5 years Integrated M. Sc. (IT)/ M. Sc. (IT)
IT7050-Full Stack Web Development
Practical List

Practical No. 1	**Enrollment No:**
Practical Problem	**Write, test, and run JavaScript code for each task. Submit the code along with screenshots of the output where required. Use meaningful variable names and proper indentation in your code.**

1.	Write HTML & JavaScript code to display “Welcome to Full Stack Batch” in the browser console.
2.	Create a script that uses window.prompt() to take a user’s name and then displays a welcome message using window.alert().
3.	Write a JavaScript program that prompts the user to enter two numbers, then uses arithmetic operators to calculate and display their sum, difference, product, and quotient. Additionally, check whether each number entered is positive, negative, or zero, and display the result.
4.	Develop a JavaScript program to simulate a traffic signal system. Prompt the user to enter a color name (red, yellow, or green), and display the appropriate action: Red → Stop, Yellow → Get Ready, Green → Go. Implement the logic using a switch statement.
5.	Create a script that uses a switch statement to display the name of the day based on the number entered (1 for Monday, etc.).
6.	Create a JavaScript program to evaluate student performance. Prompt the user to enter marks for three subjects, then calculate and display the total and average. Based on the average, assign a grade as follows: • A for average ≥ 80, • B for 60–79, • C for 40–59, and • Fail for average < 40. Implement the grading logic using if...else statements.
7.	Develop a JavaScript program to convert temperatures between Celsius and Fahrenheit. First, prompt the user to choose the conversion type (C→F or F→C). Then, take the temperature value as input and perform the conversion using the formulas: • C→F: (C × 9/5) + 32, • F→C: (F − 32) × 5/9. Finally, display the converted value in the console using console.log().

8.	Write a JavaScript program using a for loop to display numbers from 1 to 10.
9.	Create an array of five car brands. Display the first element, change it to another brand, and display the updated array.
10.	Write JavaScript code to concatenate first and last name variables into a full name and display its length.
11.	Write a JavaScript program to validate a website signup form. Prompt the user to enter a username and password using window.prompt().Verify that the username is at least 5 characters long and the password is at least 8 characters long. If either condition is not satisfied, display an alert message to the user.
12.	Create an object student with properties name, age, and course. Display all properties in the console.
13.	Write a program to loop through all properties of the student object using for...in.
14.	Develop a JavaScript function that prompts the user to enter a string, counts the total number of words in the input using string methods and loops, and then displays the word count in the console.
15.	Create a JavaScript program that generates a random number between 1 and 10 using the Math object. Prompt the user to guess the number, allowing up to 3 attempts. If the guess matches the generated number, display 'You Win!'; otherwise, display 'Try Again!'.
16.	Create a JavaScript arrow function that returns the square of a number given by the user.
17.	Create an HTML page with a button that changes a <p> tag’s text using document.getElementById().innerHTML.
18.	Write JavaScript code to merge two arrays using concat() and display the result.
19.	Write a JavaScript program that removes the last element of an array using pop() and displays the updated array.
20.	Build a JavaScript program to simulate an online shopping cart. Store product prices in an array, calculate the total using a loop, and display it in the console. If the total exceeds ₹2000, show a message 'You are eligible for free delivery!' using window.alert()."
21.	Write a JavaScript program to create a simple To-Do list. Store tasks in an array with at least three initial items. Use push() to add a new task from user input and pop() to remove the last task. Display the updated list in the console.

22.	Develop a JavaScript program to simulate a simple banking application. Create an account object with properties: accountNumber, accountHolderName, and balance. Include functions deposit(amount) and withdraw(amount) to update the balance, ensuring withdrawals are only allowed if sufficient funds are available.
23.	Create a JavaScript program for a movie ticket booking system. Store a list of movie names in an array, prompt the user to select a movie by entering its index, and display a confirmation message: 'You have booked a ticket for [Movie Name]'.
Objective(s)	Objective of this practical is students can be able to apply JS basics (variables, data types, operators, control structures), handle input/validation, use decision-making (if...else, switch), work with loops, arrays, strings, functions, and objects, use built-in objects (Math, console, DOM), and build small real-world apps for problem-solving.
Pre-requisite	Basic HTML/CSS, computer skills, and JavaScript basics.
Duration for completion	8 Hours
Nature of submission	Handwritten
Assessment	
Faculty Signature and Date	

Practical No. 2	**Enrollment No:**
Practical Problem	**Write, test, and run React code for each task. Submit the code along with screenshots of the output wherever required. Use meaningful variable names, proper indentation, JSX syntax, and react best practices in your implementation.**
1.	Develop a React component for a college website that displays a welcome message, such as “Welcome to [College Name] Portal,” utilizing JSX syntax.
2.	Create a news portal banner that dynamically displays headlines, for example, “Breaking News: React 19 Released!”
3.	Develop a reusable Button component that accepts label and color as properties. Implement this component in multiple sections of the application, such as for Submit and Cancel actions.
4.	In a contact card app, display a user’s name and phone number by passing them as props from the parent component.
5.	Develop a reusable ProductCard component that accepts name, price, and description as props and displays them.
6.	Create a ProfileCard component for a social networking site that displays a user's name, age, and bio using props.
7.	Build a Profile component for a social media profile page that takes props (name, age, location) and uses props destructuring to display details.
8.	Create a Counter App with “+” and “–” buttons using useState to manage the count value.
9.	Build a simple Quiz App where “Next” and “Previous” buttons update the current question using state to track the index.
10.	Fetch mock weather data (e.g., city & temperature) using useEffect and display it when the page loads. Refresh the data every 10 seconds to simulate live updates.
11.	Create a Digital Clock using useEffect that updates every second.
12.	Build a Search Suggestion Box where typing in a search field fetches and displays matching mock suggestions using useEffect.
13.	Add a Light/Dark Mode toggle using state to dynamically switch the background color.
14.	Design a VoteCounter component with “Like” and “Dislike” buttons, tracking and displaying counts using useState.
15.	Design a cart system where users add items with prices. Display the total bill dynamically and show “Free Delivery!” if the bill exceeds ₹500.
16.	Develop a dashboard to show the account balance. Add Deposit and Withdraw buttons to update the balance using useState. If balance < ₹500, show “Low Balance Alert!”.

17.	Create a Student Registration Form (fields: Name, Email, Course) as controlled components. Display submitted details on the page.
18.	Build a Login Form that accepts email and password, storing the values in state and logging them in the console on submission.
19.	Develop a Temperature Converter App where entering Celsius automatically updates the Fahrenheit value using onChange events.
20.	Create a To-Do App where users can type a task and click “Add” to append it to the task list.
21.	Implement a Login System that shows “Welcome, User” if isLoggedIn is true, otherwise displays “Please log in”.
22.	Build a Result Portal where the user enters marks. Display “Pass” if marks ≥ 40, otherwise “Fail” using conditional rendering.
23.	Create a shopping cart with multiple product prices (e.g., mobile ₹700, headphones ₹400, charger ₹300). Calculate total value and display “You got a discount!” only if it exceeds ₹1000.
24.	For an online store, display “In Stock” if quantity > 0, otherwise show “Out of Stock” for each product.
25.	In a library system, display available book titles from an array using map() and assign unique keys.
26.	Create a doctor list. When a user clicks Book Appointment, display “Appointment confirmed with Dr. [Doctor Name]”.
27.	Display a list of songs. When a user clicks on a song, highlight it and show “Now Playing: [Song Name]”
28.	Display a list of employees with their IDs in a table format using map().
29.	Render a list of movies with their ratings, highlighting those with ratings above 8 in bold.
30.	Create an autosave draft feature for a blogging site that saves text every 5 seconds using useEffect with setInterval.

31.	Develop a Student Management Dashboard in React for the college portal. The dashboard should enable student registration, maintain a dynamic list, and display student details. Use JSX, props, state, and useEffect to implement the complete project. • **Requirements**: 1. Display a heading: “Welcome to Student Management Dashboard”. 2. Create a StudentCard component that accepts props: name, age, and course. 3. Add a form to register a new student with fields: Name, Age, Course. 4. Use useState to manage form values and handle the Submit button to add new students. 5. Ensure the form works as a controlled component (all inputs stored in state). 6. If no students are registered, display: “No students registered yet”. Otherwise, display the student list. 7. Maintain the student list in state and render it dynamically using map(). 8. Each student entry should be displayed using the StudentCard component. 9. Use useEffect to log “Student list updated” in the console whenever a new student is added. 10. Also, update the browser tab title with the current number of registered students (e.g., “3 Students Registered”).

32.	Develop a small E-Commerce Product Dashboard in React for an online store. The dashboard should display available products, manage a shopping cart, and support product search. Implement the project using JSX, props, state, and useEffect. • Requirements: 1. Display a heading: “Welcome to My Online Store”. 2. Create a reusable ProductCard component that accepts props: name, price, and stock. 3. Use props destructuring to display product details inside ProductCard. 4. Add an “Add to Cart” button in each ProductCard, and maintain a cart counter in state using useState. Increment the count whenever the button is clicked. 5. Create a search bar as a controlled component. Typing in the input should update the searchTerm state. 6. Show “In Stock” if stock > 0; otherwise, display “Out of Stock”. 7. If no products match the search term, display “No products found”. 8. Store product data in an array and render it dynamically using map(). Pass product details as props to ProductCard. 9. Use useEffect to update the document title with the current cart count (e.g., “Cart (2 items)”). 10. Log a console message whenever the cart is updated.
Objective(s)	To do practice of React fundamentals (JSX, props, state, useEffect) by building reusable components, handling events, managing forms, applying conditional rendering, and developing small real-world applications like portals, dashboards, and e-commerce systems.
Pre-requisite	Basic HTML/CSS, computer skills, JavaScript basics and Familiarity with React basics: JSX, components, props, state, and useEffect.
Duration for completion	10 Hours
Nature of submission	Printout
Assessment	
Faculty Signature and Date	

Practical No. 3	**Enrollment No:**
Practical Problem	**Write, test, and run Node.js programs for each of the given scenario-based tasks. Submit the complete source code along with screenshots of execution/output.**
1.	The IT department of your college wants to develop a utility that can automatically check and display system details across all computer labs. To achieve this, write a Node.js program using the os module that prints the system's hostname, total memory, free memory, and operating system type.
2.	A teacher wishes to maintain student attendance in digital format. Write a Node.js program that creates a file named attendance.txt, appends new student names to the file, reads and displays its contents, and finally deletes the file at the end of the semester using the fs module.
3.	An online examination system requires a basic server for handling student requests. Develop a Node.js program using the http module that responds with a welcome message on the home route /, returns a JSON list of subjects at /subjects, and displays a "404 Not Found" error for any invalid routes.
4.	A food delivery application needs a backend to manage customer orders. Write a Node.js program using the http module that handles a GET request on /orders to return a JSON list of orders and a POST request on /orders to accept new order details from the client.
5.	A college fest website requires a simple backend system. Using Express.js, create an application that displays a welcome message on /, returns a JSON list of events on /events, and allows students to register for the fest through a POST request at /register.
6.	A smart home system generates events for controlling devices such as turning lights on, switching fans on, and locking doors. Write a Node.js program using the events module to create and trigger these events, where listeners display the corresponding action messages.
7.	A digital library catalog must provide book details to students. Create a Node.js server using the http module that returns a JSON array of books when /books is requested, displays details of a single book when /book/:id is requested, and shows an error message if the book is not available.
8.	A banking system wants to log all customer transactions for security and auditing purposes. Write a Node.js program that records transaction details (userId, amount, date) in a log file, allows reading and displaying of previous transactions, and deletes the log file automatically if its size exceeds a certain limit.
9.	A college admission portal requires a backend system to store student registration details. Develop an Express.js application that accepts student data (name, course, email) via a POST request on /register, stores the data in a local file named

	students.json, and responds with a success message along with the registered details.
10.	A photo gallery website should allow users to upload their pictures. Write a Node.js application using Express.js that accepts image uploads, stores them in an uploads/ directory, and returns the file path of the uploaded image as the response.
11.	A weather application must provide temperature details for various cities. Develop a Node.js server that stores city-wise temperatures in memory and responds with the temperature when a valid city name is requested via /weather?city=CityName. If the city is not found, the server should return an appropriate error message.
12.	A news portal wants to send instant updates to subscribers whenever breaking news or sports updates are available. Using the events module, create a Node.js program where events like "breakingNews" and "sportsUpdate" are triggered, and multiple listeners (subscribers) receive and display the updates.
13.	A task management application requires a backend API for maintaining to-do lists. Using Express.js, create routes to perform CRUD operations: GET /tasks to list tasks, POST /tasks to add a new task, PUT /tasks/:id to update an existing task, and DELETE /tasks/:id to remove a task.
14.	An online voting system must record and display poll results. Write an Express.js application where POST /vote accepts votes for candidates and GET /results displays the current vote counts in JSON format.

Objective(s)	Objective of this practical list is to help students gain hands-on experience with Node.js as a server-side JavaScript environment. Students will learn to use core modules, perform file system operations, build servers with the http module and Express.js, and handle HTTP requests and responses. By working on real-world scenarios, they will develop the skills to create efficient, scalable, and event-driven applications while following proper coding practices.
Pre-requisite	Basic knowledge of JavaScript, web concepts, and command-line usage. They must also install Node.js with npm and use a code editor like VS Code.
Duration for completion	8 Hours
Nature of submission	Handwritten
Assessment	
Faculty Signature and Date	

Practical No. 4	**Enrollment No:**
Practical Problem	**Write, test, and execute Express.js programs for each given scenario. Submit the complete source code along with screenshots of the output.**
1.	Build a simple blogging platform using Express.js. You need to create routes to add a new blog post, view all blog posts, view a single blog post, update a blog post, and delete a blog post. To keep your code organized and easy to manage, use express.Router() to place all the blog routes in a separate file called blogRoutes.js and connect it to your main app.js
2.	Create middleware in Express.js to handle incoming requests. First, every request should be logged, and then the user should be verified for authentication before reaching the route handler.
3.	You are tasked with creating a student registration system where students fill in their details through a web form. Write a code for form submission in Express.js using the express.urlencoded() middleware.
4.	Suppose you are building an e-commerce app using Express.js. Create a route /search that uses query parameters to find products, for example /search?query=phone&category=electronics, and a route/products/:id to show details of a specific product.
5.	You are creating a portfolio website using Express.js. Your project has HTML, CSS, and image files stored in a public folder. Configure your Express.js application to serve these static files so that users can access them directly through their browsers.
6.	While developing an application, reading a file may sometimes fail if the file does not exist. Create custom error-handling middleware in Express.js to handle such file errors. Additionally, configure your application to handle 404 “route not found” errors so that users see a proper response when they visit an undefined route.
7.	You are developing an online quiz application where only registered users can access quizzes. Implement authentication using JSON Web Tokens (JWT). Build the login functionality to generate a token for the user and secure protected routes so that they can only be accessed with a valid token.
8.	You are building a financial web application using Express.js that must be secure. Implement measures to protect the application from common security threats like SQL Injection, Cross-Site Request Forgery (CSRF), and brute-force login attempts. Use appropriate middleware or tools in Express.js to prevent these security risks.

Objective(s)	The objective of this practical list is to help students gain hands-on experience with Express.js as a web application framework for Node.js. Students will learn to create servers, handle routing, manage middleware, and process HTTP requests and responses. By working on scenario-based tasks, they will develop the skills to build efficient, modular, and scalable web applications while following proper coding practices.
Pre-requisite	Basic knowledge of JavaScript, web concepts, and command-line usage. They must also install Node.js with npm and use a code editor like VS Code.
Duration for completion	8 Hours
Nature of submission	Handwritten
Assessment	
Faculty Signature and Date	

Practical No. 5	**Enrollment No:**
Practical Problem	**Write, test, and run Express.js programs that connect to MongoDB for each given scenario. Submit the complete source code along with screenshots showing the database operations and output.**
1.	While building a blogging application, users should be able to add new blog posts to the database. Sometimes, they might forget to include the title. Create a Mongoose model with validation to ensure that each blog post has both title and content. Additionally, configure your application to handle invalid submissions by returning a proper JSON error response.
2.	In a task management system, implement a Mongoose query to fetch and display all tasks where the status field is set to "completed".
3.	You are working on a Task Management System where users can create tasks with different statuses, such as "completed", "in-progress", and "pending". You need to implement a Mongoose query to fetch and display all tasks that have the status field set to "completed".
4.	You are building an admin panel where admins can delete multiple inactive users from the database in one go. Write the Mongoose query to remove all documents with isActive: false and send a success response with the number of deleted documents.
5.	While designing a blogging platform, each blog post should store its comments. Create a Mongoose schema that embeds comments inside the post document. Each comment should have author and message.
6.	In a student management application, each student can enroll in multiple courses. Design a normalized schema where student documents reference course documents.
7.	You're building a social media app where users can like posts and interact with each other's profiles. Your task is to design the database schema using Mongoose (a MongoDB ODM for Node.js) with a hybrid data model, where: 1. Likes are stored directly in the post document (embedded data). 2. User profiles are stored as references in the database, linking users to the posts they like (referenced data).
8.	While developing a Node.js application, connect it to a local MongoDB instance using Mongoose. Write the code to establish the connection and log " Connected to MongoDB" on success, or an error message otherwise.
9.	While developing a school app, ensure that students younger than 15 cannot be saved to the database. Implement this using Mongoose schema validation.
10.	You are building a user registration API for a web application. When a new user tries to register, you need to ensure that the email address they provide is unique. Implement the registration logic so that if a user attempts to register with an email that already exists in the database, the API should respond with a JSON error

	message.
11.	You are building a simple user management system using Express.js and MongoDB. Perform the following tasks: • Connect to MongoDB using Mongoose in your Express.js application. • Create a User model with fields: name, email, and password. • Implement routes to: o Add a new user o View all users o View a user by ID o Update a user by ID o Delete a user by ID • Test all routes using Postman or any API client.
Objective(s)	Objective of this practical is to help students gain hands-on experience with connecting Express.js applications to MongoDB. Students will learn to define schemas and models using Mongoose, perform CRUD operations on a database, and handle data through Express.js routes. By working on scenario-based tasks, they will develop the skills to build efficient, data-driven, and scalable web applications while following proper coding and database interaction practices.
Pre-requisite	Basic knowledge of JavaScript, web concepts, and command-line usage. They must also install Node.js with npm and use a code editor like VS Code.
Duration for completion	8 Hours
Nature of submission	Handwritten
Assessment	
Faculty Signature and Date	