Want to DEBUG AND TEST MERN STACK APPLICATIONS? Checkout some tips and tricks

INTRODUCTION

Debugging and Testing, both play a huge role in developing Software Applications. They are essential in assuring the effectiveness, dependability, and caliber of software programs. Several significant factors emphasize on the significance of Testing and Debugging-

Detects and Resolves flaws: Testing and debugging aid in finding and fixing software application flaws, mistakes, and problems

Saves Time: The time that software engineers save by performing early debugging is due to the fact that sophisticated codes are not needed. It helps software engineers conserve both their time and their energy**.**

Simple Interpretations: By providing additional details about systems of data, it facilitates simple interpretations.

Now let's explore what MERN Stack is all about.....

MERN STACK

For creating full-stack online applications, developers frequently employ the MERN Stack, a JavaScript stack. MongoDB, Express.js, React, and Node.js are all abbreviated as MERN. The MERN stack's individual components each have a distinct function and help create dynamic, effective web applications.

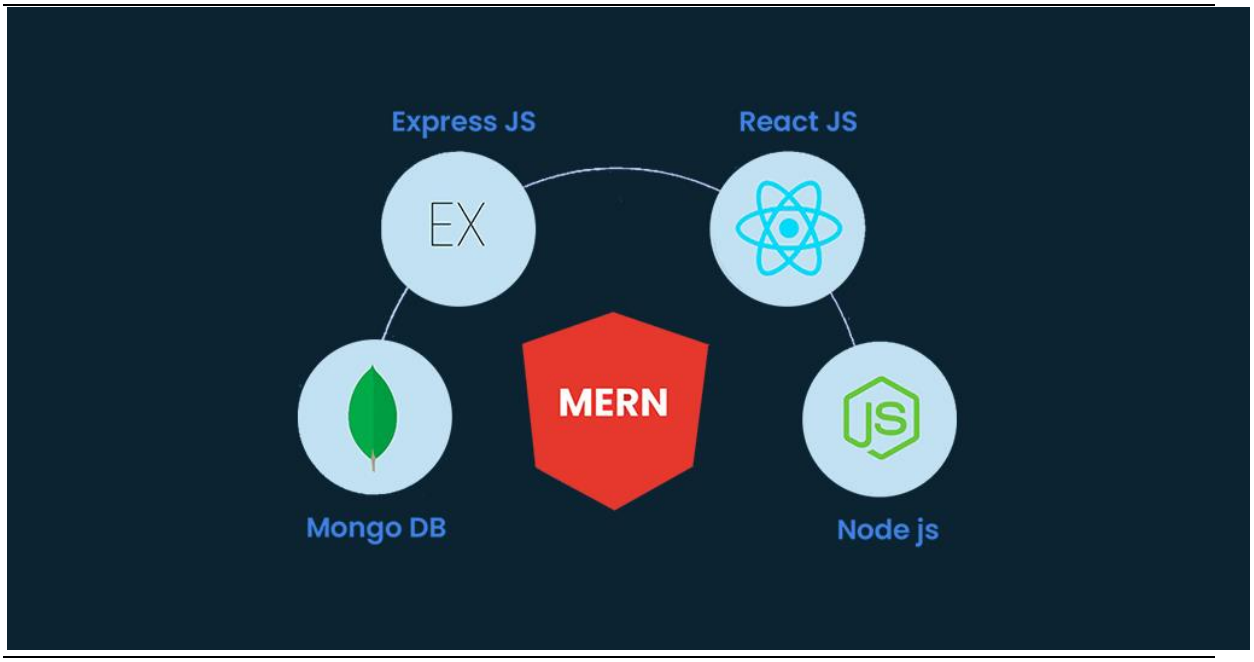
MongoDB — A Document Database

Express.js — Server-Side Web Framework

React.js — Client-side JavaScript Framework

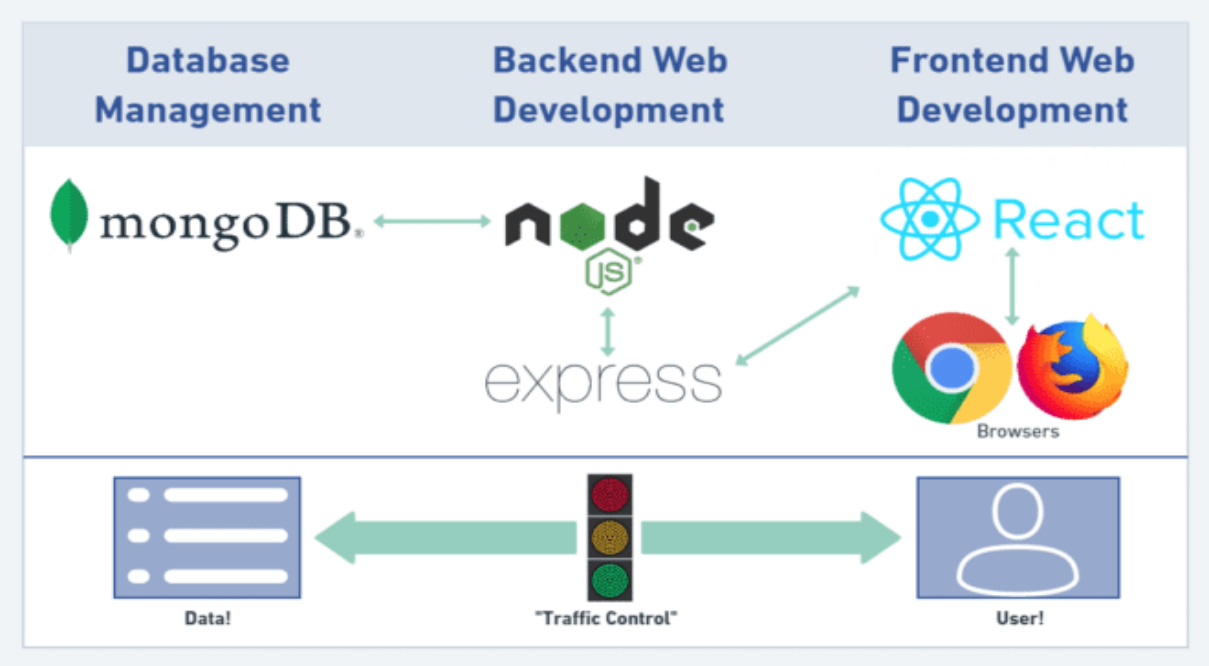
Node.js — Premier JavaScript Server Platform

These 4 potent technologies each offer a complete framework for developers to work within. Also included with the MERN stack are pre-built testing tools. JSON and JavaScript proficiency is a need for developers. Furthermore, they may rely on a sizable community. In addition to its open-source frameworks, if necessary.



HOW MERN STACK WORKS

The Stack is a 3-Tier Architecture – Front-End, Back-End and Database utilizing JavaScript and JSON.



React.js (Front-End)

A well-liked JavaScript library for creating user interfaces is React. It is applied to the development of interactive and dynamic user interfaces for online applications. With React's component-based architecture, developers can create reusable user interface (UI) components that may be combined to create complex user interfaces. React creates quick and responsive apps by rapidly updating and rendering UI components using a virtual DOM (Document Object Model).

Uses

1) High Performance

2) Developing Android/ IOS Applications

3) Error Handling

Express.js and Node.js (Server Tier)

The Node.js web application framework Express.js is simple and adaptable. It offers a collection of strong features and middleware that makes the creation of server-side applications easier. Express.js enables programmers to build routes, manage the application's middleware and routing layers effectively, handle HTTP requests and answers, and implement middleware functions.

A server-side JavaScript runtime based on Chrome's V8 JavaScript engine is called Node.js. It enables the creation of server-side apps by enabling developers to execute JavaScript code outside of a browser. Node.js has an event-driven, non-blocking architecture that makes it very scalable and effective at managing multiple concurrent requests. It is frequently employed in MERN applications to develop APIs, manage data operations, and carry out server-side logic.

Uses

1) Efficient, fast & scalable

2) Single-threaded

3) Open-source JavaScript Runtime Environment

MongoDB (Database Tier)

Data storage may be made flexible and scalable with the help of MongoDB, a NoSQL document-oriented database. It makes working with JavaScript simple by storing data in a JSON-like format called BSON (Binary JSON). MongoDB is suitable for applications with changing or evolving data structures since it enables developers to store and retrieve data in a schema-less way.

Uses-

1) **Data stored in the form of JSON** –

-> Objects, Object Members, Arrays, Values, and Strings

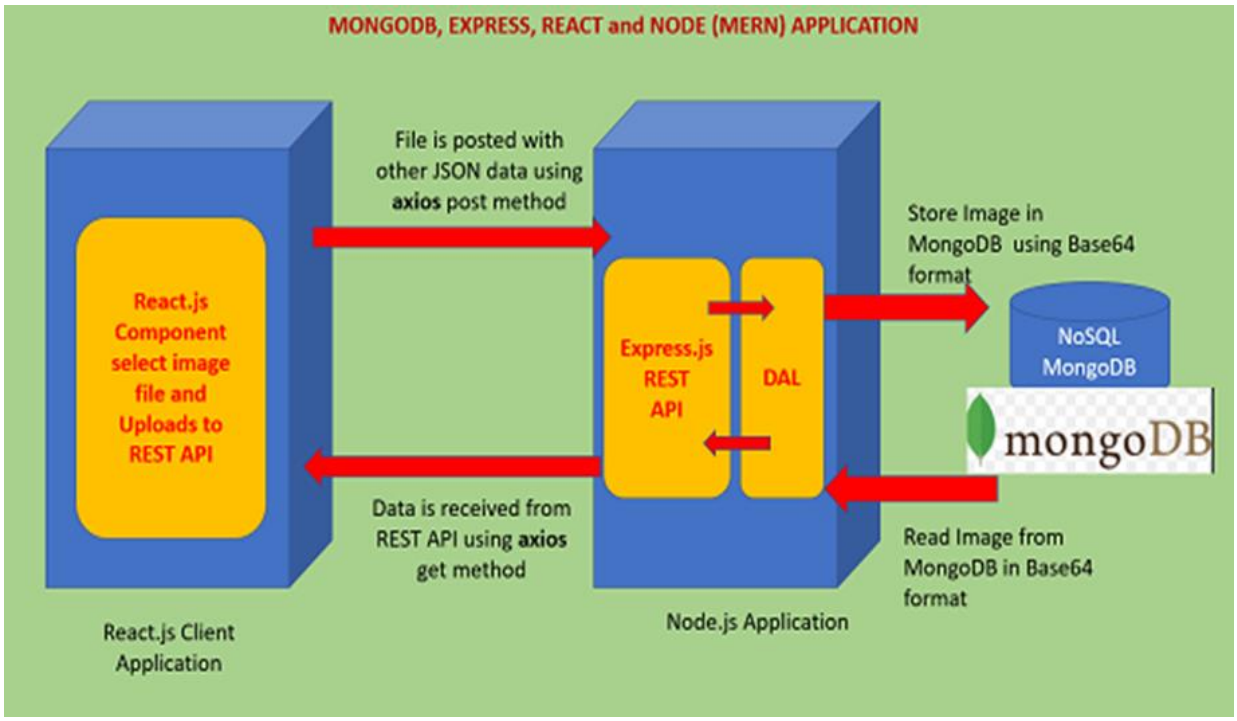
-> JSON syntax is very easy to use.

-> JSON has a wide range of browser compatibility.

-> **Sharing Data**: Data of any size and type (video, audio) can be shared easily.

2)**Flexible Document Model** – MongoDB supports document-model (tables, schemas, columns & SQL) which is faster and easier.

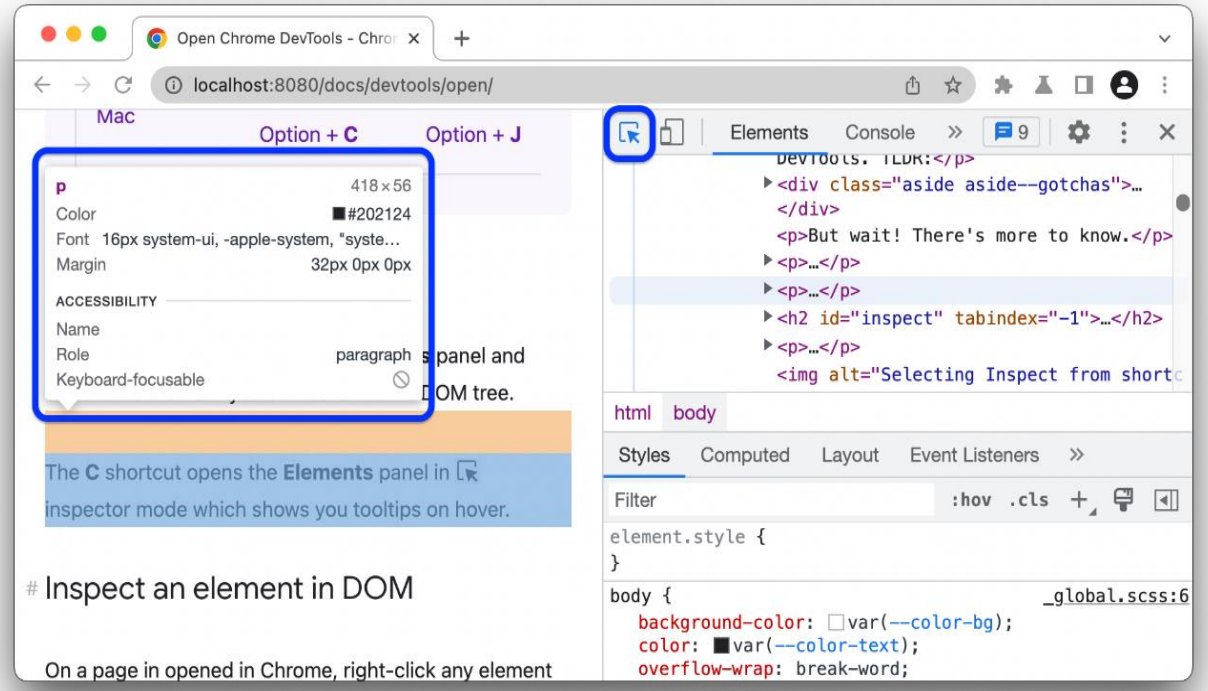
3) **Use of JavaScript** – MongoDB uses JavaScript which is the biggest advantage.



USEFUL TIPS AND TRICKS

1) Google Chrome Developer Tools-

The extensive range of debugging tools provided by the Chrome Developer Tools can be very helpful in identifying and resolving problems with your MERN application. Use the console to look for mistakes, record pertinent information, and examine network requests. Breakpoints, stepping through code execution, and inspecting variables are further options.



2) Checking Logs

Finding the problems in your code can also be done by logging. Some crucial tools for recording data in your MERN application include **Morgan** and **Winston**.

The example below shows how we utilised Winston to log in to eliminate the bug that caused user sessions to expire earlier than planned.

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

function handleUserSession(req, res) {
  const userId = req.session.userId;
  const sessionExpiresAt = req.session.expiresAt;
  const currentTime = new Date().getTime();
  if (currentTime > sessionExpiresAt) {
    // Session has expired
    winston.error(`User session expired for userId=${userId}`);
    req.session.destroy();
    res.sendStatus(401);
  } else {
    // Session is still valid
    res.sendStatus(200);
  }
}
```

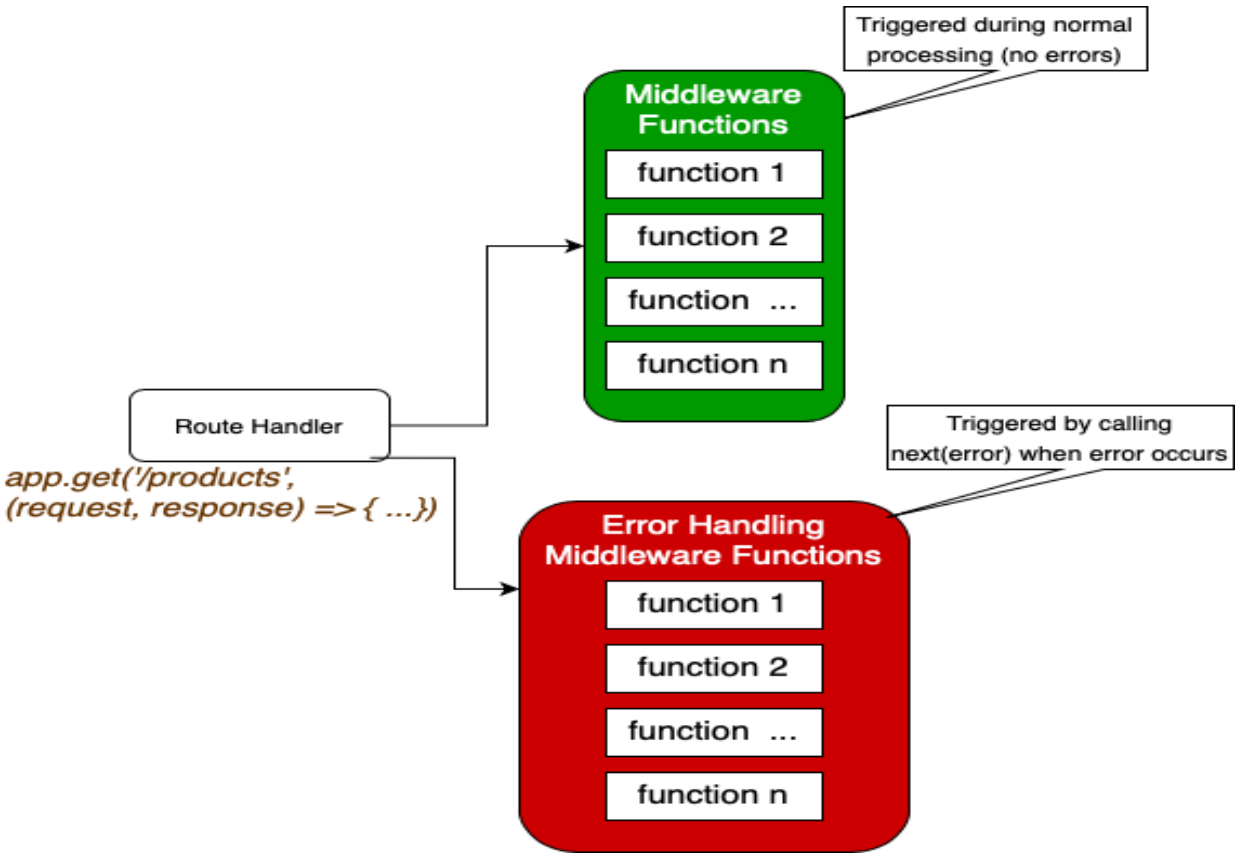
3) **Error handling middleware**

Express.JS is a framework that can be trusted to be used with error-handling middleware in MERN applications. It logs the error, logs the exception, and then returns the error messages to the user**.**

The express middleware can be used in the following scenario where the API returns an error without providing a description:

```
function handleError(err, req, res, next) {
  // Log the error
  winston.error(`Error: ${err.message}`);
  // Return an error message to the client
  res.status(500).json({ error: 'Internal server error' });
}

app.use(handleError);
```



4) **API and UI Testing**

Tools like **Postman** and **Insomnia** can be used to check whether your API endpoints are functioning properly for the hassle-free transfer of data.

If, for instance, your API endpoint retrieves data from a database, **Postman** can be used to verify that the data is being returned appropriately.

```
GET http://localhost:3000/users
```

To identify the issues persisting in the UI components, testing can be performed with the help of tools like **Jest**, **Enzyme** and **Cypress**.

In the following example, we used Enzymen UI testing to test the component if it is being rendered properly using the keyword '**expect**'.

```
import React from 'react';
import {shallow } from 'enzyme';
import UserList from './UserList';

test('renders user list correctly', () => {
  const users = [{name: 'John', age: 30}, {name: 'Mary', age: 25}];
  const wrapper = shallow(<UserList users={users} />);
  expect(wrapper.find('li')). toHaveLength(2);
  expect(wrapper.find('li').at(0).text()).toEqual('John (30 years old)');
  expect(wrapper.find('li').at(1).text()).toEqual('Mary (25 years old)');
});
```

CONCLUSION

Two of the crucial processes taken to guarantee the dependability and functionality of the MERN applications built are Debugging and Testing. By locating the problems and effectively fixing them, we can create a functioning, error-free programme. Debuggers must invest a lot of time and energy into learning the craft so they can apply what they learn in the future.