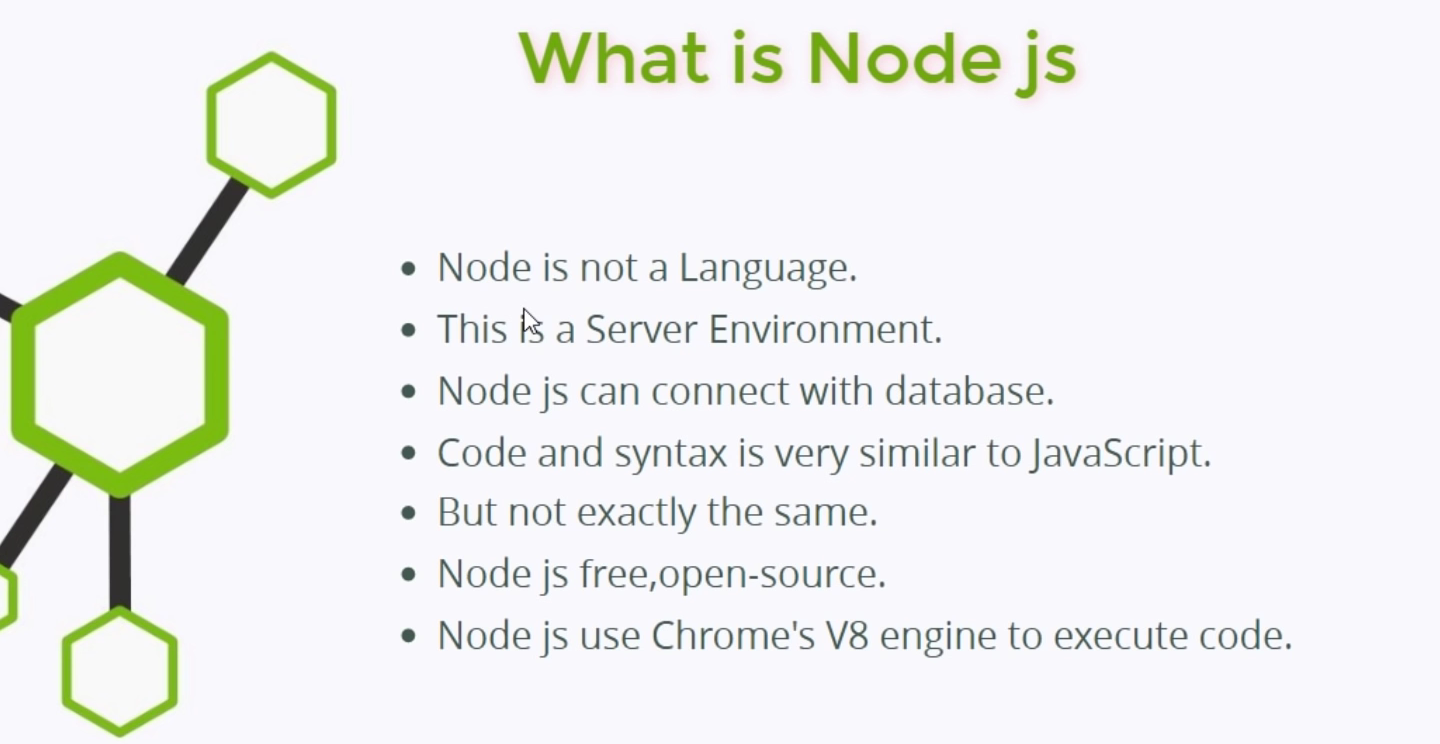
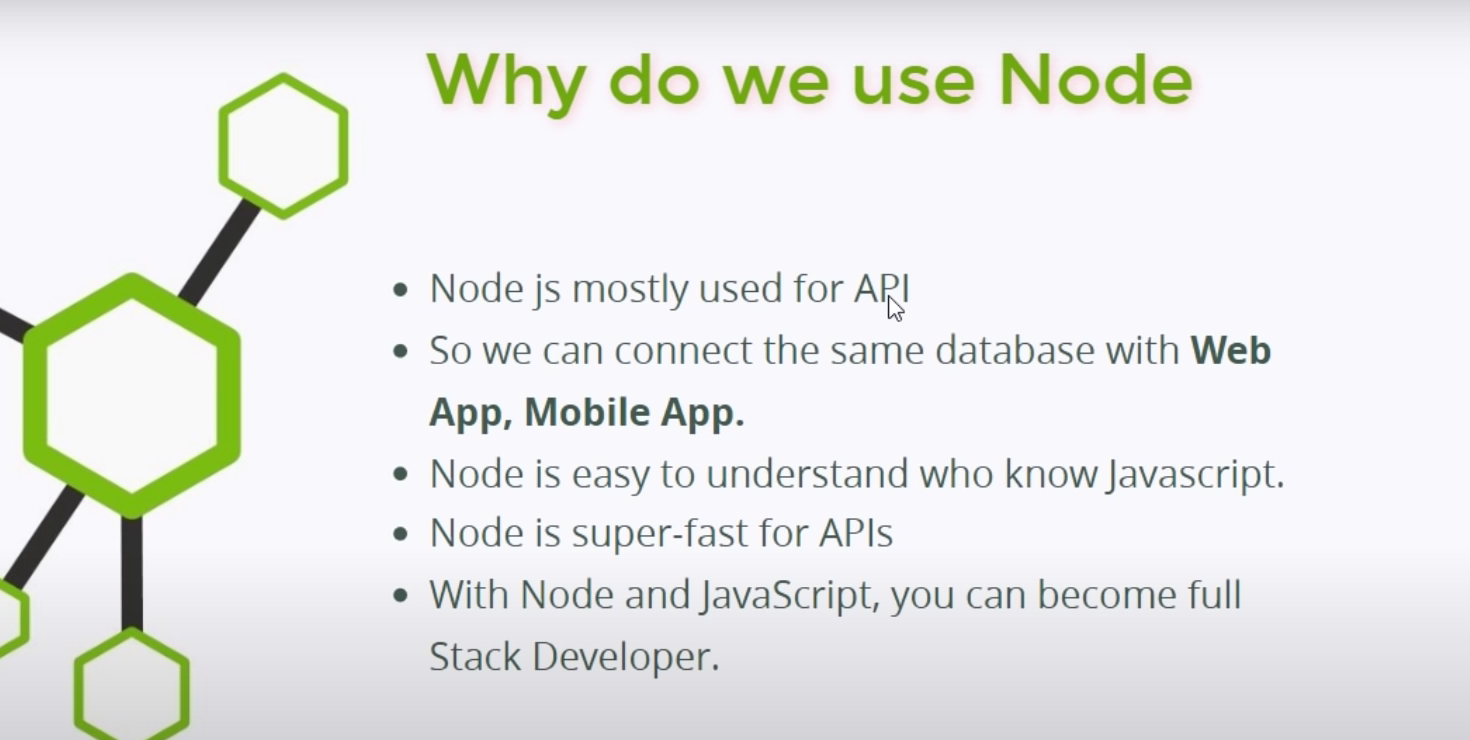
NODE JS



s

// To accept only one function from module

// This method is used to include all functions of fs module

// const fs = require("fs");

// But what if we want to use only one function in fs module

// We get only one function from fs module

const fs = require("fs").writeFileSync;

fs("fs.txt" , "My name is sumit. I am learning node js. Today day date is 10-05-2023.");

The package. Json:

The package. json file contains descriptive and functional metadata about a project, such as a name, version, and dependencies.

To create package.json file we have to enter command in terminal

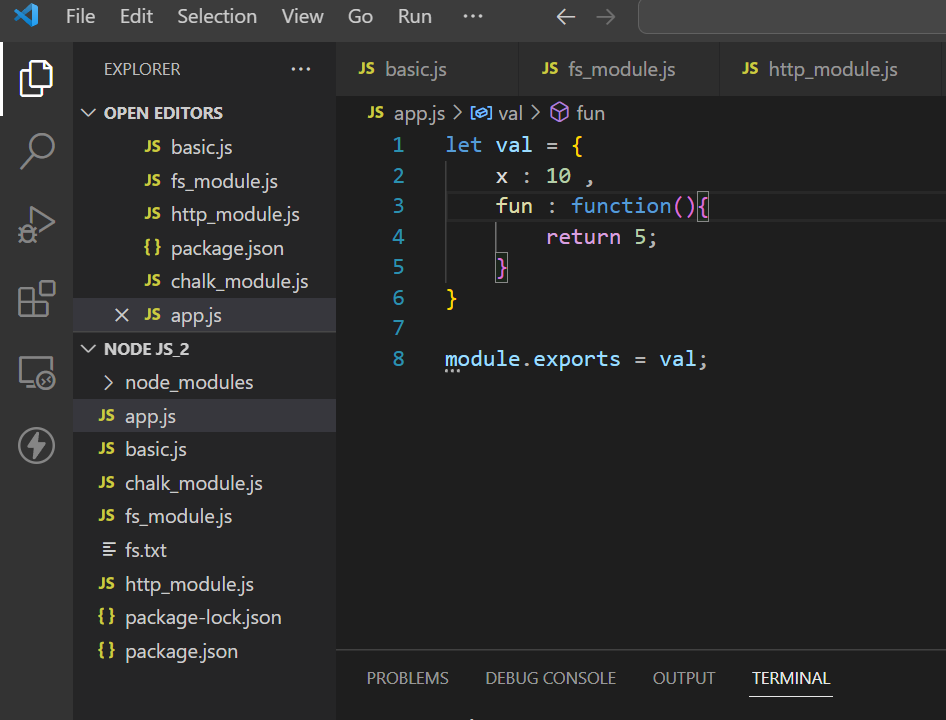
npm init

The npm init command is used to create a Node. js project.

The package-lock. json is a lockfile that holds information on the dependencies or packages installed for a node. js project, including their exact version numbers.

Node.js runs JavaScript code in a single thread, which means that your code can only do one task at a time.

Node\_modules Folder :





This node\_modules folder is automatically created when any module is installed.

We don’t need to push this folder to the github.

After installing so many packages the size of this folder goes in GB.

Because this folder get automatically installed according to the dependencies when we write this command in terminal:

npm install

* If we don’t want to push any folder to github then just create .gitignore file
* And write that file name like that :

/node\_modules

Package and module both have the same meaning.

nodemon package:

nodemon is a tool that helps develop Node. js based applications by automatically restarting the node application when file changes in the directory are detected. nodemon does not require any additional changes to your code or method of development. nodemon is a replacement wrapper for node.

To install nodemon command:

// npm i nodemon

To use nodemon

// npx nodemon “path”

* We can access in our web server only
* We can not access data from upper folder

single threaded or multithreaded:

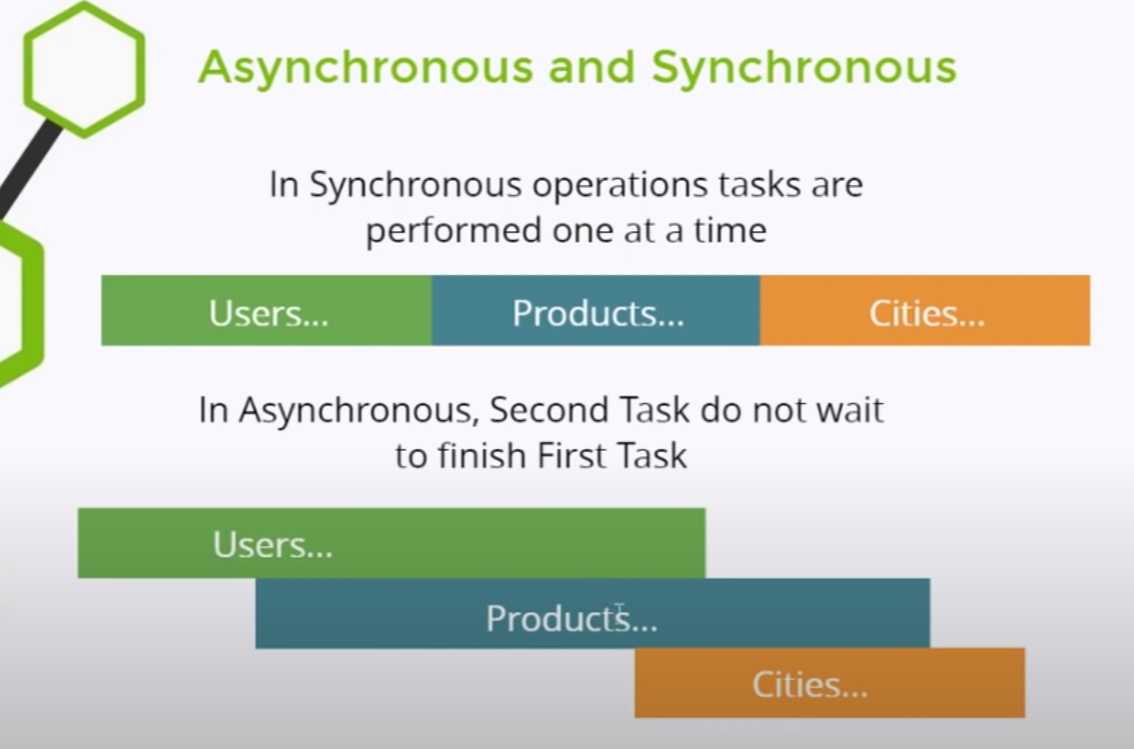
JavaScript is single threaded and only one line of code can be executed at any given time.

 Node.js itself is multithreaded.

Node.js runs JavaScript code in a single thread, which means that your code can only do one task at a time. However, Node.js itself is multithreaded

Node.js provides hidden threads through the libuv library, which handles I/O operations like reading files from a disk or network requests.

Synchronous Vs Asynchronous:



**Synchronous:**

In synchronous operation first the use task is completed. Then products task is completed. Then client task is completed.

**Asynchronous:**

In asynchronous all the tasks user, product and client are running simultaneously.

* NodeJS is an asynchronous.
* JavaScript is also asynchronous.

// How to make 404 page

// This is the method to create a 404 page

app.get('\*' , (req , res)=>{

    res.sendFile(`${filePath}/notvalid.html`);

})

ejs

* EJS is a simple templating language that lets you generate HTML markup with plain JavaScript.
* No religiousness about how to organize things.
* No reinvention of iteration and control-flow.
* It's just plain JavaScript.

It's easy to install EJS with NPM.

$ npm install ejs

This Syntax used to include another ejs html file in ejs html file

<%- include("common/header"); %>

This Syntax is used to add javascript content int ejs

<%= item %>

What if we want to use the middleware for single page:

1. First Step in ejs is to create views folder in file
2. Then create ejs files in that folder
3. Suppose login is in views folder then it automatically detect it

app.get("/login" , (req , res)=>{

    res.render("login");

})

Express.js

The **app.get()** function routes the HTTP GET Requests to the path which is being specified with the specified callback functions. Basically, it is intended for binding the middleware to your application.

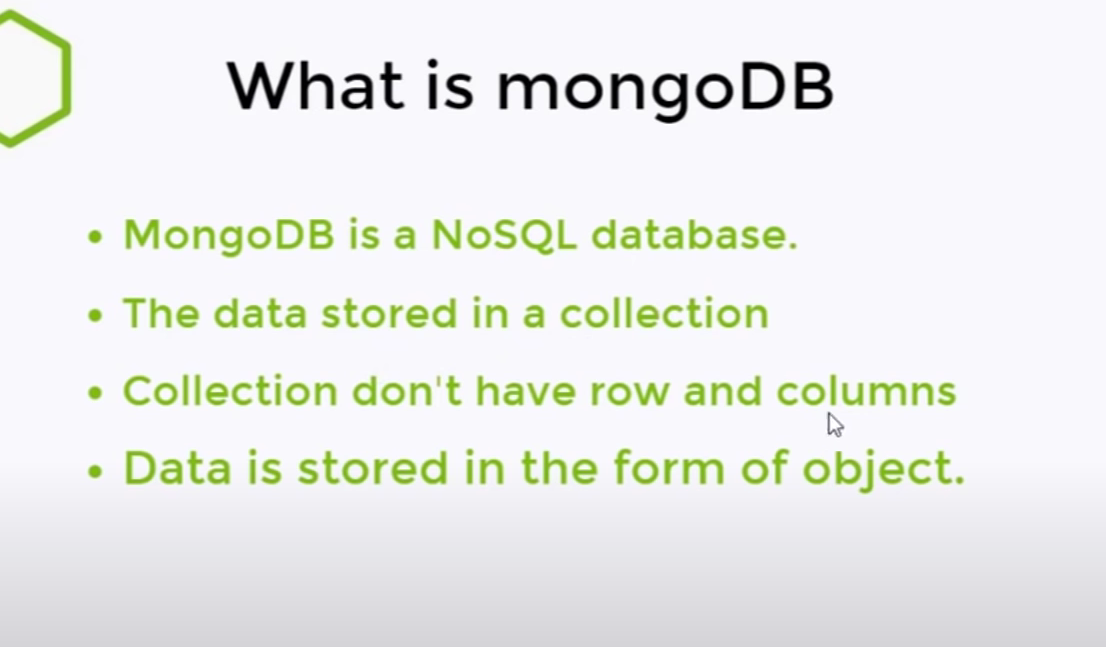
**Syntax:**

app.get( path, callback );

**Parameters:**

* **path:** It is the path for which the middleware function is being called.
* **callback:** They can be a middleware function or a series/array of middleware functions.

Mongo Db



Commands of Mongo DB:

// View all Databases

show dbs

// Create a new or switch database

use database\_name

// View current databse

db

// Delete database

db.dropDatabase()

// To view collection in database

show collections

// To create a new collection in database

db.createCollection("collection\_name")

// To delete Collection (collection\_name to delete)

db.collection\_name.drop()

/\* Commands for rows \*/

// Insert one row

db.collection\_name.insertOne({"name" : "Sumit",

"Age" : 19 , "Language" : "Javascript"})

// Insert many row : Pass The array of objects

db.collection\_name.insertMany([

{"name" : "Sumit","Age" : 19 , "Language" : "Javascript"},

{"name" : "Saurabh","Age" : 21 , "Language" : "C"},

{"name" : "Akash","Age" : 16 , "Language" : "C++"},

])

// show all rows in a collection

db.collection\_name.find()

// search in the mongodb

db.collection\_name.find({name : "Sumit"})

// Only three rows will be shown

db.collection\_name.find().limit(3)

// Only find first matching row in collection

db.collection\_name.findOne({name : "Sumit"})

// Count the number of rows in the collection

db.collection\_name.find().count()

// Count how many rows have name sumit

db.collection\_name.find({name : "Sumit"}).count()

// Sort rows in ascending order

db.collection\_name.find().sort({Age : 1})

// Sort rows in descending order

db.collection\_name.find().sort({Age : -1})

// Update a row

db.collection\_name.updateOne({name : "Sumit"} ,{$set : {Age : 24}})

// Update many row

// $inc (increment) operator

db.collection\_name.updateMany({name : "Sumit"} ,{$inc : {Age : 24}})

// if it cant find the data then it is as row added to collection {upsert : true}

db.collection\_name.updateOne({name : "Shubham"} ,{$set : {Age : 34}} , {upsert : true})

// $rename (Rename) operator

db.collection\_name.updateOne({name : "Sumit"} ,{$rename : {Language : "Coding\_language"}})

// To delete a row

db.collection\_name.removeOne({name : "Sumit"})

// To delete Many row

db.collection\_name.removeMany({name : "Sumit"})

Mongo DB With Node JS:

// To install mongodb in node js:

npm i mongodb

Mongoose

// Command to install mongoose

npm i mongoose

Advantages of mongoose:

* We can set schema in mongoose
* We can define types of the columns

Schema:

A Mongoose Schema defines the structure and property of the document in the MongoDB collection.

Error Handling

**Basics:**

It is necessary to handle the error. If error occurs in our middle of working program it interrupt the program.

try{

    console.log(sumit);

}

catch(error){

    console.warn("There is some problem : ");

    console.warn(error);

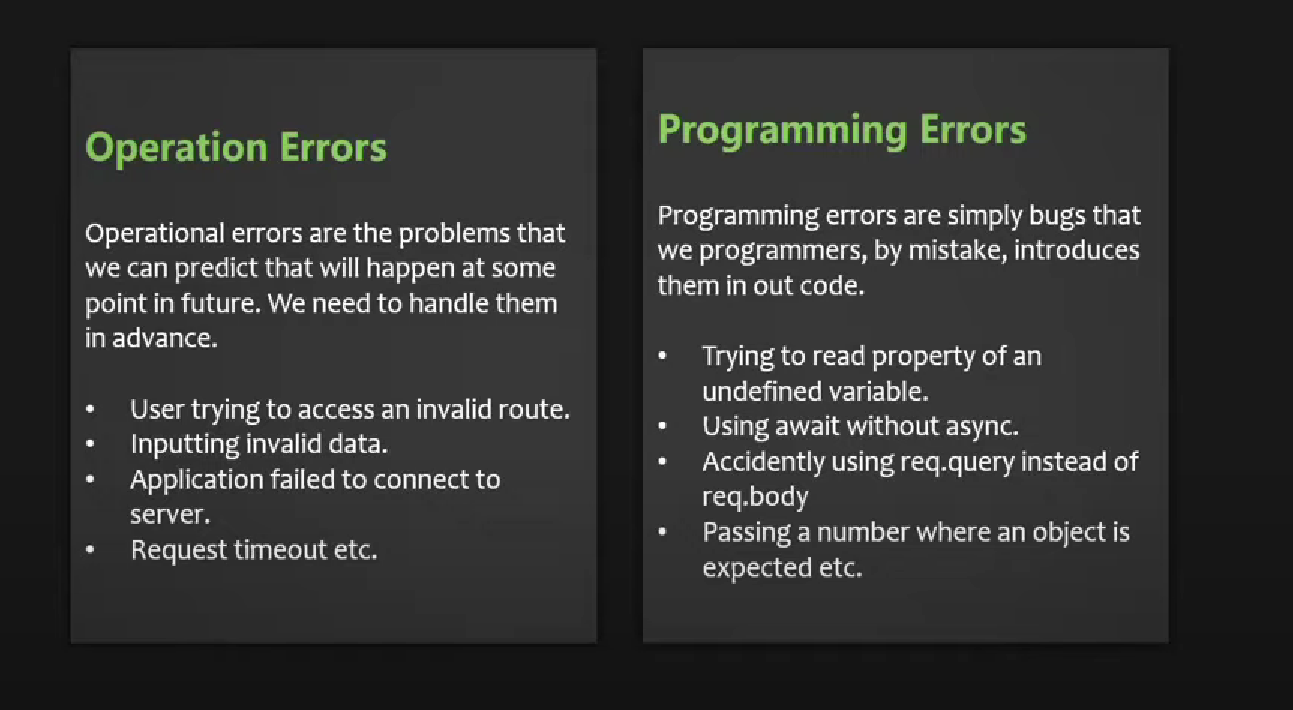
}

Here, in try block we write the code which is on the trial for the error or may cause any error.

Then catch(error) catches the error and we can take actions.

There are two types of errors:

1. Operation Error
2. Programming Error



Global error handler:

// Global error handler

app.use((error , req , res , next)=>{

    error.message = error.message || "Internal Server Error";

    error.statusCode = error.statusCode || 500;

    res.send({success : false , statusCode : error.statusCode , message : error.message});

})

Here's a simple explanation of what it does:

1. **Middleware Registration**: It registers a middleware function using **app.use()**. This middleware will be executed for every incoming request.
2. **Error Handling**: When an error occurs during the processing of a request, it captures that error. The error is typically generated by previous middleware or route handlers when something goes wrong.
3. **Default Error Message and Status Code**: If the error doesn't have a specific message (**error.message**) or status code (**error.statusCode**) set, this middleware assigns default values to them. The default message is "Internal Server Error," and the default status code is 500 (indicating a server error).
4. **Response**: It then sends a JSON response back to the client with information about the error:
   * **success**: This is set to **false** to indicate that the request was not successful.
   * **statusCode**: This is the HTTP status code associated with the error (either the one from the error itself or the default 500).
   * **message**: This is the error message (either the one from the error itself or the default "Internal Server Error").

In summary, this global error handler middleware ensures that any unhandled errors in your Express.js application are captured, and a consistent error response is sent to the client with a status code and message, making it easier to handle errors on the client side.

Custom Error Handler:

module.exports = class ErrorHandler extends Error{

    constructor(message , statusCode){

        super(message);

        this.statusCode = statusCode;

        Error.captureStackTrace(this , this.constructor);

    }

}

The provided code defines a custom error class in JavaScript. Here's a simple explanation of what it does:

1. **Class Definition**: It defines a new class called **ErrorHandler**. This class extends the built-in **Error** class in JavaScript, allowing you to create custom error objects.
2. **Constructor**: The **constructor** function is called when a new **ErrorHandler** object is created. It takes two parameters:
   * **message**: This is a string that represents the error message.
   * **statusCode**: This is a number that represents the HTTP status code associated with the error.
3. **Super Constructor**: Inside the **constructor**, it calls **super(message)**, which invokes the constructor of the parent **Error** class and sets the error message.
4. **Custom Property**: It sets a custom property called **statusCode** on the error object. This allows you to associate an HTTP status code with the error.
5. **Capture Stack Trace**: It calls **Error.captureStackTrace(this, this.constructor)**. This line captures the call stack trace at the point where the error object is created. This is useful for debugging because it provides information about where the error occurred in the code.

In summary, this code defines a custom error class (**ErrorHandler**) that you can use to create error objects with both a message and an associated HTTP status code. This can be helpful when handling errors in an application, as it allows you to provide more detailed information about the nature of the error, making it easier to diagnose and handle errors in your code.