MERN Stack Assignment Set

Module 1 – SE - Overview of IT Industry

What is a Program?

LAB EXERCISE: Write a simple "Hello World" program in two different programming languages ofyour choice. Compare the structure and syntax.

THEORY EXERCISE: Explain in your own words what a program is and how it functions.

What is Programming?

THEORY EXERCISE: What are the key steps involved in the programming process?

Types of Programming Languages

THEORY EXERCISE: What are the main differences between high-level and low-level programminglanguages?

World Wide Web & How Internet Works

LAB EXERCISE: Research and create a diagram of how data is transmitted from a client to a serverover the internet.

THEORY EXERCISE: Describe the roles of the client and server in web communication.

Network Layers on Client and Server

LAB EXERCISE: Design a simple HTTP client-server communication in any language.

THEORY EXERCISE: Explain the function of the TCP/IP model and its layers.

Client and Servers

THEORY EXERCISE: Explain Client Server Communication

Types of Internet Connections

LAB EXERCISE: Research different types of internet connections (e.g., broadband, fiber, satellite)and list their pros and cons.

THEORY EXERCISE: How does broadband differ from fiber-optic internet?

Protocols

LAB EXERCISE: Simulate HTTP and FTP requests using command line tools (e.g., curl).

THEORY EXERCISE: What are the differences between HTTP and HTTPS protocols?

Application Security

LAB EXERCISE: Identify and explain three common application security vulnerabilities. Suggestpossible solutions.

THEORY EXERCISE: What is the role of encryption in securing applications

Software Applications and Its Types

LAB EXERCISE: Identify and classify 5 applications you use daily as either system software orapplication software.

THEORY EXERCISE: What is the difference between system software and application software?

Software Architecture

LAB EXERCISE: Design a basic three-tier software architecture diagram for a web application.

THEORY EXERCISE: What is the significance of modularity in software architecture?

Layers in Software Architecture

LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and dataaccess layers of a given software system.

THEORY EXERCISE: Why are layers important in software architecture?

Software Environments

LAB EXERCISE: Explore different types of software environments (development, testing, production).Set up a basic environment in a virtual machine.

THEORY EXERCISE: Explain the importance of a development environment in software production.

Source Code

LAB EXERCISE: Write and upload your first source code file to Github.

THEORY EXERCISE: What is the difference between source code and machine code?

Github and Introductions

LAB EXERCISE: Create a Github repository and document how to commit and push code changes.

THEORY EXERCISE: Why is version control important in software development?

Student Account in Github

LAB EXERCISE: Create a student account on Github and collaborate on a small project with aclassmate.

THEORY EXERCISE: What are the benefits of using Github for students?

Types of Software

LAB EXERCISE: Create a list of software you use regularly and classify them into the followingcategories: system, application, and utility software.

THEORY EXERCISE: What are the differences between open-source and proprietary software?

GIT and GITHUB Training

LAB EXERCISE: Follow a GIT tutorial to practice cloning, branching, and merging repositories.

THEORY EXERCISE: How does GIT improve collaboration in a software development team?

Application Software

LAB EXERCISE: Write a report on the various types of application software and how they improveproductivity.

THEORY EXERCISE: What is the role of application software in businesses?

Software Development Process

LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDLC).

THEORY EXERCISE: What are the main stages of the software development process?

Software Requirement

LAB EXERCISE: Write a requirement specification for a simple library management system.

THEORY EXERCISE: Why is the requirement analysis phase critical in software development?

Software Analysis

LAB EXERCISE: Perform a functional analysis for an online shopping system.

THEORY EXERCISE: What is the role of software analysis in the development process?

System Design

LAB EXERCISE: Design a basic system architecture for a food delivery app.

THEORY EXERCISE: What are the key elements of system design?

Software Testing

LAB EXERCISE: Develop test cases for a simple calculator program.

THEORY EXERCISE: Why is software testing important?

Maintenance

LAB EXERCISE: Document a real-world case where a software application required criticalmaintenance.

THEORY EXERCISE: What types of software maintenance are there?

Development

THEORY EXERCISE: What are the key differences between web and desktop applications?

- Web Application

THEORY EXERCISE: What are the advantages of using web applications over desktop applications?

- Designing

THEORY EXERCISE: What role does UI/UX design play in application development?

- Mobile Application

THEORY EXERCISE: What are the differences between native and hybrid mobile apps?

- DFD (Data Flow Diagram)

LAB EXERCISE: Create a DFD for a hospital management system.

THEORY EXERCISE: What is the significance of DFDs in system analysis?

- Desktop Application

LAB EXERCISE: Build a simple desktop calculator application using a GUI library.

THEORY EXERCISE: What are the pros and cons of desktop applications compared to webapplications?

- Flow Chart

LAB EXERCISE: Draw a flowchart representing the logic of a basic online registration system.

THEORY EXERCISE: How do flowcharts help in programming and system design?

Module 2 – Mernstack - HTML

- **HTML Basics**

Theory Assignment

- **Question 1**: Define HTML. What is the purpose of HTML in web development?
- **Question 2**: Explain the basic structure of an HTML document. Identify the mandatory tagsand their purposes.
- **Question 3**: What is the difference between block-level elements and inline elements inHTML? Provide examples of each.
- **Question 4**: Discuss the role of semantic HTML. Why is it important for accessibility andSEO? Provide examples of semantic elements.

Lab Assignment

- **Task**: Create a simple HTML webpage that includes:
  - A header (<header>), footer (<footer>), main section (<main>), and aside section (<aside>).
  - A paragraph with some basic text.
  - A list (both ordered and unordered).
  - A link that opens in a new tab.

- **HTML Forms**

Theory Assignment

- **Question 1**: What are HTML forms used for? Describe the purpose of the input, textarea, select, and buttonelements.
- **Question 2**: Explain the difference between the GETand POSTmethods in form submission. When should each be used?
- **Question 3**: What is the purpose of the labelelement in a form, and how does it improve accessibility?

Lab Assignment

- **Task**: Create a contact form with the following fields:
  - Full name (text input)
  - Email (email input)
  - Phone number (tel input)

  - Subject (dropdown menu)
  - Message (textarea)
  - Submit button

  Additional Requirements:

  - Use appropriate form validation using required, minlength, maxlength, and pattern.
  - Link form labels with their corresponding inputs using the forattribute.

- **HTML Tables**

Theory Assignment

- **Question 1**: Explain the structure of an HTML table and the purpose of each of the followingelements: <table>, <tr>, <th>, <td>, and <thead>.

- **Question 2**: What is the difference between `colspan` and `rowspan` in tables? Provide examples.
- **Question 3**: Why should tables be used sparingly for layout purposes? What is a better alternative?

Lab Assignment

- **Task**: Create a product catalog table that includes the following columns:
  - Product Name
  - Product Image (use placeholder image URLs)
  - Price
  - Description
  - Availability (in stock, out of stock)

  Additional Requirements:

  - Use `thead` for the table header.
  - Add a border and some basic styling using inline CSS.
  - Use `colspan` or `rowspan` to merge cells where applicable.

Module 3 – Mernstack – CSS and CSS3

CSS Selectors & Styling

Theory Assignment

- **Question 1**: What is a CSS selector? Provide examples of element, class, and ID selectors.
- **Question 2**: Explain the concept of CSS specificity. How do conflicts between multiple styles get resolved?
- **Question 3**: What is the difference between internal, external, and inline CSS? Discuss the advantages and disadvantages of each approach.

Lab Assignment

- **Task**: Style the contact form (created in the HTML Forms lab) using external CSS. The following should be implemented:
  - Change the background color of the form.
  - Add padding and margins to form fields.
  - Style the submit button with a hover effect.
  - Use class selectors for styling common elements and ID selectors for unique elements.

- **CSS Box Model**

Theory Assignment

- **Question 1**: Explain the CSS box model and its components (content, padding, border,margin). How does each affect the size of an element?
- **Question 2**: What is the difference between `border-box`and `content-box`box-sizing inCSS? Which is the default?

Lab Assignment

- **Task**: Create a profile card layout using the box model. The profile card shouldinclude:
  - A profile picture.
  - The user's name and bio.
  - A button to "Follow" the user.

  Additional Requirements:

  - Add padding and borders to the elements.
  - Ensure the layout is clean and centered on the page using CSS margins.
  - Use the `box-sizing`property to demonstrate both `content-box`and `border-box`on different elements.

- **CSS Flexbox**

Theory Assignment

- **Question 1**: What is CSS Flexbox, and how is it useful for layout design? Explain the terms `flex-container` and `flex-item`.
- **Question 2**: Describe the properties `justify-content`, `align-items`, and `flex-direction` used in Flexbox.

Lab Assignment

- **Task**: Create a simple webpage layout using Flexbox. The layout should include:
  - A header.
  - A sidebar on the left.
  - A main content area in the center.
  - A footer.

  Additional Requirements:

  - Use Flexbox to position and align the elements.
  - Apply different `justify-content`and `align-items`properties to observe theireffects.
  - Ensure the layout is responsive, adjusting for smaller screens.

- **CSS Grid**

Theory Assignment

- **Question 1**: Explain CSS Grid and how it differs from Flexbox. When would you use Grid overFlexbox?
- **Question 2**: Describe the grid-template-columns, grid-template-rows, and grid-gapproperties. Provide examples of how to use them.

Lab Assignment

- **Task**: Create a 3x3 grid of product cards using CSS Grid. Each card should contain:
  - A product image.
  - A product title.
  - A price.

  Additional Requirements:

  - Use grid-template-columnsto create the grid layout.
  - Use grid-gapto add spacing between the grid items.
  - Apply hover effects to each card for better interactivity.

- **Responsive Web Design with Media Queries**

Theory Assignment

- **Question 1**: What are media queries in CSS, and why are they important for responsivedesign?
- **Question 2**: Write a basic media query that adjusts the font size of a webpage for screenssmaller than 600px

Lab Assignment

- **Task**: Build a responsive webpage that includes:
  - A navigation bar.
  - A content section with two columns.
  - A footer.

  Additional Requirements:

  - Use media queries to make the webpage responsive for mobile devices.
  - On smaller screens (below 768px), stack the columns vertically.
  - Adjust the font sizes and padding to improve readability on mobile.

- **Typography and Web Fonts**

Theory Assignment

- **Question 1**: Explain the difference between web-safe fonts and custom web fonts. Whymight you use a web-safe font over a custom font?
- **Question 2**: What is the font-familyproperty in CSS? How do you apply a custom Google Font to a webpage?

Lab Assignment

- **Task**: Create a blog post layout with the following:
  - A title, subtitle, and body content.
  - Use at least two different fonts (one for headings, one for body content).
  - Style the text to be responsive and easy to read.

  Additional Requirements:

  - Use a custom font from Google Fonts.
  - Adjust line-height, font-size, and spacing for improved readability.

Module 5 – Mernstack – HTML5

Theory Assignment

- **Question 1**: Difference b/w HTML & HTML5?
- **Question 2**: What are the additional tags used in HTML5?

Lab Assignment

- **Task**: Create a audio video tag
  - Also applied properties like muted loop autoplay
  - Create some shape using canvas tag in html
  - Create some shape using svg tag in html

Module 6 – Mernstack – Javascript Essential and Advanced

- **JavaScript Introduction**

Theory Assignment

- **Question 1**: What is JavaScript? Explain the role of JavaScript in web development.
- **Question 2**: How is JavaScript different from other programming languages like Python orJava?
- **Question 3**: Discuss the use of <script>tag in HTML. How can you link an external JavaScript file to an HTML document?

Lab Assignment

- Task:
  - Create a simple HTML page and add a <script>tag within the page.
  - Write JavaScript code to display an alert box with the message "Welcome toJavaScript!" when the page loads.

- **Variables and Data Types**

Theory Assignment

- **Question 1**: What are variables in JavaScript? How do you declare a variable using var, let, and const?
- **Question 2**: Explain the different data types in JavaScript. Provide examples for each.
- **Question 3**: What is the difference between undefinedand nullin JavaScript?

Lab Assignment

- Task:
  - Write a JavaScript program to declare variables for different data types (string,number, boolean, null, and undefined).
  - Log the values of the variables and their types to the console using console.log().

- **JavaScript Operators**

Theory Assignment

- **Question 1**: What are the different types of operators in JavaScript? Explain with examples.
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators

- **Question 2**: What is the difference between ==and ===in JavaScript?

Lab Assignment

- Task:
  - Create a JavaScript program to perform the following:
    - Add, subtract, multiply, and divide two numbers using arithmetic operators.
    - Use comparison operators to check if two numbers are equal and if onenumber is greater than the other.
    - Use logical operators to check if both conditions (e.g., a > 10and b < 5)are true.

- **Control Flow (If-Else, Switch)**

Theory Assignment

- **Question 1**: What is control flow in JavaScript? Explain how if-elsestatements work withan example.
- **Question 2**: Describe how switchstatements work in JavaScript. When should you use a switchstatement instead of if-else?

Lab Assignment

- Task 1:
  - Write a JavaScript program to check if a number is positive, negative, or zero usingan if-elsestatement.
- Task 2:
  - Create a JavaScript program using a switchstatement to display the day of the week based on the user input (e.g., 1 for Monday, 2 for Tuesday, etc.).

- **Loops (For, While, Do-While)**

Theory Assignment

- **Question 1**: Explain the different types of loops in JavaScript (for, while, do-while). Provide abasic example of each.
- **Question 2**: What is the difference between a whileloop and a do-whileloop?

Lab Assignment

- *Task 1:*
- Write a JavaScript program using a forloop to print numbers from 1 to 10.

- *Task 2:*
  - Create a JavaScript program that uses a `while` loop to sum all even numbers between 1 and 20.
- *Task 3:*
  - Write a `do-while` loop that continues to ask the user for input until they enter a number greater than 10.

- **Functions**

Theory Assignment

- **Question 1**: What are functions in JavaScript? Explain the syntax for declaring and calling a function.
- **Question 2**: What is the difference between a function declaration and a function expression?
- **Question 3**: Discuss the concept of parameters and return values in functions.

Lab Assignment

- Task 1:
  - Write a function `greetUser` that accepts a user's name as a parameter and displays a greeting message (e.g., "Hello, John!").
- Task 2:
  - Create a JavaScript function `calculateSum` that takes two numbers as parameters, adds them, and returns the result.

- **Arrays**

Theory Assignment

- **Question 1**: What is an array in JavaScript? How do you declare and initialize an array?
- **Question 2**: Explain the methods `push()`, `pop()`, `shift()`, and `unshift()` used in arrays.

Lab Assignment

- Task 1:
  - Declare an array of fruits (`["apple", "banana", "cherry"]`). Use JavaScript to:
    - Add a fruit to the end of the array.
    - Remove the first fruit from the array.
    - Log the modified array to the console.
- Task 2:
  - Write a program to find the sum of all elements in an array of numbers.

- **Objects**

Theory Assignment

- **Question 1**: What is an object in JavaScript? How are objects different from arrays?
- **Question 2**: Explain how to access and update object properties using dot notation andbracket notation.

Lab Assignment

- Task:
  - Create a JavaScript object carwith properties brand, model, and year. Use JavaScript to:
    - Access and print the car's brand and model.
    - Update the yearproperty.
    - Add a new property colorto the car object.

- **JavaScript Events**

Theory Assignment

- **Question 1**: What are JavaScript events? Explain the role of event listeners.
- **Question 2**: How does the addEventListener()method work in JavaScript? Provide an example.

Lab Assignment

- Task:
  - Create a simple webpage with a button that, when clicked, displays an alert saying"Button clicked!" using JavaScript event listeners.

- **DOM Manipulation**

Theory Assignment

- **Question 1**: What is the DOM (Document Object Model) in JavaScript? How does JavaScriptinteract with the DOM?
- **Question 2**: Explain the methods getElementById(), getElementsByClassName(),and querySelector()used to select elements from the DOM.

Lab Assignment

- Task:
  - Create an HTML page with a paragraph (<p>) that displays "Hello, World!".
  - Use JavaScript to:
    - Change the text inside the paragraph to "JavaScript is fun!".
    - Change the color of the paragraph to blue.

- **JavaScript Timing Events (setTimeout, setInterval)**

Theory Assignment

- **Question 1**: Explain the setTimeout()and setInterval()functions in JavaScript. Howare they used for timing events?
- **Question 2**: Provide an example of how to use setTimeout()to delay an action by 2 seconds.

Lab Assignment

- *Task 1:*
  - Write a program that changes the background color of a webpage after 5 secondsusing setTimeout().
- *Task 2:*
- Create a digital clock that updates every second using setInterval().

- **JavaScript Error Handling**

Theory Assignment

- **Question 1**: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.
- **Question 2**: Why is error handling important in JavaScript applications?

Lab Assignment

- Task:
  - Write a JavaScript program that attempts to divide a number by zero. Use try-catchto handle the error and display an appropriate error message.

Module 7 – Mernstack – React JS

THEORY EXERCISE

- **Question 1**: What is React.js? How is it different from other JavaScript frameworks andlibraries?

- **Question 2**: Explain the core principles of React such as the virtual DOM and component-based architecture.
- **Question 3**: What are the advantages of using React.js in web development?

LAB EXERCISE

- Task:
  - Set up a new React.js project using `create-react-app`.
  - Create a basic component that displays "Hello, React!" on the web page.

- JSX (JavaScript XML)

THEORY EXERCISE

- **Question 1**: What is JSX in React.js? Why is it used?
- **Question 2**: How is JSX different from regular JavaScript? Can you write JavaScript insideJSX?
- **Question 3**: Discuss the importance of using curly braces {}in JSX expressions.

LAB EXERCISE

- Task:
  - Create a React component that renders the following JSX elements:
    - A heading with the text "Welcome to JSX".
    - A paragraph explaining JSX with dynamic data (use curly braces to insertvariables).

- Components (Functional & Class Components)

THEORY EXERCISE

- **Question 1**: What are components in React? Explain the difference between **functionalcomponents** and **class components**.
- **Question 2**: How do you pass data to a component using props?
- **Question 3**: What is the role of `render()`in class components?

LAB EXERCISE

- Task 1:
  - Create a functional component `Greeting`that accepts a `name`as a prop and displays "Hello, [name]!".
- Task 2:
  - Create a class component `WelcomeMessage`that displays "Welcome to React!" and a `render()`method.

- Props and State

THEORY EXERCISE

- **Question 1**: What are props in React.js? How are props different from state?
- **Question 2**: Explain the concept of state in React and how it is used to manage componentdata.
- **Question 3**: Why is this.setState()used in class components, and how does it work?

LAB EXERCISE

- Task 1:
  - Create a React component UserCardthat accepts name, age, and locationasprops and displays them in a card format.
- Task 2:
  - Create a Countercomponent with a button that increments a count value using React state. Display the current count on the screen.

- Handling Events in React

THEORY EXERCISE

- **Question 1**: How are events handled in React compared to vanilla JavaScript? Explain theconcept of synthetic events.
- **Question 2**: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.
- **Question 3**: Why do you need to bind event handlers in class components?

LAB EXERCISE

- Task 1:
  - Create a button in a React component that, when clicked, changes the text from"Not Clicked" to "Clicked!" using event handling.
- Task 2:
  - Create a form with an input field in React. Display the value of the input fielddynamically as the user types in it.

- Conditional Rendering

THEORY EXERCISE

- **Question 1**: What is conditional rendering in React? How can you conditionally renderelements in a React component?
- **Question 2**: Explain how if-else, ternary operators, and &&(logical AND) are used in JSX for conditional rendering.

LAB EXERCISE

- Task 1:
  - Create a component that conditionally displays a login or logout button based onthe user's login status.
- Task 2:
  - Implement a component that displays a message like "You are eligible to vote" if theuser is over 18, otherwise display "You are not eligible to vote."

- Lists and Keys

THEORY EXERCISE

- **Question 1**: How do you render a list of items in React? Why is it important to use keyswhen rendering lists?
- **Question 2**: What are keys in React, and what happens if you do not provide a unique key?

LAB EXERCISE

- Task 1:
  - Create a React component that renders a list of items (e.g., a list of fruit names). Usethe map()function to render each item in the list.
- Task 2:
  - Create a list of users where each user has a unique id. Render the user list usingReact and assign a unique keyto each user.

- Forms in React

THEORY EXERCISE

- **Question 1**: How do you handle forms in React? Explain the concept of controlledcomponents.
- **Question 2**: What is the difference between controlled and uncontrolled components inReact?

LAB EXERCISE

- Task 1:
  - Create a form with inputs for name, email, and password. Use state to control the form and display the form data when the user submits it.
- Task 2:
  - Add validation to the form created above. For example, ensure that the emailinput contains a valid email address.

- Lifecycle Methods (Class Components)

THEORY EXERCISE

- **Question 1**: What are lifecycle methods in React class components? Describe the phases of acomponent's lifecycle.
- **Question 2**: Explain the purpose of componentDidMount(), componentDidUpdate(),and componentWillUnmount().

LAB EXERCISE

- Task 1:
  - Create a class component that fetches data from an API when the componentmounts using componentDidMount(). Display the data in the component.
- Task 2:
  - Implement a component that logs a message to the console when it updates using componentDidUpdate(). Log another message when the component unmountsusing componentWillUnmount().

- Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)

THEORY EXERCISE

- **Question 1**: What are React hooks? How do useState()and useEffect()hooks work in functional components?
- **Question 2**: What problems did hooks solve in React development? Why are hooksconsidered an important addition to React?
- **Question 3**: What is useReducer ? How we use in react app?
- **Question 4:** What is the purpose of useCallback & useMemo Hooks?
- **Question 5:** What's the Difference between the useCallback & useMemo Hooks?
- **Question 6 :** What is useRef ? How to work in react app?

LAB EXERCISE

- Task 1:
  - Create a functional component with a counter using the useState()hook. Include buttons to increment and decrement the counter.
- Task 2:
  - Use the useEffect()hook to fetch and display data from an API when the component mounts.
- Task 3:
  - Create react app with use of useSelector & useDispatch.
- Task 4:
  - Create react app to avoid re-renders in react application by useRef ?

- Routing in React (React Router)

THEORY EXERCISE

- **Question 1**: What is React Router? How does it handle routing in single-page applications?
- **Question 2**: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.

LAB EXERCISE

- Task 1:
  - Set up a basic React Router with two routes: one for a Home page and one for anAbout page. Display the appropriate content based on the URL.
- Task 2:
  - Create a navigation bar using React Router's Linkcomponent that allows users to switch between the Home, About, and Contact pages.

- React – JSON-server and Firebase Real Time Database

THEORY EXERCISE

- **Question 1:** What do you mean by RESTful web services?
- **Question 2:** What is Json-Server? How we use in React ?
- **Question 3**: How do you fetch data from a Json-server API in React? Explain the role of fetch() or axios()in making API requests.
- **Question 4:** What is Firebase? What features does Firebase offer?
- **Question 5**: Discuss the importance of handling errors and loading states when working withAPIs in React

LAB EXERCISE

- Task 1:
  - Create a React component that fetches data from a public API (e.g., a list of users)and displays it in a table format.
  - Create a React app with Json-server and use Get , Post , Put , Delete & patch method on Json-server API.
- Task 2:
  - Create a React app crud and Authentication with firebase API.
  - Implement google Authentication with firebase API.
- Task 3:
  - Implement error handling and loading states for the API call. Display a loadingspinner while the data is being fetched.

- Context API

THEORY EXERCISE

- **Question 1**: What is the Context API in React? How is it used to manage global state acrossmultiple components?
- **Question 2**: Explain how createContext()and useContext()are used in React forsharing state.

LAB EXERCISE

- Task 1:
  - Create a simple theme toggle (light/dark mode) using the Context API. The themestate should be shared across multiple components.
- Task 2:
  - Use the Context API to create a global user authentication system. If the user islogged in, display a welcome message; otherwise, prompt them to log in.

- State Management (Redux, Redux-Toolkit or Recoil)

THEORY EXERCISE

- **Question 1**: What is Redux, and why is it used in React applications? Explain the coreconcepts of actions, reducers, and the store.
- **Question 2**: How does Recoil simplify state management in React compared to Redux?

LAB EXERCISE

- Task 1:
  - Create a simple counter application using Redux for state management. Implementactions to increment and decrement the counter.
- Task 2:
  - Build a todo list application using Recoil for state management. Allow users to add,remove, and mark tasks as complete.
- Task 3:
  - Build a crud application using Redux-Toolkit for state management. Allow users to add,remove, delete and update.

Nodejs Assignments

Introduction to Node.js

Theory Assignment:

- Write an essay on the history and evolution of Node.js, discussing its architecture and key features.

- Compare Node.js with traditional server-side technologies like PHP and Java.

Practical Assignment:

- Install Node.js on your local machine and create a simple "Hello World" application. Include instructions for installation and running the application.

Node.js Environment Setup

Theory Assignment:

- Describe the role of npm (Node Package Manager) in Node.js development. Discuss common commands used in npm.

Practical Assignment:

- Create a project directory and initialize a new Node.js project using npm. Install at least two packages (e.g., Express, Nodemon) and demonstrate how to use them in your application.

Practical Assignment:

- Write a function that attempts to parse a JSON string and uses `try...catch` to handle any errors that may occur.

- Module System

Theory Assignment:

- Describe the module system in JavaScript, including CommonJS and ES Modules. Discuss their significance for code organization.

Practical Assignment:

- Create two separate JavaScript files: one for a module exporting functions and another for importing and using those functions.

- JavaScript APIs

Theory Assignment:

- Discuss what APIs are and how JavaScript can interact with them, focusing on Fetch API for making network requests.

Practical Assignment:

- Write a JavaScript program that fetches data from a public API (like JSONPlaceholder) and displays the results on a web page.

- **Fetch Data from a Public API**

Assignment:
Make a GET request to a public API and display the data.

Instructions:

- Use the Fetch API to get data from the JSONPlaceholder API (e.g., `/posts`).
- Parse the JSON response and log the title of each post to the console.

- **Display Data on a Web Page**

Assignment:
Fetch data from an API and display it on a web page.

Instructions:

- Create a simple HTML page with a `<div>` to display the data.
- Use the Fetch API to get data from the JSONPlaceholder API.
- Loop through the response and create HTML elements to display the titles and bodies of the posts in the `<div>`.

- **Search Functionality**

Assignment:
Implement a search feature that fetches data based on user input.

Instructions:

- Create a search input field and a button on your HTML page.
- When the button is clicked, fetch data from the JSONPlaceholder API based on the input (e.g., search for posts by user ID).
- Display the filtered results on the web page.

- **Error Handling**

Assignment:
Implement error handling for your API requests.

Instructions:

- Use the Fetch API to make a request to a public API (like JSONPlaceholder).
- Implement `.catch()` to handle any errors during the fetch operation.
- Display a user-friendly message if an error occurs.

- **POST Request to Create Data**

Assignment:
Make a POST request to an API to create new data.

Instructions:

- Create a form on your HTML page to collect user input (e.g., title and body for a new post).
- When the form is submitted, use the Fetch API to send a POST request to the JSONPlaceholder API.
- Display the response (the created post) on the web page.

• Update Existing Data with PUT

Assignment:
Make a PUT request to update an existing resource.

Instructions:

- Use the Fetch API to update a post in the JSONPlaceholder API (e.g., update a post with ID 1).
- Log the updated post to the console and display it on the web page.

• DELETE Request to Remove Data

Assignment:
Make a DELETE request to remove a resource.

Instructions:

- Use the Fetch API to send a DELETE request to the JSONPlaceholder API to delete a post.
- Log a confirmation message to the console and display the updated list of posts after deletion.

• Handling Promises with Async/Await

Assignment:
Use async/await syntax to handle API requests.

Instructions:

- Create a function that fetches data from the JSONPlaceholder API using async/await.
- Log the user data to the console and handle any errors.

• Using API Response Data

Assignment:
Process and use data received from an API.

Instructions:

- Fetch data from the OpenWeatherMap API (you'll need an API key).
- Extract the temperature and weather description from the response and log them to the console.

• Create a Weather App

Assignment:
Build a simple weather application that fetches and displays weather data.

Instructions:

- Create an input field to accept a city name and a button to fetch the weather data.
- Use the OpenWeatherMap API to get the weather data for the entered city when the button is clicked.
- Display the temperature, weather condition, and city name on the web page.

• Pagination with API Data

Assignment:
Implement pagination for API data.

Instructions:

- Fetch data from the JSONPlaceholder API.
- Display only a limited number of posts (e.g., 5) at a time on the web page.
- Create "Next" and "Previous" buttons to navigate between pages of results.

• Caching API Responses

Assignment:
Implement basic caching for API responses.

Instructions:

- Fetch data from the JSONPlaceholder API.
- Store the response data in a variable or use local storage to cache the data.
- Check if the data is cached before making a new request.

• Creating a Simple API Client

Assignment:
Create a simple API client with functions to GET, POST, PUT, and DELETE.

Instructions:

- Create a JavaScript object that contains methods for each of the HTTP methods (GET, POST, PUT, DELETE).

- Use the Fetch API within these methods to interact with the JSONPlaceholder API.
- Test each method by calling them and logging the results.

- **Form Validation before API Call**

Assignment:
Implement form validation before making an API call.

Instructions:

- Create a form to collect user data (e.g., name and email).
- Validate the form input to ensure the fields are not empty before making a POST request to a mock API (use JSONPlaceholder).
- Display appropriate error messages if validation fails.

- Functional Programming Concepts

Theory Assignment:

- Explain the principles of functional programming as they apply to JavaScript.
  Discuss concepts like immutability and pure functions.

Practical Assignment:

- Write a program that demonstrates functional programming techniques, such as using `map`, `filter`, and `reduce` on an array of numbers.

- Understanding Node.js Modules

Theory Assignment:

- Explain the CommonJS module system in Node.js. Discuss how it differs from ES modules.

Practical Assignment:

- Create a simple application that utilizes at least three custom modules. Each module should export a function that can be imported and used in the main application.

- Working with File Systems

Theory Assignment:

- Discuss the importance of the File System module in Node.js. Explain the difference between synchronous and asynchronous file operations.

Practical Assignment:

- Write a program that reads the contents of a text file and writes the output to another file. Implement both synchronous and asynchronous methods to perform the same task.

• Reading a File

Assignment:
Read the contents of a text file and log it to the console.

Instructions:

- Create a text file named `sample.txt` with some sample text.
- Use the `fs.readFile` method to read the file asynchronously.
- Log the contents of the file to the console.

• Writing to a File

Assignment:
Write data to a text file.

Instructions:

- Create a new text file named `output.txt`.
- Use the `fs.writeFile` method to write some sample data to the file.
- Log a success message to the console once the write operation is complete.

• Appending Data to a File

Assignment:
Append data to an existing file.

Instructions:

- Create an existing file named `append.txt` and add some initial content.
- Use the `fs.appendFile` method to add new content to the file.
- Log the contents of the file after appending to confirm the changes.

• Reading a File Line by Line

Assignment:
Read a text file line by line.

Instructions:

- Use the `fs.readFile` method to read the contents of a file.
- Split the file contents into an array of lines and log each line to the console.
- Use a loop to iterate through the lines and print them one by one.

- **Checking if a File Exists**

Assignment:
Check if a specific file exists in the directory.

Instructions:

- Use the `fs.existsSync` method to check for the existence of a file (e.g., `check.txt`).
- Log a message indicating whether the file exists or not.

- **Renaming a File**

Assignment:
Rename an existing file.

Instructions:

- Create a file named `oldname.txt`.
- Use the `fs.rename` method to change the file name to `newname.txt`.
- Log a success message once the rename operation is complete.

- **Deleting a File**

Assignment:
Delete a specific file from the directory.

Instructions:

- Create a file named `deleteMe.txt`.
- Use the `fs.unlink` method to delete the file.
- Log a success message upon successful deletion.

- **Creating a Directory**

Assignment:
Create a new directory.

Instructions:

- Use the `fs.mkdir` method to create a new directory named `newDirectory`.
- Log a success message once the directory is created.

- **Reading Files from a Directory**

Assignment:
List all files in a specific directory.

Instructions:

- Create a directory named `myFiles` and add several files to it.
- Use the `fs.readdir` method to read the contents of the directory.
- Log the names of all files to the console.

• Copying a File

Assignment:
Copy a file from one location to another.

Instructions:

- Create a file named `original.txt` with some sample content.
- Use the `fs.copyFile` method to create a copy of the file named `copy.txt`.
- Log a success message after the copy operation is complete.

• Watching for File Changes

Assignment:
Watch a file for changes and log the changes.

Instructions:

- Use the `fs.watch` method to monitor changes to a specific file (e.g., `watch.txt`).
- Log a message to the console whenever the file is modified.

• Reading a JSON File

Assignment:
Read a JSON file and parse its contents.

Instructions:

- Create a JSON file named `data.json` with some sample data.
- Use `fs.readFile` to read the file and parse its contents using `JSON.parse`.
- Log the parsed object to the console.

• Writing JSON Data to a File

Assignment:
Write a JavaScript object to a JSON file.

Instructions:

- Create a JavaScript object with sample data (e.g., user details).
- Use `fs.writeFile` to write the object to a file named `output.json` after converting it to a JSON string using `JSON.stringify`.

- Log a success message once the write operation is complete.

• Reading a File Asynchronously with Promises

Assignment:
Read a file using Promises and handle errors.

Instructions:

- Create a text file named `promiseFile.txt` with some content.
- Use the `fs.promises.readFile` method to read the file.
- Handle any potential errors using `.catch()` and log the contents to the console.

• Using Streams to Read and Write Files

Assignment:
Read and write files using streams.

Instructions:

- Create a large text file named `largeFile.txt` with repetitive content.
- Use `fs.createReadStream` to read the file in chunks and log each chunk to the console.
- Use `fs.createWriteStream` to create a new file named `outputStream.txt` and write data to it.

- HTTP and Web Servers

Theory Assignment:

- Describe how Node.js handles HTTP requests and responses. Discuss the request-response lifecycle.

Practical Assignment:

- Create a simple web server using the built-in http module. The server should respond with a "Hello World" message for GET requests.

• Creating a Basic HTTP Server

Assignment:
Create a simple HTTP server using Node.js.

Instructions:

- Use the `http` module to create a basic server that listens on a specified port (e.g., 3000).
- Respond with a plain text message like "Hello, World!" when accessed in a browser.
- Log a message to the console when the server starts.

- **Handling Different URL Routes**

Assignment:
Implement routing to handle different URL paths.

Instructions:

- Create an HTTP server that responds with different messages based on the requested URL path:
  - `/` should respond with "Welcome to the Home Page!"
  - `/about` should respond with "This is the About Page."
  - `/contact` should respond with "This is the Contact Page."
- Log the requested URL to the console.

- **Handling HTTP GET Requests**

Assignment:
Implement handling for GET requests.

Instructions:

- Create a server that responds to a GET request to `/api/users` with a JSON array of user objects.
- Use `res.setHeader` to set the response content type to `application/json`.
- Include at least three sample users in the JSON response.

- **Handling HTTP POST Requests**

Assignment:
Implement handling for POST requests.

Instructions:

- Set up a POST route (`/api/users`) to accept user data (e.g., name and email).
- Use the `body-parser` middleware to parse the incoming JSON data.
- Respond with a success message and the received user data in JSON format.

- **Serving Static Files**

Assignment:
Serve static files using Node.js.

Instructions:

- Create a directory named `public` and add some HTML, CSS, and JavaScript files.
- Use the `fs` module to serve the files in response to requests (e.g., `index.html` when accessing `/`).

- Ensure that the correct content type is set for each file type (e.g., `text/html`, `text/css`, `application/javascript`).

- **Creating a Simple RESTful API**

Assignment:
Create a RESTful API for managing a list of items.

Instructions:

- Set up an HTTP server that handles the following routes:
  - `GET /api/items` to return a list of items (array of objects).
  - `POST /api/items` to add a new item.
  - `PUT /api/items/:id` to update an existing item by ID.
  - `DELETE /api/items/:id` to delete an item by ID.
- Use a simple in-memory array to store the items.

- **Error Handling**

Assignment:
Implement error handling for your HTTP server.

Instructions:

- Add a middleware function to handle 404 errors for undefined routes.
- Log the error message and respond with a 404 status code and a friendly message ("Page not found!").
- Handle any potential server errors with appropriate responses.

- **Using Query Parameters**

Assignment:
Handle query parameters in your HTTP requests.

Instructions:

- Create an HTTP server that handles a GET request to `/api/search` with query parameters (e.g., `?q=node`).
- Respond with a JSON object containing the search query and a message (e.g., "You searched for: node").
- Log the received query parameters to the console.

- **Creating a Basic Form and Handling Form Submission**

Assignment:
Create a basic HTML form and handle its submission.

Instructions:

- Serve an HTML form with fields for a name and email.
- Use the POST method to submit the form data to `/submit`.
- Handle the form submission on the server side and respond with a success message including the submitted data.

- ## Implementing Middleware

Assignment:
Create a middleware function for logging requests.

Instructions:

- Create a middleware function that logs the request method and URL for every incoming request.
- Use the middleware in your server to track all requests made to your API.
- Log the request details to the console.

- ## Basic Authentication

Assignment:
Implement basic authentication for a protected route.

Instructions:

- Create a simple login route (`/login`) that accepts a username and password.
- Validate the credentials and respond with a success message or an unauthorized error.
- Protect a route (e.g., `/api/protected`) that requires successful login to access.

- ## Redirecting Requests

Assignment:
Implement request redirection.

Instructions:

- Set up a redirect from `/old-url` to `/new-url`.
- Use the `res.writeHead` method to set the status code to 301 (Moved Permanently) and redirect users to the new URL.

- ## Setting Custom Headers

Assignment:
Set custom HTTP headers in your responses.

Instructions:

- Create a server that responds to requests with custom headers (e.g., `X-Powered-By`, `Content-Type`).
- Log the headers sent in the response to the console.

- **Rate Limiting**

Assignment:
Implement a simple rate-limiting mechanism.

Instructions:

- Create a middleware that limits the number of requests a user can make to the server within a specific time frame (e.g., 5 requests per minute).
- Use an in-memory store to track request counts and reset them after the time window.

- **Setting Up a Proxy**

Assignment:
Set up a simple proxy for an external API.

Instructions:

- Create a route that proxies requests to an external API (e.g., a public API like JSONPlaceholder).
- Use the `http-proxy-middleware` package to handle the proxying.
- Log the requests and responses to the console.

- Express.js Framework

Theory Assignment:

- Explain the advantages of using the Express.js framework over the built-in HTTP module.

Practical Assignment:

- Build a basic Express.js application with at least three routes (e.g., GET, POST, DELETE) and demonstrate how to handle request parameters and query strings.

- **Setting Up a Basic Express Server**

Assignment:
Create a basic Express server.

Instructions:

- Set up an Express project using npm.

- Create a server that listens on a specified port (e.g., 3000).
- Respond with "Welcome to Express!" when accessed at the root route (`/`).
- Log a message to the console indicating the server has started.

• Creating Routes

Assignment:
Implement multiple routes in your Express application.

Instructions:

- Create routes for:
  - `/` - respond with "Home Page"
  - `/about` - respond with "About Page"
  - `/contact` - respond with "Contact Page"
- Log the requested route in the console.

• Using Express Middleware

Assignment:
Implement custom middleware in your Express application.

Instructions:

- Create a middleware function that logs the request method and URL.
- Use this middleware in your Express app to log all incoming requests.
- Ensure the middleware does not block the request from reaching the route handlers.

• Handling Query Parameters

Assignment:
Handle query parameters in an Express route.

Instructions:

- Create a route `/api/search` that accepts a query parameter (e.g., `?q=searchTerm`).
- Respond with a JSON object that includes the search term and a message (e.g., `{"query": "searchTerm", "message": "Search received"}`).

- Log the received query parameter to the console.

• Handling Form Data with POST Requests

Assignment:
Set up a route to handle form submissions.

Instructions:

- Create an HTML form with fields for name and email.
- Set the form method to POST and action to `/submit`.
- Create a POST route `/submit` that processes the form data and responds with a success message.

• Serving Static Files

Assignment:
Serve static files using Express.

Instructions:

- Create a directory called `public` and add HTML, CSS, and JavaScript files.
- Use `express.static` to serve the static files from the `public` directory.
- Access the files via URLs (e.g., `/index.html`).

• Implementing Error Handling Middleware

Assignment:
Create a custom error handling middleware.

Instructions:

- Implement an error handling middleware function that catches errors and sends a JSON response with the error message.
- Ensure the middleware is used after all route handlers.
- Simulate an error in one of your routes and verify the error handling works correctly.

• Using Router for Modular Routing

Assignment:
Organize your routes using Express Router.

Instructions:

- Create a new router for user-related routes (`/api/users`).
- Implement routes to:
  - `GET /api/users` - respond with a list of users.
  - `POST /api/users` - add a new user.
- Import the router into your main Express app and use it.

• Connecting to a MongoDB Database

Assignment:
Connect your Express application to a MongoDB database using Mongoose.

Instructions:

- Install and set up Mongoose in your project.
- Create a User model with fields for name and email.
- Set up a POST route (`/api/users`) to save user data to the MongoDB database.

• Implementing Basic Authentication

Assignment:
Create a simple login system using basic authentication.

Instructions:

- Set up a login route that accepts username and password.
- Validate the credentials (hardcoded for simplicity).
- Respond with a success or failure message based on the validation.

• Implementing JSON Web Tokens (JWT)

Assignment:
Set up JWT authentication for protecting routes.

Instructions:

- Create a registration route that generates a JWT for new users.
- Protect a route (e.g., `/api/protected`) that requires a valid JWT to access.
- Use middleware to verify the JWT on the protected route.

• Using Environment Variables

Assignment:
Utilize environment variables in your Express application.

Instructions:

- Use the `dotenv` package to load environment variables from a `.env` file.
- Store sensitive information like the database connection string in the `.env` file.
- Access the environment variables in your code and connect to the database using these variables.

• Implementing Rate Limiting

Assignment:
Create a rate-limiting middleware.

Instructions:

- Implement a middleware function that limits the number of requests from a single IP address within a time frame (e.g., 100 requests per hour).
- Use an in-memory store or a package like `express-rate-limit` to handle rate limiting.

- ## Sending Emails with Nodemailer

Assignment:
Set up email functionality using Nodemailer.

Instructions:

- Install and configure Nodemailer in your Express application.
- Create a route `/send-email` that sends an email when accessed.
- Customize the email content (subject, body) and log the result of the sending operation.

- ## Implementing CORS

Assignment:
Enable Cross-Origin Resource Sharing (CORS) in your Express application.

Instructions:

- Use the `cors` middleware to allow requests from different origins.
- Configure CORS to only allow specific origins if desired.
- Test your setup by making requests from a different domain.

- ## Implementing File Uploads

Assignment:
Set up file uploads in your Express application.

Instructions:

- Use the `multer` middleware to handle file uploads.
- Create a form for uploading files and set the form's `enctype` to `multipart/form-data`.
- Create a POST route to handle the file upload and respond with the uploaded file's details.

- ## Creating a Simple Todo Application

Assignment:
Build a basic todo application with CRUD operations.

Instructions:

- Set up routes to:
  - `GET /api/todos` - retrieve the list of todos.
  - `POST /api/todos` - create a new todo.
  - `PUT /api/todos/:id` - update a todo by ID.
  - `DELETE /api/todos/:id` - delete a todo by ID.
- Store the todos in an in-memory array or a MongoDB database.

- **Handling Cookies and Sessions**

Assignment:
Implement cookie and session handling in your Express application.

Instructions:

- Use `express-session` to manage user sessions.
- Create a login route that sets a session upon successful login.
- Protect a route that requires the user to be logged in.

- **Setting Up Unit Tests with Mocha and Chai**

Assignment:
Write unit tests for your Express routes.

Instructions:

- Set up a testing environment using Mocha and Chai.
- Write tests for your routes to verify correct responses for various scenarios (e.g., success, error).
- Run the tests and ensure they pass.

- **Deploying Your Express Application**

Assignment:
Deploy your Express application to a cloud service.

Instructions:

- Choose a cloud platform (e.g., Heroku, Vercel, or AWS) to deploy your application.
- Follow the platform's guidelines to deploy your Express app.
- Ensure that the application runs correctly in the production environment.

- Restful APIs

Theory Assignment:

- Define RESTful API principles and discuss their importance in web application development.

Practical Assignment:

- Create a RESTful API for a simple resource (e.g., books). Implement CRUD (Create, Read, Update, Delete) operations using Express.js.

- **Setting Up a Basic RESTful API**

Assignment:

Create a basic RESTful API using Node.js and Express.

Instructions:

- Initialize a new Node.js project with `npm init`.
- Install Express using `npm install express`.
- Create a server that listens on a specified port (e.g., 3000).
- Implement a simple GET endpoint (`/api`) that returns a JSON response (e.g., `{ message: "Welcome to the API!" }`).

• Implementing CRUD Operations

Assignment:
Create a RESTful API for managing a list of items (e.g., tasks, products).

Instructions:

- Set up routes for the following CRUD operations:
  - `GET /api/items` - Retrieve all items.
  - `GET /api/items/:id` - Retrieve a single item by ID.
  - `POST /api/items` - Create a new item.
  - `PUT /api/items/:id` - Update an existing item by ID.
  - `DELETE /api/items/:id` - Delete an item by ID.
- Store items in an in-memory array (or use a simple database).

• Handling Request and Response

Assignment:
Implement detailed request and response handling for your API.

Instructions:

- For the `POST /api/items` route, accept JSON data for creating a new item.
- Validate incoming data and send appropriate responses for success or error (e.g., 400 for bad requests).
- Use the `res.status()` method to set the HTTP status code based on the operation's result.

• Implementing Middleware for Logging

Assignment:
Create middleware to log incoming requests to your API.

Instructions:

- Implement a logging middleware that logs the request method, URL, and timestamp.
- Apply this middleware to all incoming requests.
- Use `console.log` to output the log information to the terminal.

- **Using Query Parameters**

Assignment:
Enhance your API to handle query parameters for filtering.

Instructions:

- Modify the `GET /api/items` route to accept optional query parameters (e.g., `?search=keyword`).
- Filter the list of items based on the search query and return the filtered results.
- Respond with a message if no items match the search criteria.

- **Connecting to a MongoDB Database**

Assignment:
Integrate MongoDB with your RESTful API.

Instructions:

- Install Mongoose using `npm install mongoose`.
- Set up a connection to a MongoDB database.
- Replace the in-memory array with a MongoDB collection to store items.
- Implement the CRUD operations to interact with the MongoDB database.

- **Implementing Error Handling**

Assignment:
Create a centralized error handling mechanism for your API.

Instructions:

- Implement an error handling middleware that catches errors and responds with a JSON error message.
- Ensure that the error handling middleware is the last middleware in your app.
- Simulate an error in one of your routes and verify that the error handling works correctly.

- **Adding Authentication (Basic or JWT)**

Assignment:
Secure your API with basic authentication or JWT (JSON Web Tokens).

Instructions:

- Choose between basic authentication or JWT.
- For JWT:
    - Set up user registration and login routes to issue tokens.
    - Protect certain routes by requiring a valid token.

- For basic authentication:
    - Implement a simple check for username and password in the headers.

• Implementing Pagination

Assignment:
Add pagination to your `GET /api/items` route.

Instructions:

- Allow clients to request specific pages of items by accepting `page` and `limit` query parameters.
- Calculate the appropriate items to return based on the requested page and limit.
- Respond with the paginated list of items and include metadata (e.g., total items, current page).

• Testing Your API with Postman

Assignment:
Use Postman to test your RESTful API.

Instructions:

- Install Postman and create a new collection for your API.
- Create requests for each of the CRUD operations and test their functionality.
- Document the expected responses for each request and check that they match your implementation.

• Implementing CORS

Assignment:
Enable Cross-Origin Resource Sharing (CORS) for your API.

Instructions:

- Install the CORS middleware using `npm install cors`.
- Configure CORS to allow requests from specific origins or all origins.
- Test your API from a frontend application to ensure CORS is working correctly.

• Versioning Your API

Assignment:
Implement versioning for your RESTful API.

Instructions:

- Set up your API to support multiple versions (e.g., `/api/v1/items` and `/api/v2/items`).

- Differentiate the implementations between versions, allowing for changes and improvements in newer versions.
- Test both versions to ensure they function correctly.

- **Implementing Rate Limiting**

Assignment:
Add rate limiting to your RESTful API.

Instructions:

- Use the `express-rate-limit` package to limit the number of requests to your API.
- Configure the rate limiter for specific routes (e.g., limit requests to `/api/items`).
- Test the rate limiting by making repeated requests.

- **Creating a Simple Client to Consume Your API**

Assignment:
Create a simple frontend application to interact with your RESTful API.

Instructions:

- Use HTML and JavaScript (or a library like Axios) to create a basic user interface.
- Implement functions to make API calls to your backend for CRUD operations.
- Display the results (e.g., list of items, responses) in the UI.

- **Deploying Your RESTful API**

Assignment:
Deploy your RESTful API to a cloud service.

Instructions:

- Choose a cloud platform (e.g., Heroku, Vercel, AWS) for deployment.
- Follow the platform's guidelines to deploy your Node.js application.
- Ensure that the API runs correctly in the production environment and can be accessed publicly.

- Asynchronous Programming

Theory Assignment:

- Discuss the importance of asynchronous programming in Node.js. Explain the concepts of callbacks, promises, and async/await.

Practical Assignment:

- Refactor a callback-based code snippet to use promises and then implement async/await for the same functionality.

- **Understanding Callbacks**

Assignment:
Create a simple Node.js application that demonstrates the use of callbacks.

Instructions:

- Write a function `fetchData` that simulates fetching data with a delay (using `setTimeout`).
- Use a callback to return the fetched data after the delay.
- Create another function that calls `fetchData` and logs the result to the console.

Example:

```
javascript
Copy code
function fetchData(callback) {
setTimeout(() => {
const data = { message: "Data fetched successfully!" };
callback(data);
}, 2000);
}
fetchData((result) => {
console.log(result);
});
```

- **Promises**

Assignment:
Convert the previous callback-based implementation to use promises.

Instructions:

- Modify the `fetchData` function to return a promise.
- Resolve the promise with the fetched data after the delay.
- Create a function that calls `fetchData` and uses `.then()` to log the result.

Example:

```
javascript
Copy code
function fetchData() {
return new Promise((resolve) => {
setTimeout(() => {
const data = { message: "Data fetched successfully!" };
resolve(data);
}, 2000);
});
}
fetchData().then((result) => {console.log(result);
});
```

- **Chaining Promises**

Assignment:
Create a sequence of asynchronous operations using promise chaining.

Instructions:

- Create a function `processData` that takes data as input and returns a promise.
- Chain two calls to `fetchData` and `processData` to demonstrate how to work with multiple asynchronous operations.
- Log the final result after both operations are complete.

Example:

```
javascript
Copy code
function processData(data) {
return new Promise((resolve) => {
setTimeout(() => {
const processed = { message: data.message.toUpperCase()
};resolve(processed);
}, 1000);
});
}
fetchData()
.then((result) => processData(result))
.then((processedResult) => {
console.log(processedResult);
});
```

- **Async/Await**

Assignment:
Refactor the promise-based implementation to use async/await syntax.

Instructions:

- Create an `async` function that fetches data and processes it using `await`.
- Use `try/catch` to handle any errors that may occur during the asynchronous operations.
- Log the final result.

Example:

```
javascript
Copy code
async function fetchAndProcessData() {
try {
const result = await fetchData();
const processedResult = await processData(result);
console.log(processedResult);
} catch (error) {
console.error("Error:", error);
}
}
fetchAndProcessData();
```

- **Error Handling in Promises**

Assignment:
Implement error handling in your promise-based functions.

Instructions:

- Modify the `fetchData` function to randomly throw an error (e.g., using `Math.random()`).
- Use `.catch()` to handle errors when calling the function.
- Log the error message to the console.

Example:

```
javascript
Copy code
function fetchData() {
return new Promise((resolve, reject) => {
setTimeout(() => {
if (Math.random() < 0.5) {
const data = { message: "Data fetched successfully!" };resolve(data);
} else {
reject(new Error("Failed to fetch data."));
}
}, 2000);
});
}
fetchData()
.then((result) => console.log(result))
.catch((error) => console.error("Error:", error.message));
```

- **Using `Promise.all()`**

Assignment:
Demonstrate the use of `Promise.all()` to handle multiple asynchronous operations.

Instructions:

- Create multiple functions that return promises (e.g., `fetchData1`, `fetchData2`).
- Use `Promise.all()` to wait for all promises to resolve.
- Log the results of all resolved promises.

Example:

```
javascript
Copy code
function fetchData1() {
return new Promise((resolve) => {
setTimeout(() => resolve("Data 1"), 1000);
});
}
function fetchData2() {
return new Promise((resolve) => {
setTimeout(() => resolve("Data 2"), 1500);
});
}
```

```
Promise.all([fetchData1(), fetchData2()])
.then((results) => {
console.log("All data:", results);
});
```

- **Using `Promise.race()`**

Assignment:
Use `Promise.race()` to handle the first promise that resolves.

Instructions:

- Create two functions that return promises with different timeouts.
- Use `Promise.race()` to get the result of the promise that resolves first.
- Log the result of the winning promise.

Example:

```
javascript
Copy code
function fetchData1() {
return new Promise((resolve) => {
setTimeout(() => resolve("Data 1"), 2000);
});
}
function fetchData2() {
return new Promise((resolve) => {
setTimeout(() => resolve("Data 2"), 1000);
});
}
Promise.race([fetchData1(), fetchData2()])
.then((result) => {
console.log("First resolved:", result);
});
```

- **Handling Asynchronous File Operations**

Assignment:
Demonstrate asynchronous file operations using the `fs` module with promises.

Instructions:

- Use the `fs.promises` API to read and write files.
- Create a function that reads a file, processes its content, and writes the result to a new file.
- Log the success or error message.

Example:

```
javascript
Copy code
const fs = require('fs').promises;
async function readAndWriteFile() {
try {
const data = await fs.readFile('input.txt', 'utf8');
```

```
const processedData = data.toUpperCase();
await fs.writeFile('output.txt', processedData);
console.log("File written successfully!");
} catch (error) {
console.error("Error:", error.message);
}
}
readAndWriteFile();
```

- **Asynchronous HTTP Requests**

Assignment:
Make asynchronous HTTP requests using the `axios` library.

Instructions:

- Install the `axios` library using `npm install axios`.
- Create an async function that fetches data from a public API (e.g., JSONPlaceholder).
- Log the fetched data to the console.

Example:

```
javascript
Copy code
const axios = require('axios');
async function fetchPosts() {
try {
const response = await
axios.get('https://jsonplaceholder.typicode.com/posts');
console.log(response.data);
} catch (error) {
console.error("Error:", error.message);
}
}
fetchPosts();
```

- **Combining Async/Await with Express**

Assignment:
Create an Express application that uses async/await for route handlers.

Instructions:

- Set up an Express server with a route that fetches data from an external API.
- Use async/await in the route handler to manage asynchronous requests.
- Return the fetched data as a JSON response.

Example:

```
javascript
Copy code
const express = require('express');
const axios = require('axios');
const app = express();
app.get('/posts', async (req, res) => {
```

```
try {
const response = await
axios.get('https://jsonplaceholder.typicode.com/posts');
res.json(response.data);
} catch (error) {
res.status(500).send("Error fetching data.");
}
});
app.listen(3000, () => {
console.log("Server is running on port 3000");
});
```

- Database Integration

Theory Assignment:

- Compare SQL and NoSQL databases. Discuss how to connect Node.js applications to databases.

Practical Assignment:

- **Setting Up a Node.js Project**

Assignment:
Create a basic Node.js application.

Instructions:

- Initialize a new Node.js project using `npm init -y`.
- Install the required packages: `express`, `mongoose`, and `nodemon`.
- Create a simple Express server that listens on a specified port.

- **Connecting to MongoDB**

Assignment:
Connect your Node.js application to a MongoDB database.

Instructions:

- Create a MongoDB cluster using MongoDB Atlas.
- Use Mongoose to connect to the MongoDB database in your Node.js application.
- Log a message to the console to confirm the connection.

- **Creating a Model**

Assignment:
Define a Mongoose model for a collection.

Instructions:

- Create a Mongoose schema and model for a simple `User` collection with fields like `name`, `email`, and `password`.
- Export the model for use in other parts of your application.

• Creating and Saving Documents

Assignment:
Implement a route to create a new document in the MongoDB database.

Instructions:

- Create a POST route (`/users`) in your Express app to accept user data.
- Use the Mongoose model to save the data to the database.
- Return a success message or the created user data in the response.

• Retrieving Documents

Assignment:
Implement a route to fetch documents from the MongoDB database.

Instructions:

- Create a GET route (`/users`) to retrieve all users from the database.
- Use the Mongoose model to find all documents and return them as JSON.

• Updating Documents

Assignment:
Implement a route to update an existing document.

Instructions:

- Create a PUT route (`/users/:id`) that accepts user ID as a URL parameter.
- Use Mongoose to find the user by ID and update their information.
- Return the updated user data in the response.

• Deleting Documents

Assignment:
Implement a route to delete a document from the MongoDB database.

Instructions:

- Create a DELETE route (`/users/:id`) that accepts user ID as a URL parameter.
- Use Mongoose to delete the user by ID.
- Return a confirmation message upon successful deletion.

- ## Using Query Parameters

Assignment:
Implement filtering of users using query parameters.

Instructions:

- Modify the GET route (`/users`) to accept optional query parameters (e.g., `name` or `email`).
- Use these parameters to filter the results when retrieving users from the database.

- ## Validating User Input

Assignment:
Add validation for user input before saving to the database.

Instructions:

- Use the `express-validator` package to validate user input in the POST route.
- Return appropriate error messages if validation fails.

- ## Implementing Pagination

Assignment:
Implement pagination for user retrieval.

Instructions:

- Modify the GET route (`/users`) to accept `page` and `limit` query parameters.
- Use Mongoose's `.skip()` and `.limit()` methods to paginate the results.

- ## Setting Up a Simple Authentication System

Assignment:
Create a simple user authentication system.

Instructions:

- Implement a registration route to create new users and hash passwords using `bcrypt`.
- Implement a login route to authenticate users and return a JSON Web Token (JWT).

- ## Protecting Routes with Middleware

Assignment:
Implement middleware to protect certain routes.

Instructions:

- Create a middleware function that checks for a valid JWT in the authorization header.

- Protect the `/users` route so that only authenticated users can access it.

- **Aggregating Data**

Assignment:
Use MongoDB aggregation to perform data analysis.

Instructions:

- Create a route that returns the total number of users and any other relevant statistics (e.g., count by email domain).
- Use Mongoose's `.aggregate()` method to perform the aggregation.

- **Handling Errors**

Assignment:
Implement error handling for your application.

Instructions:

- Create a centralized error-handling middleware in your Express application.
- Use this middleware to handle errors from your routes and return appropriate responses.

- **Deployment**

Assignment:
Deploy your Node.js application with MongoDB to a cloud platform.

Instructions:

- Deploy your application to platforms like Heroku, Vercel, or Render.
- Ensure your MongoDB connection string is set up properly for the deployed environment.

- Error Handling

Theory Assignment:

- Explain the importance of error handling in Node.js applications. Discuss the various error handling techniques.

Practical Assignment:

- Write a Node.js application that includes error handling for asynchronous operations. Use try/catch blocks and promise.catch to manage errors gracefully.

- Testing and Debugging

Theory Assignment:

- Discuss the importance of testing in software development. Describe popular testing frameworks for Node.js.

Practical Assignment:

- Write unit tests for your Express.js application using Mocha and Chai. Test at least two routes of your API to ensure they return the expected results.

- Middleware in Express.js

Theory Assignment:

- Define middleware in the context of Express.js and discuss its role in request handling.

Practical Assignment:

- Create custom middleware for logging request details (method, URL, timestamp) and apply it to your Express.js application.

- Authentication and Security

Theory Assignment:

- Discuss the importance of authentication and security in web applications. Explain common strategies for user authentication in Node.js.

Practical Assignment:

- Implement user authentication in your Express.js application using Passport.js or JWT (JSON Web Tokens). Create registration and login routes.