ABOUT THE COURSE:

TOTAL DURATION:	45 HRS
MODE OF DELIVERY	Virtual Instructor led by Industry Experts + Physical Session conducted by FDP faculty
TRAINER TO STUDENT RATIO:	1:50
TOTAL MARKS:	75

TABLE 1	
OVERALL COURSE OBJECTIVE:	• Analyze Web Development Architecture Critically examine how frontend and backend systems communicate within the client-server model. Deconstruct the roles and responsibilities of each layer in a full-stack environment. • Design and Construct Structured Web Pages Using HTML & CSS Apply advanced HTML and CSS techniques to create semantically structured and visually cohesive web interfaces, evaluating design decisions for optimal user experience. • Develop and Evaluate JavaScript Solutions Create dynamic functionality by implementing complex JavaScript logic, including asynchronous programming. Assess code performance and refactor for maintainability and efficiency. • Design and Optimize Responsive Interfaces Create responsive layouts adaptable to various devices and screen sizes. Evaluate and refine designs to ensure usability and performance across platforms. • Build and Refine Interactive UIs with React.js Construct modular, component-based applications using React.js. Analyze the flow of data through props and state to ensure scalability and responsiveness. • Implement and Evaluate Web Accessibility Practices Design interfaces that adhere to accessibility standards. Critically assess user interactions and modify code to

	improve inclusivity and compliance with ally guidelines. • Integrate and Test Full Stack Functionality Design and implement seamless integration between frontend and backend components. Evaluate data flow and system interactions to ensure functional, reliable application behaviour.

LEARNING OUTCOME:	Design and Develop Responsive User Interfaces Using HTML/CSS Apply and evaluate foundational and advanced HTML/CSS techniques—including Flexbox and Grid—to construct structured, accessible, and responsive web layouts. Analyze and manipulate the Document Object Model (DOM) to enhance user interaction and interface behavior. Create and Optimize Interactive Functionality Using JavaScript Construct dynamic and event-driven web applications by applying core JavaScript concepts, including ES6+ features and asynchronous programming patterns. Analyze user interactions to implement responsive behavior, and evaluate code efficiency in real-world scenarios. Build, Analyze, and Enhance Modular UI Components in React Develop reusable and maintainable components using JSX. Evaluate and implement state management strategies, including React hooks and lifecycle methods, to optimize performance

	and maintain clarity in data flow through props and conditional rendering. Create responsive, event-driven form handling and enhance user experience through structured component architecture.

TABLE 2: MODULE-WISE COURSE CONTENT AND OUTCOME				
SL.NO	MODULE NAME	MODULE CONTENT	MODULE LEARNING OUTCOME	DURATION (HRS)
1	Introduction to Frontend Development and Basic Web Technologies	Intro to Web Development What is Frontend? Roles & Responsibility of Frontend Developer Basics Web Technologies Introduction to HTML Basic HTML structure Introduction to CSS Basic CSS syntax Basic elements, DOM-create/delete elements. Selectors. Advanced CSS techniques like flexbox and grid Best practices for HTML and CSS development	Analyze the Core Concepts of Web Development Architecture Evaluate the roles and interactions between frontend and backend systems within web development. Differentiate their functionalities by deconstructing real-world examples of client-server communication Apply and Integrate Essential Web Technologies Select, apply, and justify the use of core web technologies in frontend development scenarios. Synthesize knowledge of these technologies to construct basic	3

			yet functional user interfaces. Design and Structure Web Pages Using Semantic HTML Create well-structured web pages by implementing HTML elements purposefully. Analyze content hierarchy and utilize appropriate tags such as headings, links, lists, and images to support usability and accessibility. Construct and Refine Web Layouts Using CSS Implement CSS selectors strategically to apply visual styles. Analyze and construct responsive layouts using Flexbox and Grid systems. Evaluate layout designs for adaptability across different devices and screen sizes. Evaluate and Apply Best Practices in Code Quality Assess and apply industry standards for	

			writing clean, maintainable, and semantic HTML and CSS. Continuously refine code for readability, scalability, and performance.	
2	JavaScript & ES6 Essentials	Introduction to Javascript Variables, datatypes, and operators Control flow statements (if-else, for, while, switch) Introduction to ES6 (let, const, template strings) Arrow function, Spread operator, destructing,Callback, Promise. JavaScript fundamentals: functions, objects, arrays Manipulating the DOM with JavaScript Handling events and user interactions with JavaScript	Design and Implement Dynamic Logic Using JavaScript Fundamentals Construct decision-based programs by analyzing data types, control flow logic (e.g., if-else, switch, loops), and variable scope using let and const. Evaluate different approaches to program structure to ensure clarity and efficiency. Develop Modular and Maintainable Code Using Advanced JavaScript Features Create concise functions using arrow syntax. Apply destructuring and the spread operator to streamline array and object manipulation. Evaluate and refactor code for readability and reusability using modern ES6+ features such as	4

			template literals and function closures. Apply Functional Programming Techniques for Data Handling Use higher-order functions like map, filter, and reduce to transform and analyze datasets effectively. Create reusable utility functions that simplify complex data operations. Create and Manage Asynchronous Workflows Construct and evaluate asynchronous JavaScript operations using callbacks and Promises. Design clean, responsive interactions such as data fetching and delayed execution through proper use of asynchronous control structures. Construct and Modify the DOM Programmatically Develop interactive user experiences by dynamically selecting, creating, and removing DOM elements. Apply event handling	

			techniques to manage user input and interaction, including the use of event propagation and delegation for complex UI behavior. Evaluate User Interaction Flow and Optimize DOM Manipulation Assess how user actions affect the DOM and implement efficient patterns to handle events and updates. Optimize performance by minimizing reflows and maximizing code efficiency in response-driven interfaces.	
3	React Fundamentals	Introduction to React - Basics, component based Architecture, virtual DOM Setting up the development environment - Node & npm installation, code editor installation & configuartion, Development environment Create React App (using npm CLI) - Understand Project structure and file organization Components: Functional and class components.	Explain the basics of React, including component-based architecture and the virtual DOM. Describe how React differs from other JavaScript frameworks. Set Up a React Development Environment: Install and configure Node.js and npm. Set up a code editor (e.g., VS Code) with	6

		Props: Passing data to components. State: Managing state within components. Lifecycle Methods: Component lifecycle in class components (componentDidMount, componentDidUpdate, etc.). Event Handling: Handling user inputs and events. Conditional Rendering: Rendering elements based on conditions. Lists and Keys: Rendering lists and understanding the importance of keys. Forms: Controlled vs. uncontrolled components.	necessary extensions and configurations for React development. Create and Navigate a React Project: Use Create React App to initialize a new React project. Understand the project structure and organize files appropriately within a React application. Work with React Components: Create both functional and class components. Explain the differences between functional and class components and when to use each. Pass data between components using props. Understand the concept of props and how to use them to make components reusable and dynamic. Use the useState hook to manage state in functional	

			components. Manage state within class components and understand the differences between state and props. Use lifecycle methods in class components. Implement event handling in React to respond to user inputs such as clicks, form submissions, and keyboard events. Understand the concept of synthetic events in React. Render elements conditionally based on component state or props. Use logical operators and ternary expressions to control what is rendered. Render lists of data efficiently using the map function. Understand the importance of keys in React for maintaining component identity and optimizing rendering performance.	

			Handle form inputs using controlled components to manage form state explicitly. Understand the difference between controlled and uncontrolled components and when to use each.	
4	React Hooks & Routing	Hooks useState useEffect useRef Custom Hooks React Router Setting up React Router Route and Link components Nested Routes Route parameters and query strings Programmatic navigation	Design and Manage State Logic Using React Hooks Construct dynamic and stateful user interfaces by implementing useState and evaluating different strategies for managing component state. Analyze component behavior with useEffect to coordinate side effects such as data fetching and subscriptions, while ensuring proper resource cleanup. Develop Efficient Component Interactions with Advanced Hooks Apply useRef to interact directly with DOM	4

			elements and preserve mutable values across renders. Create and abstract reusable logic through custom hooks to enhance maintainability and consistency across components. Architect and Evaluate Routing Strategies in React Applications Configure client-side routing with React Router to enable seamless navigation. Analyze and implement various route structures using <Route> and <Link> components for intuitive user flows. Implement and Organize Complex Routing Structures Design nested and dynamic routes to handle multi-level navigation. Evaluate the use of route parameters	

			and query strings to pass and retrieve dynamic data, ensuring flexibility in component rendering and user navigation paths. Control Navigation Flow Programmatically Apply programmatic navigation techniques using React Router's navigation utilities (e.g., useNavigate or history object) to direct user flow based on logic, conditions, or application state changes.	
5	Integrating APIs and Backend Communication	Fetching data with Fetch API and Axios CRUD operations Handling API responses and errors Using Async/Await in React Authentication and Authorization	Fetch API: Use the Fetch API to make HTTP requests. Axios: Set up and use Axios for more streamlined and advanced HTTP requests. Perform CRUD Operations Implement Create, Read, Update, and Delete (CRUD) operations in a React application.	3

			Integrate CRUD operations with RESTful APIs to interact with backend services. Handle API Responses and Errors Process and utilize data from API responses to update the UI. Implement error handling strategies for HTTP requests to manage errors and provide feedback to users. Use Async/Await in React Understand and apply async and await syntax for managing asynchronous operations in React. Implement Authentication and Authorization Authentication: Set up user authentication in a React application, including login and logout functionality. Authorization: Implement authorization to restrict access to certain parts	

			of the application based on user roles or permissions.	

TABLE 3: OVERALL COURSE LEARNING OUTCOME ASSESSMENT CRITERIA AND USECASES		
LEARNING OUTCOME	ASSESSMENT CRITERIA	USE-CASES
Analyze and Construct Responsive Frontend Interfaces using HTML, CSS, and DOM Manipulation	Frontend and Backend: Assessing the ability to differentiate between frontend and backend development, and describe the responsibilities associated with each role. Proficiency in HTML/CSS and DOM Manipulation: Evaluating proficiency in HTML/CSS fundamentals, including creating/deleting DOM elements, and applying advanced CSS techniques like flexbox and grid layouts.	Use Case 1: Personal Blog Website. Scenario: Sanjana is passionate about cooking and wants to share her recipes, cooking tips, and culinary adventures with the world. She envisions a personal blog website where she can showcase her content in an organized and visually appealing manner. She also aims to enhance her digital presence through the website. Task: Design the website layout using HTML and CSS to ensure responsiveness across various devices and screen sizes. Apply responsive design principles such as fluid grids, flexible images, and media queries to adapt the layout dynamically. Create a visually appealing design by incorporating custom fonts, colors, and graphics that reflect Emily's culinary theme. Utilize CSS styling techniques to enhance the aesthetics of the website, including typography, spacing, and transitions. Use Case 2: Online Portfolio for a Freelance

		Graphic Designer. Scenario: Gopal is a freelance graphic designer looking to establish a strong online presence and attract potential clients. He wants to showcase his portfolio of design projects, including logos, branding materials, and website designs, in a professional and visually compelling manner. Task: Develop a responsive online portfolio website using HTML and CSS to effectively showcase Gopal's design work across various devices and screen sizes. Implement a clean and modern layout that emphasizes visual elements such as images, graphics, and interactive design components. Utilize CSS techniques to create polished animations, transitions, and hover effects that enhance the user experience and engage visitors.
Design and Evaluate Interactive JavaScript Logic and DOM Operations	Demonstrate proficiency in acquiring essential knowledge for ES6 like arrow functions, spread operator, rest operator,etc. Acquire knowledge of promises which will help in understanding asynchronous programming. And also get to know about Document Object Model (DOM).	Use Case 1: Dynamic Event Booking Website. Scenario: Shyam is an event organizer planning a series of workshops and conferences. He wants to create an interactive website where attendees can view upcoming events, register for tickets, and receive event updates in real-time. He aims to build a user-friendly platform that dynamically updates event information, handles user registrations, and provides a seamless booking experience for

		attendees. Task: Develop a dynamic event booking website using JavaScript DOM manipulation to enhance interactivity and functionality. Design a responsive and visually appealing layout that displays upcoming events, event details, and registration forms. Utilize JavaScript to manipulate the DOM elements dynamically, updating event information and user interface elements in response to user actions. Utilize DOM manipulation techniques to dynamically add, remove, or modify HTML elements based on user input or server responses. Integrate form validation using JavaScript to ensure that user input is accurate and complete before submitting registration details. Implement asynchronous requests using AJAX to communicate with the server, fetch event data, and handle registration submissions without reloading the entire page. Use Case 2: Interactive Task Management Application. Scenario: Jessica is a project manager overseeing multiple teams and tasks. She needs a centralized platform to manage project workflows, assign tasks to team members, and track progress in real-time. Jessica envisions an interactive task management application that allows users to create tasks, set

		deadlines, assign priorities, and collaborate with team members seamlessly. Task: Develop an interactive task management application using JavaScript DOM manipulation to facilitate efficient task tracking and collaboration among team members. Design a responsive and intuitive user interface that enables users to create, edit, and delete tasks dynamically. Utilize JavaScript to manipulate the DOM elements in real-time, updating task lists, statuses, and details based on user interactions and server responses.
Develop and Optimize Modular Components with React's Architecture and Lifecycle	Demonstration of React Concepts: Assess the implementation of JSX syntax, component creation, data handling with props, state management, form usage, and understanding of the React lifecycle. Implement event handlers for common events (e.g., onClick, onChange) in React components and create forms that capture user input and update the state accordingly. Also implement conditional rendering.	Use Case 1: Random Quote Generator Scenario: Emma, an enthusiast of motivational quotes, discovers a Random Quote Display project online but finds its lack of interactive features frustrating. Despite her interest, she's unable to easily refresh the page or generate new quotes, hindering her browsing experience. Emma desires seamless interaction, envisioning clickable buttons or swipe gestures for effortless navigation. Feeling dissatisfied, she considers providing feedback and ultimately seeks alternatives that prioritize user engagement. Task: Develop the pagination functionality to display a limited number of users per page, ensuring optimal performance and user experience. Develop

		functionality to enable users to refresh the page or generate a new random quote with ease. Create buttons or controls within the application interface to trigger the actions of refreshing the page or fetching a new random quote from the external API. Improve the integration with the external API to fetch random quotes efficiently and reliably. Implement error handling mechanisms to gracefully handle API request failures and provide feedback to users. Enhance the visual presentation of quotes within the application interface to ensure clear and appealing display to users. Utilize Bootstrap or similar styling frameworks to optimize the layout and design for improved readability and aesthetics. Use Case 2: Digital timer Scenario: Imagine you are tasked with developing a productivity application that incorporates the Pomodoro Technique—a time management method that uses a timer to break work into intervals, traditionally 25 minutes in length, separated by short breaks. The application should allow users to track their work sessions, take breaks, and customize timer intervals. Task: Implement a digital timer component. Add functionality to start, pause, and reset the timer. Enable users to set

		custom time limits for the timer. Display the timer in a visually appealing format. Ensure accurate tracking of time intervals.
Construct Advanced Functional Interfaces using React Hooks and Client-Side Routing	Proficiency with React Hooks: Evaluate the adeptness in utilizing React hooks, encompassing both built-in and custom hooks, for effective state management and performance optimization in React applications. Set up React Router correctly, define routes using Route and Link components, implement nested routes, handle route parameters and query strings, and perform programmatic navigation. Practical projects and coding exercises will evaluate their understanding and application of these concepts, ensuring they can build dynamic, navigable React applications.	Use Case 1: Tic Tac Toe Game Scenario: Imagine yourself sitting across from a friend, each poised with anticipation as you gaze upon the grid before you. The Tic Tac Toe board, a canvas of possibilities, awaits your strategic moves. With each turn, the tension mounts as you strive to outmaneuver your opponent, placing Xs and Os in a bid to claim victory. The challenge is simple yet exhilarating: three in a row, horizontally, vertically, or diagonally, and the glory is yours. Will you emerge triumphant, or will your adversary outwit you in this timeless battle of wit and tactics? It's time to find out as you embark on a thrilling journey into the world of Tic Tac Toe. Task: Create App Function Component: Define the main functional component named App. useState Hooks: Utilize useState Hook to manage state for board, currentPlayer, winner, and confetti. Handle Cell Click Function: Implement a function handleCellClick to handle cell clicks on the board and update the game state accordingly. Check Winner Function: Define a function checkWinner to determine if there's a winning player based on the current

		board state. Reset Game Function: Create a handleReset function to reset the game state to its initial values. Render Cell Function: Implement a renderCell function to render individual cells on the game board. Render Board Function: Create a renderBoard function to render the entire Tic-Tac-Toe board using the renderCell function. Use Case 2: User Dashboard Scenario: During Sarah's interaction with the SocialConnect platform, the access control component encounters challenges, hindering the seamless authentication and redirection process. The access control component struggles to accurately manage and update the authentication status, leading to inconsistencies in determining whether the user is authenticated or not. Sarah experiences issues with the redirection mechanism, where she may encounter unexpected redirects or errors when attempting to access authenticated content. Task: Create App Function Component: Define the main functional component named App. useState Hooks: Utilize useState Hook to manage state for board, currentPlayer, winner, and confetti. Handle Cell Click Function: Implement a

		function handleCellClick to handle cell clicks on the board and update the game state accordingly. Check Winner Function: Define a function checkWinner to determine if there's a winning player based on the current board state. Reset Game Function: Create a handle Reset function to reset the game state to its initial values. Render Cell Function: Implement a render Cell function to render individual cells on the game board. Render Board Function: Create a render Board function to render the entire Tic-Tac-Toe board using the render Cell function.
Create Secure, Data-Driven React Applications with Full CRUD Operations and Authentication	Implement data fetching in React applications using both Fetch API and Axios. Demonstrate proficiency in performing CRUD operations (Create, Read, Update, Delete) with a clear understanding of how to interact with RESTful APIs. Show competence in handling API responses and errors, implementing appropriate error handling and user feedback mechanisms. The ability to use async and await for managing asynchronous operations should be evident in their code, ensuring readability and maintainability. For authentication and authorization, implement secure user authentication processes, manage tokens, and protect routes based on user roles or permissions.	Use Case 1: Github Profile Viewer in React js Scenario: Imagine you're a developer building a portfolio website. You want to include a feature that allows visitors to view your GitHub profile directly on your site. You decide to create a GitHub Profile Viewer component using React. Visitors can enter your GitHub username, and the component will fetch and display your profile information, including your name, avatar, bio, followers, following, and public repositories. If there's an error, such as an incorrect username, the component will gracefully handle it and display an error message. This feature adds interactivity to your portfolio and showcases your GitHub activity to potential employers or

		collaborators. Task: Input field for entering a GitHub username. Submit button to fetch and display the user's GitHub profile information. Error handling for cases where the username is not found or there is an issue with the API request. Display of user's name, avatar, bio, followers, following, and public repositories. Use Case 2: Weather app Scenario: Emma, a traveler, relies on the Weather Information Application to plan her outdoor activities during her vacation. She inputs her destination city to check the weather forecast, ensuring a pleasant and enjoyable trip without unexpected weather disruptions. Task: Obtain an API key from OpenWeatherMap for accessing weather data. Design the HTML structure for the weather application, including input fields and display areas. Write JavaScript code to fetch weather data from the OpenWeatherMap API based on user input. Handle API responses and update the UI with the retrieved weather information. Implement error handling to manage cases where the city entered by the user is not found. Style the application using CSS to enhance the

		user experience and visual appeal.

TABLE 4: LIST OF FINAL PROJECTS (10 PROJECTS THAT COMPREHENSIVELY COVER ALL THE LEARNING OUTCOME)	
SL.NO	FINAL PROJECT
1	Personal Blog Website
2	Online Portfolio for a Freelance Graphic Designer.
3	Dynamic Event Booking Website.
4	Interactive Task Management Application.
5	Random Quote Generator
6	Digital Timer
7	Tic Tac Toe Game
8	User Dashboard
9	Github Profile Viewer in React JS
10	Weather App

TABLE 5: COURSE ASSESSMENT RUBRICS (TOTAL MARKS: 75)				
ASSESSMENT CRITERIA	DESCRIBE THE CRITERIA OF THE BELOW CATEGORY PERFORMANCE			TOTAL MARKS
	FAIR	GOOD	EXCELLENT	
Problem Definition & Design Thinking	3	5	8	10
Innovation & Problem Solving	1	2	4	5
Implementation of Project	6	12	18	20
Performance of the Project	1	2	4	5

Project Demonstration & Documentation	3	5	8	10
MCQ-based assessment 25 Questions				25