Copied!

6. Stop the server pressing `Ctrl+c` and restart the server by running the following command.

```
node serverapp.js
```

Copied! Executed!

7. Click on the button below to open the application in a browser and try accessing the `/squarenumber/three` endpoint again.

Server Application

You will see an error message and the server stops.

Adding the error handler middleware

1. Add the following method to your code before starting the express application. This method will handle the error and send an appropriate message instead of letting the application crash.

```
app.use((err, req, res, next) => {
    // Set default values for status code and status if not provided in the error object
    err.statusCode = err.statusCode || 500;
    err.status = err.status || "Error";

    // Log the error stack to the console for debugging purposes
    console.log(err.stack);

    // Send a JSON response with formatted error details
    res.status(err.statusCode).json({
        status: err.status,
        message: err.message,
    });
});
```

Copied!

2. Change the code for `/squarenumber/:num` api endpoint as below.

```
// GET endpoint
app.get('/squarenumber/:num', async (req, res,next) => {
    let x = req.params.num;
    if (isNaN(x)){
        next(new Error("Input not a number")); // Pass the error to the next middleware
        return;
    }
    res.json({"square":x*x});
});
```

Copied!

3. Stop the server pressing `Ctrl+c` and restart the server by running the following command.

```
node serverapp.js
```

Copied! Executed!

4. Click on the button below to open the application in a browser and try accessing the `/squarenumber/three` endpoint again.

Server Application

You will see that the server doesn't crash, but you get the error message as a response. You can also go and see that the stack trace from the error was thrown is logged on the console. The difference between handling this for every route or function and globally is that, the error need not be handled over and over and it can be simply delegated using next to the global error handler.

Adding other function that use error handler middleware

1. Now add the following code to `serverapp.js`. This endpoint gives the cube of the parameter passed if it is a number. It throws an error object with custom status code and message if the parameter is not a number.

```
// GET endpoint
app.get('/cubenumber/:num', async (req, res,next) => {
    let x = req.params.num;
    if (isNaN(x)){
        const err = new Error('Invalid input');
        err.statusCode = 400;
        err.details = 'The input must be a number';
        next(err);
    } else {
        res.json({"cube":x*x*x});
    }
});
```

Copied!

The `/cubenumber/:num` endpoint will return the cube if a number is passed. It will pass the error object to the global error handler if the input is not a valid number.

2. Stop the server pressing `Ctrl+c` and restart the server by running the following command.

```
node serverapp.js
```

Copied! Executed!

3. Click on the button below to open the application in a browser and try accessing the `/cubenumber/3` endpoint again.

Server Application

It would return a JSON object showing the cube of 3.

4. Change the endpoint to `/cubenumber/three`. This will go through the check for number we have within the implementation and delegate the error to the error handler.

You will see that the server doesn't crash, but you get the error message as a response. The difference between handling this for every route or function and globally is that, in all the other functions, the error need not be handled. It can be delegated using next to the global error handler.

Practice Exercise

1. Add the following endpoint to the code that will return the character at a particular index for the given string. If the index passed is greater than the length of the string, it should raise and error and delegate it to the error handler.

`/getelementatindex/:mystr/:idx`

▶ Click here for the solution

Author(s)

Lavanya T S