# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**JNANA SANGAMA, BELGAVI-590018, KARNATAKA**

# A J INSTITUTE OF ENGINEERING & TECHNOLOGY

**(A unit of Laxmi Memorial Education Trust. (R))**
**NH - 66, Kottara Chowki, Kodical Cross, Mangalore- 575 006.**

## DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

# MASTER MANUAL

## Course: Full Stack Development

## (SubjectCode:BISE601)

## VI-SEMESTER


Prepared by:
**Dr. Lokesh M R**
**Professor**
Department of Information Science & Engineering, AJIET, Mangalore


**ACADEMIC YEAR: 2024-25**

# VISVESVARAYATECHNOLOGICALUNIVERSITY,BELAGAVI

**B.E. in Information Science and Engineering**
**Scheme of Teaching and Examinations 2022**

**Outcome-Based Education (OBE) and Choice Based Credit System (CBCS)**

**(Effective from the academic year 2022 - 23)**

## Course: Full Stack Development

## Course Code: BIS601

# TABLE OF CONTENTS

# VISION OF THE INSTITUTE

"To produce top-quality engineers who are groomed for attaining excellence in their profession and competitive enough to help in the growth of nation and global society."

# MISSION OF THE INSTITUTE

**M1:** To offer affordable high-quality graduate program in engineering with value education and make the students socially responsible.

**M2:** To support and enhance the institutional environment to attain research excellence in both faculty and students and to inspire them to push the boundaries of knowledge base.

**M3:** To identify the common areas of interest amongst the individuals for the effective industry- institute partnership in a sustainable way by systematically working together.

**M4:** To promote the entrepreneurial attitude and inculcate innovative ideas among the engineering professionals.

# VISION OF THE DEPARTMENT

"To be a center of excellence in Information Science & Engineering education, research and training to meet the growing needs of the industry and society."

# MISSION OF THE DEPARTMENT

**M1**: To impart theoretical and practical knowledge through the concepts and technologies in Information Science and Engineering.

**M2**: To foster research, collaboration and higher education with premier institutions and industries.

**M3**: Promote innovation and entrepreneurship to fulfill the needs of the society and industry.

# PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

After 4 years of graduation, graduates will be able to

**PEO1:** Analyse, design and implement solutions to the real-world problems in the field of Information Science and Engineering with multidisciplinary setup

**PEO2:** Keep abreast with the technology, innovation and pursue higher education with high standards of social and professional ethics

**PEO3:** Develop professional and entrepreneurship skills to work effectively as an individual and in a team to meet the ever-changing goals of the organization

# PROGRAM OUTCOMES (POs)

**PO1: Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2: Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3: Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4: Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

**PO6: The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7: Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and Team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11: Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-Long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# PROGRAM SPECIFIC OUTCOMES (PSOs)

At the end of the program, graduates will be able to

**PSO1:** Design, implement and maintain the information systems that fulfill the current needs of the industry.

**PSO2:** Apply computational theory, storage, and networking concepts to address the societal problems.

# GENERAL LAB GUIDELINES

## Do's

1. Maintain discipline in the Laboratory.
2. Before entering the Laboratory, keep the footwear on the shoe rack.
3. Proper dress code has to be maintained while entering the Laboratory.
4. Students should carry a lab observation book, student manual and record book completed in all aspects.
5. Read and understand the logic of the program thoroughly before coming to the laboratory.
6. Enter the login book before switching on the computer.
7. Enter your batch member names and other details in the slips for hardware kits.
8. Students should be at their concerned places; unnecessary movement is restricted.
9. Students should maintain the same computer until the end of the semester.
10. Report any problems in computers/hardware kits to the faculty member in-charge/laboratory technician immediately.
11. The practical result should be noted down into their observation and the result must be shown to the faculty member in-charge for verification.
12. After completing the experiments, students should switch off the computers, enter logout time, return the hardware kits and keep the chairs properly.

## Don'ts

1. Do not come late to the Laboratory.
2. Do not enter the laboratory without an ID card, lab dress code, observation book and record.
3. Do not leave the laboratory without the permission of the faculty in-charge.
4. Never eat, drink while working in the laboratory.
5. Do not handle any equipment before reading the instructions/instruction manuals.
6. Do not exchange the computers with others and hardware kits also.
7. Do not misbehave in the laboratory.
8. Do not alter computer settings/software settings.
9. External Disk/drives should not be connected to computers without permission, doing so will attract fines.
10. Do not remove anything from the kits/experimental set up without permission. Doing so will attract fines.
11. Do not mishandle the equipment / Computers.
12. Do not leave the laboratory without verification of hardware kits by the lab instructor.
13. Usage of Mobile phones, tablets and other portable devices are not allowed in restricted places.

# INSTRUCTIONS TO STUDENTS

- Students must bring Observation book, record and manual along with pen, pencil, and eraser etc., no borrowing from others.
- Students must handle the trainer kit and other components carefully, as they are expensive.
- Before switch on the trainer kit, must show the connections to one of the faculties or instructors.
- After the completion of the experiment should return the components to the respective lab instructors.
- Before leaving the lab, should check whether they have switch off the power supplies and keep their chairs properly.
- Be regular to the Lab Do not come late to the Lab
- Do not throw connecting wires on the Floor
- Wear your College ID card Do not operate the IC trainer kits without permission
- Avoid unnecessary talking while doing the experiment
- Avoid loose connection and short circuits
- Take the signature of the lab in charge before taking the components
- Do not interchange the ICs while doing the experiment
- Handle the trainer kit properly
- Do not panic if you do not get the output
- Keep your work area clean after completing the experiment.
- After completion of the experiment switch off the power and return the components

Arrange your chairs and tables before leaving.

# RULES FOR MAINTAINING LABORATORY RECORD

- Put your name, USN and subject on the outside front cover of the record. Put that same information on the first page inside.
- Update Table of Contents every time you start each new experiment or topic
- Always use pen and write neatly and clearly
- Start each new topic (experiment, notes, calculation, etc.) on a right-side (odd numbered) page
- Obvious care should be taken to make it readable, even if you have bad handwriting
- Date to be written every page on the top right side corner
- On each right-side page
  - Title of experiment
  - Aim/Objectives
  - Components Required
  - Theory
  - Procedure described clearly in steps
  - Result
- On each left side page
  - Pin diagrams
  - Circuit diagram
  - Tables
  - Graphs
- Use labels and captions for figures and tables
- Attach printouts and plots of data as needed. Stick printouts (A4 Size) on the right side of the lab record

Strictly observe the instructions given by the Teacher/ Lab Instructor

# SYLLABUS

| | | | |
|---|---|---|---|
| Course Code | BISE601 | Semester | VI |
| Teaching Hours/ Week (L:T:P:S) | 3:0:2:0 | CIE MARKS | 50 |
| Total Hours of Pedagogy | 40 hours Theory + 8-10 Lab slots | SEE MARKS | 50 |
| Credits | 04 | Total Marks | 100 |

# COURSE DESCRIPTION

The FULL STACK DEVELOPMENT Laboratory course immerses students in a dynamic learning environment focusing on the MERN (MongoDB, Express.js, React.js, Node.js) stack. Through hands-on experiments, students delve into full-stack development, understanding each component of the MERN stack. They gain knowledge and apply it in database management with MongoDB, server-side scripting with Node.js and Express.js, and front-end development with React.js.

# COURSE OBJECTIVES

This course will enable the students to:

1. To understand the essential javascript concepts for web development
2. To style Web applications using bootstrap
3. To utilize React JS to build front end User Interface.
4. To understand the usage of API's to create web applications using Express JS.
5. To store and model data in a no sql database.

## Suggested Learning Resources:

1. Jon Duckett, "JavaScript & jQuery: Interactive Front-End Web Development", Wiley, 2014.
2. Vasan Subramanian, Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Apress, 2019.

# COURSE OUTCOMES

After completion of the course, the students will be able to:

1. Apply Javascript to build dynamic and interactive Web projects.
2. Implement user interface components for JavaScript-based Web using React.JS.
3. Apply Express/Node to build web applications on the server side.
4. Develop data model in an open source nosql database. Demonstrate modularization and packing of the front-end modules

| Course Articulation Matrix | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Course Outcomes (COs) | Program Outcomes (POs) | | | | | | | | | | | | Program Specific Outcomes (PSOs) | |
| CO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 |
| CO1 | 2 | 3 | | | 2 | - | | - | - | - | - | 0 | 2 | - |
| CO2 | 3 | | 2 | | | - | | - | - | - | - | 0 | 3 | |
| CO3 | | | 2 | | 2 | - | | - | 2 | - | - | 0 | 2 | |
| CO4 | | 1 | | 2 | 2 | - | | - | - | - | - | | 3 | |
| CO5 | 1 | | | | 3 | - | | - | - | - | - | 3 | 2 | |
| Avg | 2 | 2 | 2 | 2 | 2.25 | - | - | - | 2 | - | - | 3 | 2.4 | - |

## Justification of CO-PO mapping

The CO-PO mapping establishes a clear alignment between the Course Objectives (COs) and Program Objectives (POs), emphasizing the comprehensive development of engineering competencies.

- CO1 equips students with foundational JavaScript skills, contributing to PO1 (Engineering Knowledge), PO2 (Problem Analysis), and PO5 (Modern Tool Usage) by enabling problem-solving through programming techniques.
- CO2 focuses on styling web applications using Bootstrap, aligning with PO1 and PO3 (Design/Development of Solutions) as students creatively design aesthetically pleasing interfaces.
- CO3 builds proficiency in React JS for front-end development, fostering collaboration (PO9: Individual and Team Work), while ensuring modern design practices (PO5).
- CO4 introduces API integration with Express JS, encouraging research-driven approaches to problem-solving (PO4: Conduct Investigations) and robust analytical skills (PO2). Finally,
- CO5 empowers students to work with NoSQL databases, resonating with PO1, PO5, and PO12 (Life-long Learning) as they adapt to evolving technological advancements. Together, these mappings enable a holistic learning experience that prepares students for tackling complex engineering challenges.

# COURSE PRE-REQUISITES

- Proficiency in programming is crucial, preferably in web page development.

# ASSESSMENT DETAILS (BOTH CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End

## Split-up of Marks used Practical Sessions(25M)

| **Split up Of Marks** | **Practical Sessions- Continuation Evaluation (CE) Methodology / Process Steps per Experiment** | **Marks( 25M)** |
|---|---|---|
| #R1 | Observation, Write up of Procedure / Algorithm/ Program and Execution of experiment | 7 |
| #R2 | Record writing | 5 |
| #R3 | Viva – Voce (Questions & Answers on relevant Experiment /Topic) | 3 |
| | **Total Marks **** | **15** |
| | **Practical Sessions-Internal Assessment (IA)** | |
| #R1 | Write-up of Procedure/Program/Algorithm | 10 |
| #R2 | Conduction/Execution | 30 |
| #R3 | Viva-Voce | 10 |
| | **Total Marks** | **50/5= 10** |

****Set as per the syllabus rubrics**
**Total marks for CIE=15+10=25MARKS**

# SEMESTER END EXAMINATION(SEE):

- SEE marks for the practical course is 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the internal/external examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

- General rubrics suggested for SEE are mentioned here, writeup-15marks, Conduction procedure and result is 70marks, Viva-voce is for 15marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course

# RUBRICS FOR PRACTICAL SESSIONS

**Course: Full Stack Development Course Code: BIS601**

| **Practical Sessions- Continuous Evaluation (CE)-25Marks** | | | | |
|---|---|---|---|---|
| **Evaluation Parameter** | **Level of Achievement** | | | |
| #R1: Observation/ Conduction (7 Marks) | Excellent(7-5) | Good (4-3) | Average (3-2) | Poor (1-0) |
| | Observation neatly written. Handwriting is clear. Programs written with no mistakes. Programs executed with no errors | Observation neatly written. Handwriting is clear. Programs written with few mistake. Programs executed with less errors | Observation is written in unclear manner.Handwriting is not very clear. Programs written with fewer mistakes. Programs executed with few errors | Observation is written in unclear manner. Handwriting is not clear, Programs written with lot of mistakes. Programs executed with a large number of errors. |
| #R2: Record (5 Marks) | Excellent (5-4) | Good (4-3) | Average (3-2) | Poor (1-0) |
| | Record is neatly written, handwriting is clear. Mistakes are covered and corrected properly and neatly. Record submitted on time | Record is neatly written, handwriting is clear. Most mistakes are covered and corrected properly and neatly. Record submitted with a delay of 1 - 3 days | Record is written in an unclear manner. Handwriting is not very clear. Mistakes are sometimes corrected properly. Record submitted with a delay of 4 to 5 days | Record is written in an unclear manner. Handwriting is not very clear. Mistakes are not corrected. Record submitted after a delay of 1 week |
| #R3: Viva (3 Marks) | Excellent (3-2) | Good (2-1) | Average (1-0) | Poor (1-0) |
| | Answered all questions with elaboration has excellent | Answered most of the questions Failed to elaborate some of the concepts | Answered a few questions. Subject knowledge is not adequate | Not able to answer any of the questions. Subject |

| | | | | |
|---|---|---|---|---|
| | understanding of the topic. | | | knowledge not adequate |
| **Practical Sessions- Internal Assessment (IA)(25-->10Marks)** | | | | |
| #R1: Write-Up (10 Marks) | Excellent (10-8) | Good (8-6) | Average (5-2) | Poor (2-0) |
| | Program neatly written. Handwriting is clear. Programs written with no mistakes. | Program neatly written. Handwriting is clear. Programs written with very few mistakes. | Program is written in unclear manner. Handwriting is not very clear. Programs written with fewer mistakes. | Program is written in unclear manner. Handwriting is not clear, Programs written with lot of mistakes. |
| #R2: Conduction/ Execution (30Marks) | Excellent (30-24) | Good (24-16) | Average (15-10) | Poor (10-0) |
| | Execution of the program done as per the procedure. Programs had less than 10 errors. The errors were debugged without any help. The Result was tabulated for all the cases. | Execution of the program done as per the procedure. Programs had less than 20 errors. The errors were debugged with a little help. The Result was tabulated for almost all the cases | Execution of the program done as per the procedure. Programs had more than 20 errors. The errors were debugged with the help of instructor. The Result was tabulated for few of the cases | Execution of the program was not done as per the procedure. Programs was full of syntax and logical error. The errors were resolved by the instructor. The Result was tabulated only for 1 or 2 Cases |
| #R3:Viva (10 Marks) | Excellent (10-8) | Good (8-6) | Average (6-4) | Poor (3-0) |
| | Answered all questions with elaboration has excellent understanding of the topic. | Answered most of the questions Failed to elaborate some of the concepts | Answered a few questions. Subject knowledge is not adequate | Not able to answer any of the questions. Subject knowledge not adequate |

**Course Instructor** **Domain Coordinator** **Head of Department**

## LIST OF MAJOR EQUIPMENT

## Name of the Laboratory: COMPUTING LABORATORY-11

| Sl. No. | Name of the Equipment | Specialization | Quantity |
|---|---|---|---|
| 1. | Desktop | Intel(R) Core (TM) i5-12400 upto 4.4Ghz Processor<br>Intel H610 Chipest Motherboard<br>16.0 GB DDR4 3200Mhz Memory 512GB NVMe SSD Acer USB Keyboard & Optical Mouse Windows 11 Professional-factory preloaded<br>RAM, 64-bit operating system | 70 |
| 2. | UPS | 60KVA | 1 |
| 3. | Switches | 24 Port Gigabytes | 2 |
| 4. | Internet | 700mbps | 1 |
| 5. | Projector | Epson EB-982W with 10*8 Mounting Projector Screen – Epson Dongle | 1 |

**Room Number** : **A-423**

**Total Area of the laboratory** : **1575 Sq Ft**

**Total Amount Spent** : Rs. 51,11,403 /-

**Name of the HOD** : **Dr. Lokesh M R**

**Name of the lab in charge** : Ms.Apoorva

**Name of the lab instructor** : Ms.Sushmitha

# CONTENT

# Introduction

## Full Stack Web Development

**what is the full Stack Web Development?**

- Full stack web development refers to the building of web applications that encompass both frontend and backend components.
- A full-stack developer is proficient in frontend technologies, such as HTML, CSS and JavaScript, as well as backend technologies, such as server-side programming languages, databases, and server management.

## Technologies

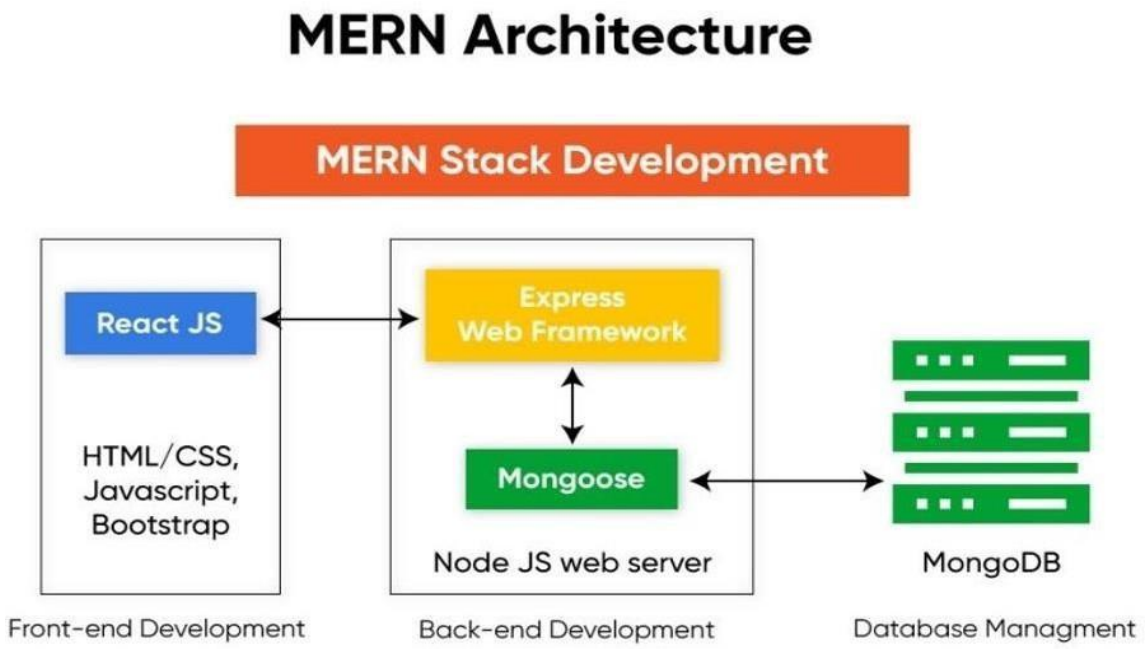
- **MERN (MongoDB, Express.js, React and Node.js)**
- **MEAN ((MongoDB, Express.js, Angular and Node.js)**
- **MEVN (MongoDB, Express.js, Vue and Node.js)**
- **LAMP (Linux, Apache, MySQL, PHP)**

**MongoDB: A cross-platform document-oriented database program**

**Express.js: A web application framework for Node.js**

**React: JavaScript library for building user interfaces**

**Node.js: An open source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser**



# Basics of Full-Stack using VS Code

**Requirements:**

1. **Install VS Code (Editor)**

   **Steps:**

   - Go to **chrome** and search **VS code download**


   - Select the file based on your **operating system** and download that
   - Once you download the file **install** it.
   - Ensure the **"Add to PATH"** option is checked.

2. **Install NodeJS**

   **Node.js** is an open-source, cross-platform JavaScript runtime that allows you to run JavaScript outside of a web browser. It is commonly used for **backend development**, enabling the creation of fast, scalable web applications.

   **Steps:**

   - Open **Google Chrome**.
   - In the search bar, type **"Download Node.js"** and press **Enter**.
   - Click on the **official Node.js website** from the search results.
   - Click the version that matches your **Operating System (Windows/macOS/Linux)**.
   - Once you click the download button, the installer file (**.exe for Windows, .pkg for macOS**) will start downloading.
   - After the download finishes install it.
   - Ensure the **"Add to PATH"** option is checked.

   **Verify Installation**

- Open **Command Prompt (Windows)** or **Terminal (macOS/Linux)**.
- Enter the following commands and check

**node -v**

**npm -v**

**What is NPM?**

**NPM (Node Package Manager)** is the default package manager for **Node.js**. It helps developers install, manage, and share JavaScript libraries and dependencies for their projects.

**Why is NPM Used?**

1. **Install Packages:** Download and use open-source JavaScript libraries like React, Express, and Lodash.
2. **Manage Dependencies:** Keeps track of all installed packages in a project.
3. **Run Scripts:** Automate tasks like starting a server or running tests.
4. **Version Control:** Ensures compatibility with different versions of packages.
5. **Publish Packages:** Developers can create and share their own NPM packages.

**What is Bun?**

**Bun** is a fast, all-in-one JavaScript runtime, package manager, and task runner designed as an alternative to **Node.js** and **NPM/Yarn**. It is built using **Zig**, a low-level systems programming language, making it highly optimized for performance.

| Feature | Bun □ | Node.js |
|---|---|---|
| **Startup Speed** | ⚡ Extremely fast (due to native Zig optimizations) | 🚀 Fast, but slower than Bun |
| **Package Manager** | ✅ Built-in (replaces npm and yarn) | ❌ Uses external NPM/Yarn |
| **Performance** | 🚀 3x faster than Node.js in many cases | 🔄 Slower compared to Bun |
| **Native APIs** | ✅ Supports Web APIs like fetch() & WebSockets | ⚠ Some require polyfills |
| **Low Memory Usage** | ✅ More efficient (lower RAM consumption) | ❌ Uses more memory |
| **Bundling & Transpiling** | ✅ Built-in bundler (replaces Webpack & Babel) | ❌ Requires third-party tools |
| **Ecosystem** | 🔄 Still growing | 🏆 Mature with many libraries |

**How to Install Bun Using NPM in PowerShell**

If you already have **Node.js and NPM** installed, you can install **Bun** using NPM.

**Step 1: Open PowerShell**

Press Win + X → Click **PowerShell (Admin)** or **Terminal**.

**Step 2: Install Bun via NPM**

Run the following command:

*npm install -g bun*

**Verify the Installation**

Check if Bun is installed by running:

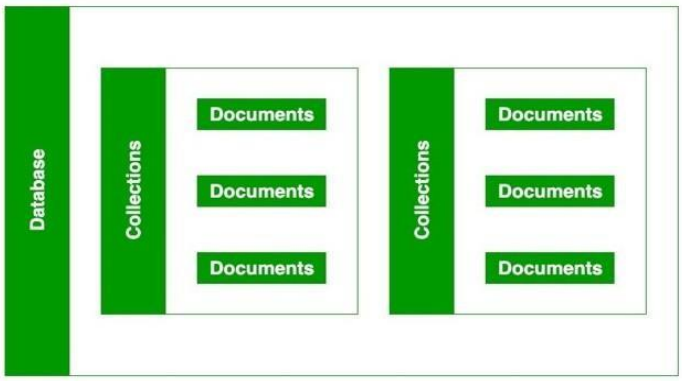
*bun -v*

If installed correctly, it will display the **Bun version**.

mongoDB.

**What is it?**

- MongoDB is a document-oriented NoSQL database system that provides high scalability, flexibility, and performance.
- It is categorized under the NoSQL database



# MongoDB Features

- **Schema-less Database:**

  A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size.

- **Document Oriented:**

  In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair).

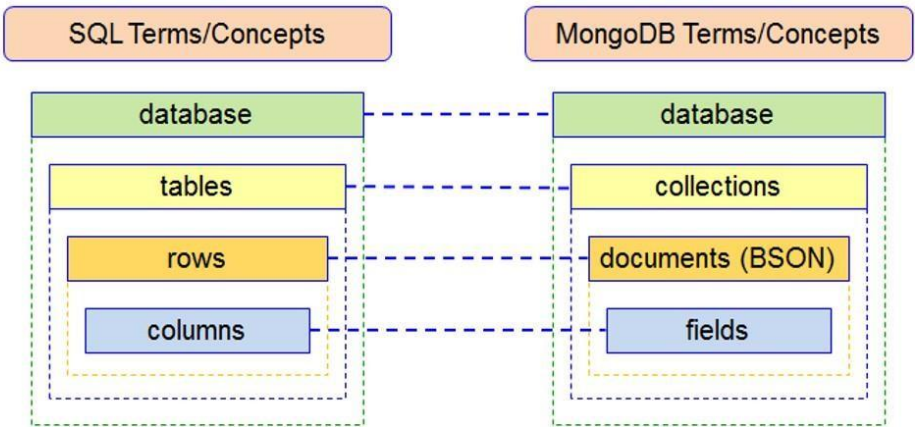
- **Scalability:**

  MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers.

- **Indexing:**

  Every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data.

## Comparison between SQL & MongoDB Terms/Concepts



# Experiment 1

1. **a) Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box**

## PROGRAM:

**Method1: Using Traditional function**

```
console.log("Hello, World");


function CalculateSum(num1, num2){
   let sum = num1+num2;
   console.log("The sum is: ",sum);
}
CalculateSum(5,7);
```

**Method2 : Using fat arrow function**

```
console.log("Hello, World");


const CalculateSum = (num1, num2) => {
   let sum = num1+num2;
   console.log("The sum is: ",sum);
}
CalculateSum(5,7);
```

### To run program use below command:

- Open the vscode integrated terminal through the menu by selecting and type the below command to run the program.

  **node fileName.js**

## OUTPUT:

Hello, World!

The sum is: 12

**1b) Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.**

## PROGRAM:

```
let cities = ["New York", "Los Angeles", "Chicago", "Miami", "Houston"];
console.log("Total number of cities: "+ cities.length);
// push() - To add one or more elements to end
//pop() - To remove elem from end
// unshift() - To add elements at the beginning
//shift() - To remove elements from beginning
cities.push("San Francisco");
console.log("Cities after adding a new one: " + cities);

cities.shift();
console.log("Cities after removing the first one: " + cities);

let cityIndex = cities.indexOf("Miami");
console.log("Index of Miami: " + cityIndex);
```

**OUTPUT:**

Total number of cities: 5

Cities after adding a new one: New York,Los Angeles,Chicago,Miami,Houston,San Francisco

Cities after removing the first one: Los Angeles,Chicago,Miami,Houston,San Francisco

Index of Miami: 2

# Experiment 2

**Read a string from the user, Find its length. Extract the word** "JavaScript" **using** substring() **or** slice()**. Replace one word with another word and log the new string. Write a function** isPalindrome(str) **that checks if a given string is a palindrome (reads the same backward).**

## PROGRAM:

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

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function isPalindrome(str) {
  const reversedStr = str.split('').reverse().join('');
  return str === reversedStr;
}
rl.question('Enter a string: ', (inputString) => {
  const lengthOfString = inputString.length;
  console.log(`Length of the string: ${lengthOfString}`);

  const start = inputString.indexOf("JavaScript");
  if (start !== -1) {
    const extractedWord = inputString.slice(start, start + "JavaScript".length);
    console.log(`Extracted word: ${extractedWord}`);
  } else {
    console.log("Word 'JavaScript' not found in the string.");
  }
  const replacedString = inputString.replace('JavaScript', 'JavaScript lab program');
  console.log(`New string after replacement: ${replacedString}`);
```

```
  const isStringPalindrome = isPalindrome(inputString);
  console.log(`Is the string a palindrome? ${isStringPalindrome}`);


  rl.close();
});
```

## To run program use below command:

- Open the vscode integrated terminal through the menu by selecting and type the below command to run the program.

**node index.js**

## OUTPUT:

********************************output 1********************************

**Enter a string: I am learning JavaScript from vtucircle website**

**Length of the string: 47**

**Extracted word: JavaScript**

**New string after replacement: I am learning JavaScript lab program from vtucircle website**

**Is the string a palindrome? false**

********************************output 2********************************

**Enter a string: madam**

**Length of the string: 5**

**Word 'JavaScript' not found in the string.**

**New string after replacement: madam**

**Is the string a palindrome? True**

# Experiment 3

**Create an object student with properties: name (string), grade (number), subjects (array),** displayInfo() **(method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.**

## PROGRAM:

```
let student = {
    name: "John Doe",
    grade: 85,
    subjects: ["Math", "Science", "English"],

    displayInfo: () => {
        console.log(`Name: ${student.name}`);
        console.log(`Grade: ${student.grade}`);
        console.log(`Subjects: ${student.subjects.join(', ')}`);
    }
};

// Dynamically adding the 'passed' property based on grade
student.passed = student.grade >= 50;

// Calling displayInfo()
student.displayInfo();

// Looping through student object properties
for (let key in student) {
    if (typeof student[key] !== 'function') {
        console.log(`${key}: ${student[key]}`);
    }
}
```

**OUTPUT:**

Name: John Doe

Grade: 85

Subjects: Math, Science, English

name: John Doe

grade: 85

subjects: Math,Science,English

passed: true

# Experiment 4

**Create a button in your HTML with the text** "Click Me"**. Add an event listener to log** "Button clicked!" **to the console when the button is clicked. Select an image and add a** mouseover **event listener to change its** border color**. Add an event listener to the document that logs the key pressed by the user.**

## PROGRAM:

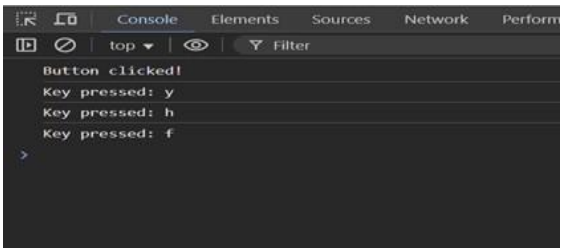
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Listeners Example</title>
    <style>
        img {
            width: 200px;
            height: 140px;
            border: 5px solid black;
            margin-top: 10px;
        }

        button {
            padding: 5px 10px;
            background:  #000;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
        }
    </style>
</head>
<body>
```

```
    <button id="myButton">Click Me</button><br>
    <img id="myImage" src="https://vtucircle.com/wp-content/uploads/2025/01/Full-Stack-Development-BIS601.jpg" alt="Image">
    <script>
        // Button Click Event
        const button = document.getElementById('myButton');
        button.addEventListener('click', () => {
            console.log('Button clicked!');
        });
        // Image Mouseover Event
        const image = document.getElementById('myImage');
        image.addEventListener('mouseover', () => {
            image.style.borderColor = 'red';
        });

        // Keydown Event for Document
        document.addEventListener('keydown', (event) => {
            console.log(`Key pressed: ${event.key}`);
        });
    </script>

</body>
</html>
```

**OUTPUT:**



**KEYBOARD PRESS**



# Experiment 5

**Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.**

### Step 1: Install Node.js and npm

Before you start, ensure you have **Node.js** and **npm** (Node Package Manager) installed on your machine.

**node -v**
**npm –v**

### Step 2: Create a folder

- Create a folder in **any location**.
- Open folder in **vscode**.

### Step 3: Open Integrated vscode Terminal

- Type below command in your terminal to create **react app**.

**npx create-react-app issue-tracker**

- Once the process is complete, you'll see a folder named **issue-tracker** in your directory. Navigate into the folder:

**cd issue-tracker**

### Step 4: Modify the App.js File and App.css file

- Modify the app by editing **src/App.js** copy the below code and paste it, save it.
- Modify the app by editing **src/App.css** copy the below code and paste it, save it.

### Step 5: Start the Development Server

Now, you can start the development server by using below command:
**npm start**

## PROGRAM:

### App.js:

```
import React from 'react';
import './App.css';

const issues = [
  {
    id: 1,
    title: "Bug in login page",
    description: "The login page throws an error when submitting invalid credentials.",
    status: "Open",
  },
  {
    id: 2,
    title: "UI glitch on homepage",
    description: "There is a UI misalignment issue on the homepage for smaller screens.",
    status: "Closed",
  },
  {
    id: 3,
    title: "Missing translation for settings page",
    description: "The settings page is missing translations for the Spanish language.",
    status: "Open",
  },

{
    id: 4,
    title: "Database connection error",
    description: "Intermittent database connection issue during peak hours.",
    status: "Open",
  },
];
const Issue = ({ title, description, status }) => {
  return (
    <div className="issue">
      <h3>{title}</h3>
      <p>{description}</p>
      <span className={`status ${status.toLowerCase()}`}>{status}</span>
    </div>
  );
};
const App = () => {
  return (
    <div className="App">
      <h1>Issue Tracker</h1>
      <div className="issue-list">
        {issues.map((issue) => (
          <Issue
```

```
            key={issue.id}
            title={issue.title}
            description={issue.description}
            status={issue.status}
          />
        ))}
      </div>
    </div>
  );
};

export default App;
```

## App.css:

```
.App {
  font-family: Arial, sans-serif;
  text-align: center;
  margin: 20px;
}
h1 {
  color: #333;
}
.issue-list {
  display: flex;
  flex-direction: column;
  align-items: center;
}


.issue {
  background-color: #f9f9f9;
  border: 1px solid #ccc;
  border-radius: 5px;
  padding: 15px;
  margin: 10px;
  width: 80%;
  max-width: 600px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}
.issue h3 {
  margin: 0 0 10px;
  font-size: 1.5em;
}

.issue p {
  margin: 0 0 10px;
  color: #555;
}
```

```
.status {
  font-weight: bold;
}
.status.open {
  color: #e74c3c;
}
.status.closed {
  color: #2ecc71;
}
```

**Start the development server by using below command:**

**npm start**

This will start a local development server and automatically open your new React application in your default web browser. The server will reload the page automatically whenever you make changes to your code.

**"Fixing the Module not found: Error: Can't resolve 'web-vitals' Error in React"**

The error you're seeing occurs because the web-vitals package, which is used for performance monitoring in a React app, is not installed by default in the project or has been removed. Since web-vitals is an optional package, you can safely resolve this issue by either installing the package or removing the code that imports it.

**Option 1: Install the web-vitals package**

If you want to keep the performance monitoring functionality and resolve the error, simply install the web-vitals package.

1. In the terminal, navigate to your project folder (if not already there):
   **cd issue-tracker**
2. Install **web-vitals** by running the following command:
   **npm install web-vitals**
3. After installation is complete, restart the development server:
   **npm start**

This should resolve the error, and your application should compile correctly.

**Option 2: Remove the Web Vitals Code (If Not Needed)**

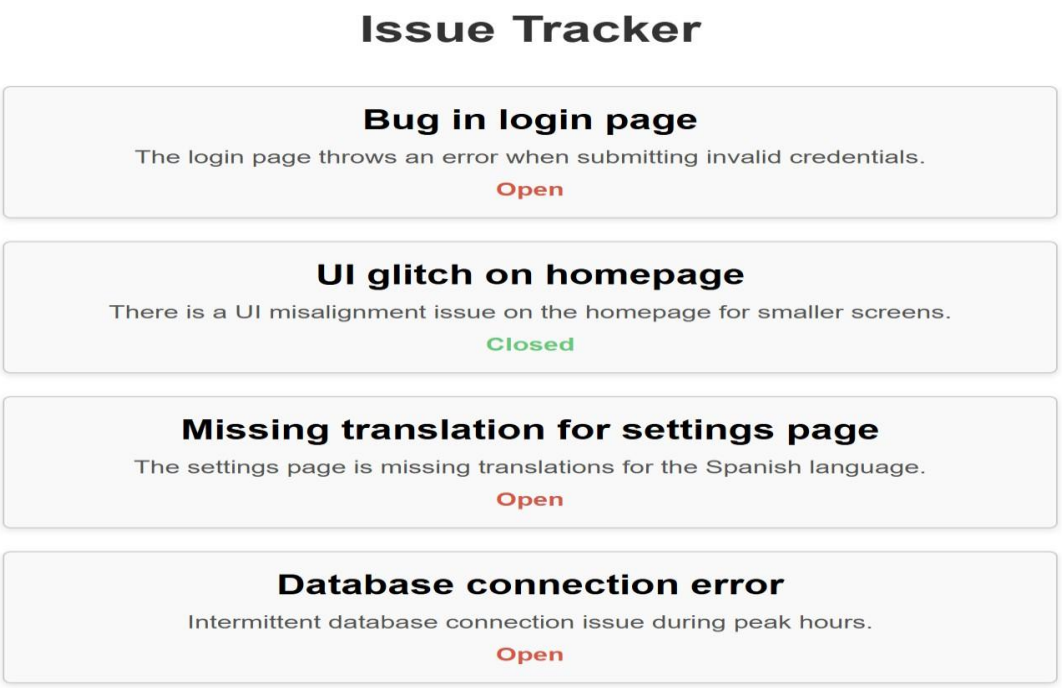
If you don't need performance monitoring and want to get rid of the error, you can safely remove the import and usage of web-vitals from your code.

1. Open **src/reportWebVitals.js** and remove its contents or just comment out the code:

```
// import { reportWebVitals } from './reportWebVitals';

// You can safely remove the call to reportWebVitals or leave it commented out
// reportWebVitals();
```

Save the file, and the application should compile without the error. You can now continue developing your app.

**OUTPUT:**


Issue Tracker
Bug in login page
The login page throws an error when submitting invalid credentials.
Open
UI glitch on homepage
There is a UI misalignment issue on the homepage for smaller screens.
Closed
Missing translation for settings page
The settings page is missing translations for the Spanish language.
Open
Database connection error
Intermittent database connection issue during peak hours.
Open

# Experiment 6

**Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.**

## Step 1: Create Project

- Create a folder and open with **vscode**.
- Open **integrated vscode terminal**.
- After then **create a react app** using below command.

**npx create-react-app counter-app**

- After creating successfully then change the directory using below command.

**cd counter-app**

## Step 2: Create File inside the src folder

- Create two separate file **Counter.js** and **CounterClass.js** inside the **src** folder.
- After then copy the below **code and paste** within the respective file **Counter.js** and **CounterClass.js**
- After then Modify the **App.js** file present in the **src** folder. Copy the below code and paste it.
- After then Modify the **App.css** file present in the **src** folder. Copy the below code and paste it.

## Step 3: Start the Development Server

## PROGRAM:

### Counter.js:

```
import React, { useState, useEffect } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Fetching initial data...');
  }, []);

  const increment = () => setCount(prevCount => prevCount + 1);
```

```
  const decrement = () => setCount(prevCount => prevCount - 1);
  const doubleCount = () => setCount(prevCount => prevCount * 2);
  const resetCount = () => setCount(0);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={doubleCount}>Double</button>
      <button onClick={resetCount}>Reset</button>
    </div>
  );
};

export default Counter;
```

**App.js:**

```
import React from 'react';
import './App.css';
import Counter from './CounterClass';
function App() {
  return (
    <div className="App">
      <h1 className='head'>Welcome to the Counter App</h1>
      <Counter />
    </div>
  );
}
export default App;
```

**App.css:**

```
.App {
  border-radius: 7px;
  background: #ffeaea;
  text-align: center;
}

#root {
  margin: 20px auto;
  width: 40%;
  height: 400px;
}

h1 {
  color: #fff;
  font-size: 25px;
  border-radius: 6px;
  margin: 20px auto;
  background: #000000;
  width: fit-content;
  padding: 10px 60px;
}
.head {
  width: 100%;
  font-size: 25px;
  margin: 0;
  border-radius: 6px;
  color: #fff;
```

```
  background: #000000;
  padding: 10px 0;
}

button {
  margin: 10px;
  padding: 8px 10px;
  font-size: 16px;
  border: none;
  font-weight: 600;
  border-radius: 5px;
  cursor: pointer;
  background: #e02020;
  color: #fff;
}
```

**Now, you can start the development server by using below command:**

**npm start**

**This will start a local development server and automatically open your new React** application in your default web browser. The server will reload the page automatically whenever you make changes to your code.

"Fixing the **Module not found: Error: Can't resolve 'web-vitals' Error in React"**


**The error you're seeing occurs because the web-vitals package, which is used for performance monitoring in a React app, is not installed by default in the project or has been removed. Since web-vitals is an optional package, you can safely resolve this issue by either installing the package or removing the code that imports it.**

**Option 1: Install the web-vitals package**
If you want to keep the performance monitoring functionality and resolve the error, simply install the web-vitals package.

In the terminal, navigate to your project folder (if not already there):

**cd counter-app**

**Install web-vitals by running the following command:**

**npm install web-vitals**

**After installation is complete, restart the development server:**

**npm start**

This should resolve the error, and your application should compile correctly.

**Option 2: Remove the Web Vitals Code (If Not Needed)**
If you don't need performance monitoring and want to get rid of the error, you can safely remove the import and usage of web-vitals from your code.

**Open src/reportWebVitals.js and remove its contents or just comment out the code:**

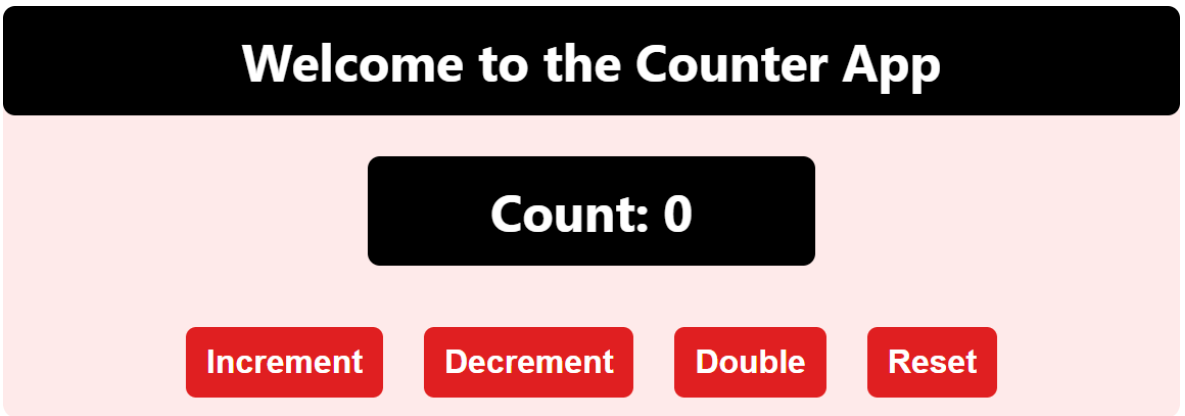
**// import { reportWebVitals } from './reportWebVitals';**

**// You can safely remove the call to reportWebVitals or leave it commented out**

**// reportWebVitals();**

Save the file, and the application should compile without the error. You can now continue developing your app.

## OUTPUT:



# Experiment 7

**Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST: Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.**

## PROGRAM:

**server.js**

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

// Middleware to log requests to the console
app.use((req, res, next) => {
    console.log(`${req.method} request made to: ${req.url}`);
    next();
});
// Middleware to parse JSON payloads
app.use(express.json());

// Root endpoint to return "Hello, Express!"
app.get('/', (req, res) => {
    res.send('Hello, Express!');
});

// Sample in-memory data for products
let products = [
    { id: 1, name: 'Laptop', price: 1000 },
    { id: 2, name: 'Phone', price: 500 }
];

// GET /products: Returns a list of all products
```

```
app.get('/products', (req, res) => {
    res.json(products);
});

// POST /products: Adds a new product
app.post('/products', (req, res) => {
    const { name, price } = req.body;
    if (!name || !price) {
        return res.status(400).json({ message: 'Name and price are required' });
    }
    const newProduct = { id: products.length + 1, name, price };
    products.push(newProduct);
    res.status(201).json(newProduct);
});

// GET /products/:id: Returns details of a specific product
app.get('/products/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) {
        return res.status(404).json({ message: 'Product not found' });
    }
    res.json(product);
});

// PUT /products/:id: Updates an existing product
app.put('/products/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) {
        return res.status(404).json({ message: 'Product not found' });
    }
```

```
    const { name, price } = req.body;
    if (!name || !price) {
        return res.status(400).json({ message: 'Name and price are required' });
    }
    product.name = name;
    product.price = price;
    res.json(product);
});
// DELETE /products/:id: Deletes a product
app.delete('/products/:id', (req, res) => {
    const productIndex = products.findIndex(p => p.id === parseInt(req.params.id));
    if (productIndex === -1) {
        return res.status(404).json({ message: 'Product not found' });
    }
    products.splice(productIndex, 1);
    res.status(204).send();
});

// Start the server on port 3000
const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

**Step 1: Create Project**

- Create a folder and open with **vscode**.
- Open **integrated vscode terminal**.

## Step 2: Install Express:

Execute one by one below command.

**mkdir express-api**

**cd express-api**

**npm init -y**

**npm install express**

## Step 3: Create server.js

- Create **server.js** file inside the **express-api** folder. Right click on the **express-api** folder and create file **server.js**.

After then copy the below code and paste it in that **server.js** file, save it.

## Step 4: Download and Install Postman:

- **Visit the Postman download page**:
  - Go to the Postman download page.
- **Download Postman**:
  - Click on the **"Download"** button for Windows (or your respective operating system). The website will automatically detect your OS.
- **Run the installer**:
  - Once the download is complete, open the **.exe** file to begin the installation process.
  - Follow the on-screen instructions to install Postman.
- **Launch Postman**:
  - After installation, you can launch Postman from your desktop or Start Menu.
- **Create a Postman Account (Optional)**
  - **Sign up for a Postman account**:
    - Once Postman is installed, open the application. You can **sign up for a free Postman account** or continue using Postman without an account. Signing up will allow you to sync collections across devices, but it is optional.

## Step 5: Run the Server:

- Run the server using below command. It will start server to communicate with the postman.

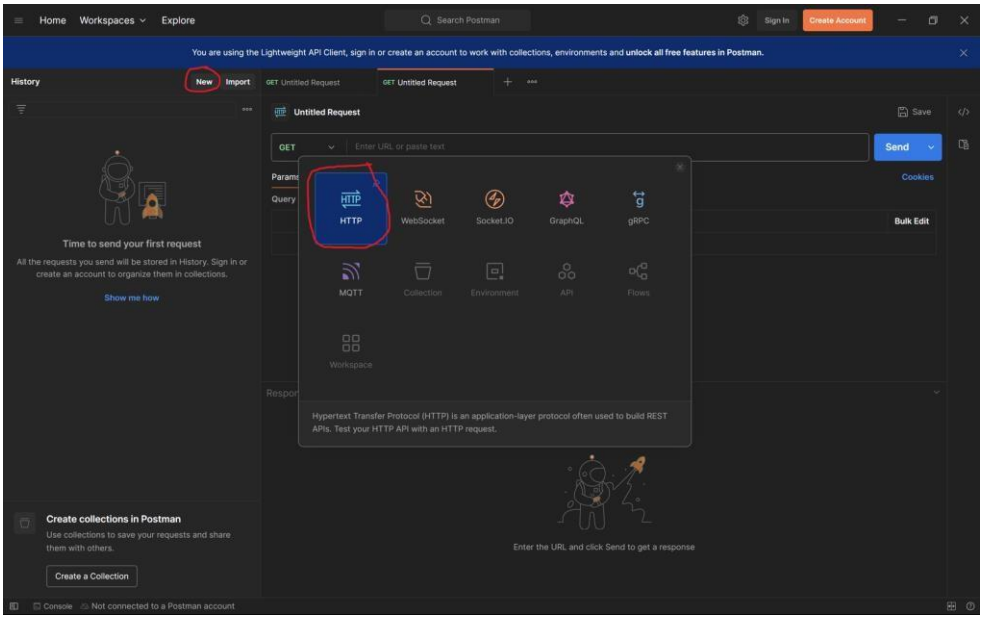
```
node server.js
```

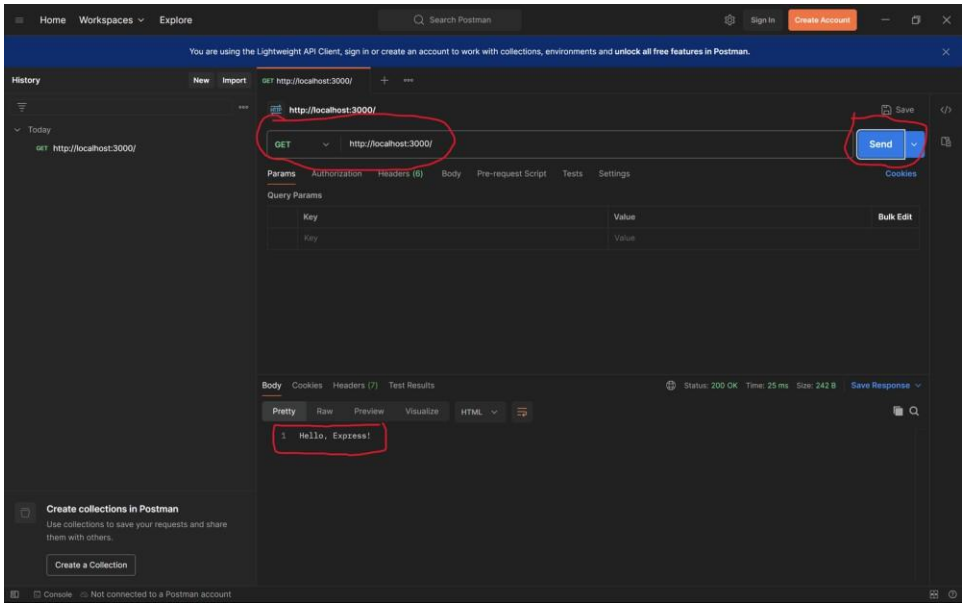

## Step 6: Open Postman:

- Once your server is up and running, follow these steps to test the various endpoints.

## Step 7: Create a New Request:

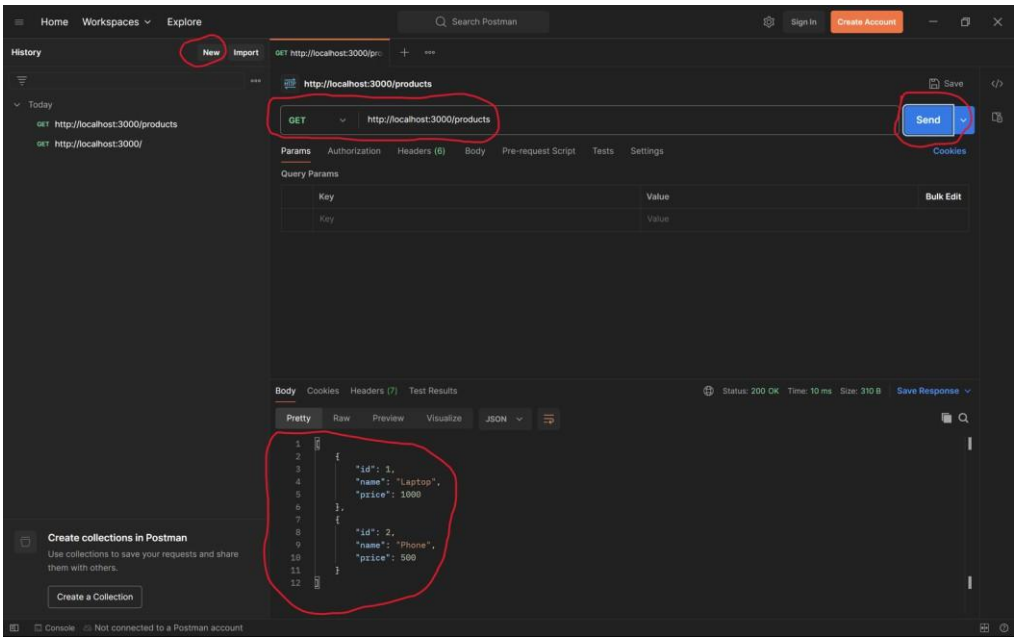
- **Open Postman**.
- Click on the **New** button on the top left, then select **Http** from the options.
- Enter **url** as per **screenshot** or **generated on the terminal** while **running server**.
- **Click on send button** and then you will get **output hello express** as per screenshot.






## Step 8: Test Fetching All Products: GET /products

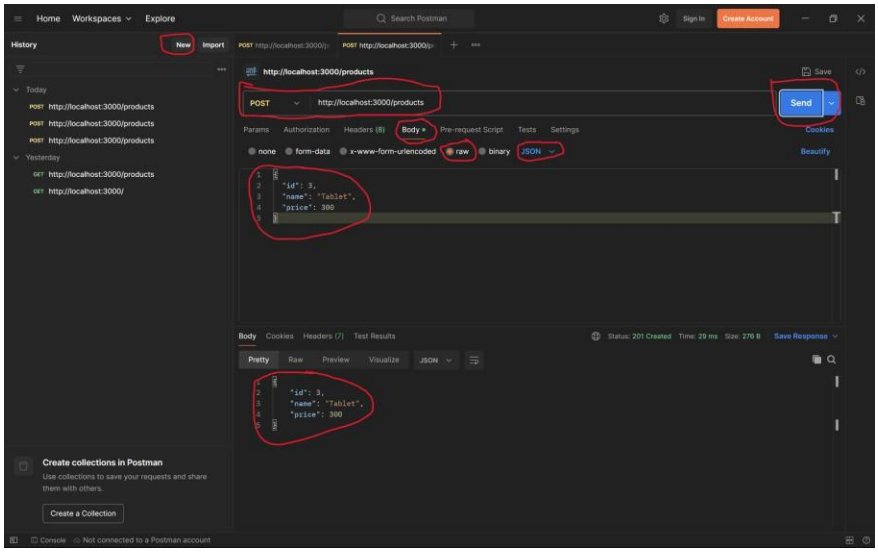
- **Steps to Test:**
  - Again follow step 7 and this time Choose **Method**: **GET** and the **url** is **http://localhost:3000/products**
  - Then click on send button to see the output of the all product.




## Step 9: Test Adding a New Product: POST /products

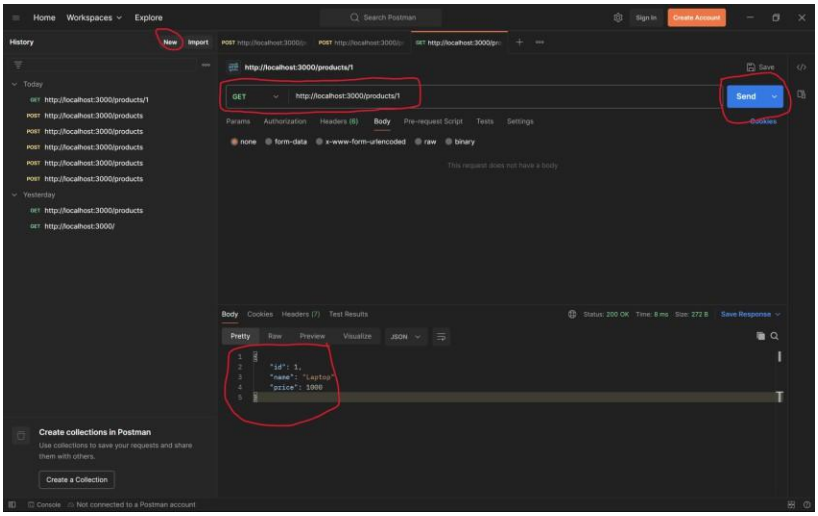
- **Steps to Test:**
  - Again follow step 7 and this time Choose **Method**: **POST** and the **url** is **http://localhost:3000/products**
  - Then click on send button you can see message like **"message": "Name and price are required"**
  - **Go to the Body tab** in Postman.
  - **Select raw and choose JSON** from the dropdown.
  - **Enter the following JSON data**:

```
{
  "id": 3,
  "name": "Tablet",
  "price": 300
}
```




## Step 10: Test Fetching a Specific Product: GET /products/:id

- **Steps to Test:**
  - **Method**: GET
  - **URL**: http://localhost:3000/products/1 (replace 1 with the ID of the product you want to fetch)
  - **Click Send**.



## Step 11: Test Updating a Product: PUT /products/:id

- **Steps to Test:**
  - **Method**: PUT
  - **URL**: http://localhost:3000/products/1 (replace 1 with the ID of the product you want to update)
  - **Go to the Body tab** in Postman.
  - **Select raw and choose JSON** from the dropdown.

**Enter the following JSON data**:

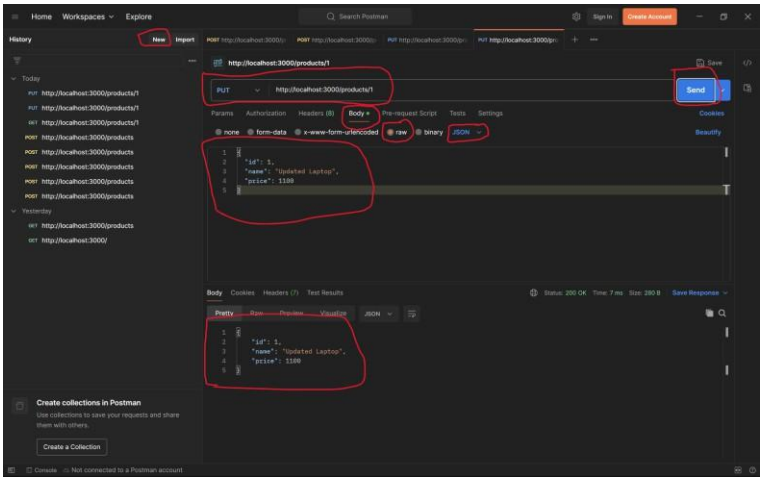
**{**

**"id": 1,**

**"name": "Updated Laptop",**

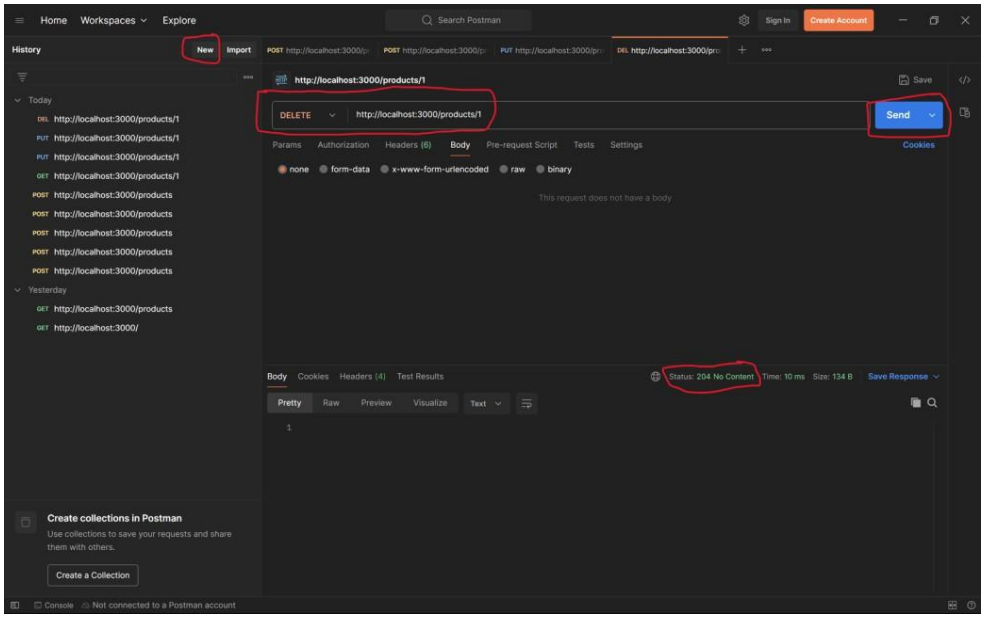
**"price": 1100**

**}**



## Step 12: Test Deleting a Product: DELETE /products/:id

- Steps to Test:
  - **Method**: DELETE
  - **URL**: http://localhost:3000/products/1 (replace 1 with the ID of the product you want to delete)
  - **Click Send**.
  - **Expected Response:**
  - You should get a **204 No Content** response, indicating that the product has been deleted successfully.



# Experiment 8

**Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.**

## PROGRAM:

### server.js

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

const app = express();
app.use(cors());
app.use(express.json());

// MongoDB Connection
mongoose.connect('mongodb://localhost:27017/shop', {
useNewUrlParser: true,
  useUnifiedTopology: true
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', () => console.log('Connected to MongoDB'));

// Define Mongoose Schema for Products
const productSchema = new mongoose.Schema({
  name: String,
  price: Number,
  description: String
});


const Product = mongoose.model('Product', productSchema);

// CRUD API Endpoints

// Fetch all products
app.get('/products', async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
```

```
  } catch (error) {
    res.status(500).send(error.message);
  }
});

// Add a new product
app.post('/products', async (req, res) => {
try {
    const newProduct = new Product(req.body);
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

// Update a product
app.put('/products/:id', async (req, res) => {
  try {
    const updatedProduct = await Product.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    );
    res.json(updatedProduct);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

// Delete a product
app.delete('/products/:id', async (req, res) => {
  try {
    await Product.findByIdAndDelete(req.params.id);
    res.status(204).send();
  } catch (error) {
    res.status(500).send(error.message);
  }
});
// Start Express server
const PORT = 4000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

## App.js:

```
import React, { useEffect, useState } from 'react';
import './App.css';

const App = () => {
  const [products, setProducts] = useState([]);
  const [newProduct, setNewProduct] = useState({
  name: '',
    price: '',
    description: ''
  });
  const [editProduct, setEditProduct] = useState(null);
  const [errors, setErrors] = useState({});

  const fetchProducts = () => {
    fetch('http://localhost:4000/products')
      .then(response => response.json())
      .then(data => setProducts(data));
  };
  useEffect(() => {
    fetchProducts();
  }, []);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    if (editProduct) {
      setEditProduct({ ...editProduct, [name]: value });
    } else {
      setNewProduct({ ...newProduct, [name]: value });
    }
    setErrors({ ...errors, [name]: '' });
  };

const validateInputs = (product) => {
    const newErrors = {};
    if (!product.name) newErrors.name = 'Product name is required.';
    if (!product.price) newErrors.price = 'Product price is required.';
    if (!product.description) newErrors.description = 'Product description is required.';
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const addProduct = () => {
    if (!validateInputs(newProduct)) return;

    fetch('http://localhost:4000/products', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
```

```
    },
    body: JSON.stringify(newProduct)
  })
    .then(response => response.json())
    .then(() => {
      fetchProducts();
      setNewProduct({ name: '', price: '', description: '' });
    });
};

const updateProduct = () => {
  if (!validateInputs(editProduct)) return;

  fetch(`http://localhost:4000/products/${editProduct._id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(editProduct)
  })
    .then(response => response.json())
    .then(() => {
      fetchProducts();
      setEditProduct(null);
    });
};

const deleteProduct = (productId) => {
  if (window.confirm('Are you sure you want to delete this product?')) {
    fetch(`http://localhost:4000/products/${productId}`, {
      method: 'DELETE'
    })
      .then(() => fetchProducts());
  }
};

return (
  <div className="container">
    <h1 className="title">Product Management</h1>

    {products.length === 0 ? (
      <p className="no-products-message">No products available. Please add a new product.</p>
    ) : (
      <table className="product-table">
        <thead>
          <tr>
            <th>Name</th>
            <th>Price</th>
            <th>Description</th>
            <th>Actions</th>
```

```
          </tr>
        </thead>
        <tbody>
          {products.map(product => (
            <tr key={product._id} className="product-row">
              <td>{product.name}</td>
              <td>₹{product.price}</td>
              <td>{product.description}</td>
              <td>
                <div className="action-buttons">
                  <button onClick={() => setEditProduct(product)} className="edit-button">Edit</button>
                  <button onClick={() => deleteProduct(product._id)} className="delete-
button">Delete</button>
                </div>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    )}

    <div className="form-section">
      <h2 className="form-title">{editProduct ? 'Edit Product' : 'Add New Product'}</h2>
      <div className="input-group">
        <div className="input-wrapper">
          <input
            type="text"
            name="name"
            value={editProduct ? editProduct.name : newProduct.name}
            onChange={handleInputChange}
            placeholder="Product Name"
            className="input-field"
          />
          {errors.name && <p className="error-message">{errors.name}</p>}
        </div>


<div className="input-wrapper">
          <input
            type="number"
            name="price"
            value={editProduct ? editProduct.price : newProduct.price}
            onChange={handleInputChange}
            placeholder="Product Price ₹"
            className="input-field"
          />
          {errors.price && <p className="error-message">{errors.price}</p>}
        </div>

        <div className="input-wrapper">
```

```
            <textarea
              name="description"
              value={editProduct ? editProduct.description : newProduct.description}
              onChange={handleInputChange}
              placeholder="Product Description"
              className="input-field"
              rows={3}
            />
            {errors.description && <p className="error-message">{errors.description}</p>}
          </div>
        </div>

        {editProduct ? (
          <button onClick={updateProduct} className="update-button">Update Product</button>
        ) : (
          <button onClick={addProduct} className="add-button">Add Product</button>
        )}
      </div>
    </div>
  );
};
export default App;
```

## App.css:

```
.container {
  max-width: 1000px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  background-color: #ffffff;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
}
.title {
  border-radius: 7px;
  color: #fff;
  background: #000;
  padding: 10px 0;
  font-size: 1.6rem;
```

```
    text-align: center;
    margin: 0;
  }
  .product-table {
    width: 100%;
    border-collapse: collapse;
    margin: 20px 0;
    background-color: #f9f9f9;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.05);
    border-radius: 8px;
    overflow: hidden;
  }
  .product-table thead {
    background-color: #007bff;
    color: white;
    text-align: left;
    font-size: 1rem;
  }
  .product-table th,
  .product-table td {
    padding: 12px 15px;
    border: 1px solid #ddd;
  }
  .product-table tr {
    transition: background-color 0.2s ease-in-out;
  }
  td button {
    display: block;
  }
.product-table tbody tr:nth-child(even) {
```

```
  background-color: #f9f9f9;
}
.input-group {
  display: flex;
  flex-direction: column;
  gap: 10px;
}
.input-field {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
.add-button,
.edit-button,
.update-button {
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.2s;
}
.add-button {
  font-weight: 500;
  margin-top: 10px;
  background-color: #28a745;
  color: white;
}
.add-button:hover {
  background-color: #218838;
}
```

```
.edit-button {
  background-color: #007bff;
  color: white;
}
.edit-button:hover {
  background-color: #0056b3;
}
.update-button {
  margin-top: 20px;
  background-color: #ffc107;
  color: black;
  font-weight: 600;
}
.update-button:hover {
  background-color: #e0a800;
}
.action-buttons {
  display: flex;
  gap: 10px;
  justify-content: center;
  align-items: center;
}
.form-section {
  margin-top: 40px;
}
.form-title {
  width: fit-content;
  background: #0000001a;
  padding: 7px 7px;
  border-radius: 7px;
```

```
  font-size: 16px;
  color: #000000;
}
.delete-button {
  background-color: #dc3545;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
.delete-button:hover {
  background-color: #c82333;
}
.no-products-message {
  text-align: center;
  font-size: 1.2rem;
  color: #666;
  margin-top: 20px;
  padding: 10px;
  background-color: #f9f9f9;
  border-radius: 8px;
}
.error-message {
  color: red;
  background-color: #ffe6e6;
  padding: 5px;
  border-radius: 5px;
  margin-top: 5px;
  font-size: 0.9rem;
```

```
}
.input-wrapper {
  display: flex;
  flex-direction: column;
  justify-content:  center;
  margin-bottom: 10px;
}
```

## Step 1: Create Project:

- **Create a folder** and open with **vscode**.
- Open **integrated vscode terminal**.

## Step 2: Set up the Backend:

- Run the following commands to set up the environment for your project:

Run one by one command to install the mongoose.

**mkdir shop-backend**

**cd shop-backend**

## Step 3: Initialize Node.js Project:

**npm init -y**

## Step 4: Install Dependencies:

**npm install express mongoose mongodb cors**

## Step 5: Create server.js file:

- **Right click on shop-backend folder** and create the **server.js** file.
  Copy the below **code and paste** it in that file, save it.

## Step 6: Start the backend:

**Start the backend server** using below command to see in the terminal **message server is running on port 4000**, **connected to MongoDB**.

**node server.js**

## Step 7: Set up the Frontend:

Open new terminal without deleting running backend server terminal.

Execute 1st command to create frontend folder.

After successfully creating product-client folder then change the directory using 2nd command.

**npx create-react-app product-client**

**cd product-client**

## Step 8: Edit App.js and App.css file:

- Copy the below code and paste it into the **App.js** file and then save it.
- After then Copy the below code and paste it into the **App.css** file and then save it.

## Step 9: Start the development server:

Copy the below command to run the frontend to see the app.

**npm start**

**"Fixing the Module not found: Error: Can't resolve 'web-vitals' Error in React"**

The error you're seeing occurs because the web-vitals package, which is used for performance monitoring in a React app, is not installed by default in the project or has been removed. Since web-vitals is an optional package, you can safely resolve this issue by either installing the package or removing the code that imports it.

## Option 1: Install the web-vitals package

If you want to keep the performance monitoring functionality and resolve the error, simply install the web-vitals package.

In the terminal, navigate to your project folder (if not already there):

**cd product-client**

**Install web-vitals by running the following command:**

**npm install web-vitals**

**After installation is complete, restart the development server:**

**npm start**

**This should resolve the error, and your application should compile correctly.**

### Option 2: Remove the Web Vitals Code (If Not Needed)

If you don't need performance monitoring and want to get rid of the error, you can safely remove the import and usage of web-vitals from your code.

Open src/reportWebVitals.js and remove its contents or just comment out the code:

**// import { reportWebVitals } from './reportWebVitals';**

**// You can safely remove the call to reportWebVitals or leave it commented out**

**// reportWebVitals();**

- Save the file, and the application should compile without the error. You can now continue developing your app.

```
return (
  <div>
    <h2>Data Collection</h2>
    <button onClick={fetchData}>Fetch Data</button>
    <ul>
      {data.map((item, index) => (
        <li key={index}>{item.employee_name}</li>
      ))}
    </ul>
  </div>
);
}
```

export default DataCollector;

**OUTPUT:**