HTML

- **Understanding Full Stack Web Development**
- **Web Application**
- **Full Stack Engineer / Full Stack Developer**
- **Types of Applications**
    - Standalone Application
    - Web Application
    - Mobile application
- **High Level Architecture of Web Application**
    - Frontend Development
    - Backend Development
    - Database
- **Technologies in Web Development**
    - Html
    - Css
    - Javascript
    - Bootstrap
    - React JS
    - Node Js
    - Express JS
    - MongoDB
    - AWS Basics
    - Deployment on Vercel
    - Git and Github
    - Soft Skills
    - Aptitude Training
    - Mock Interview
    - Test
    - Assignment
    - Practice Questions

- **Understanding MERN Stack**
- **Installation of VS Code**
- **Installation of Extension**
- **Introduction to HTML**
- **Purpose of HTML**
- **Elements in HTML**
- **Tags**
- **Types of Tags**
    - Paired Tag
        - Opening Tag
        - Closing Tag
    - Self Closing Tag
- **Structure of the Tag / Representation of the Tag**
- **Button Element**
    - Definition
    - Syntax
    - Tag Name
    - Type of Tag
    - Purpose of the element
- **Heading Element**
    - Definition
    - Syntax
    - Tag Name
    - Type of Tag
    - Purpose of the element
- **Paragraph element**
    - Definition
    - Syntax
    - Tag Name
    - Type of Tag
    - Purpose of the element

- **Structure of HTML Document**
    - Document Declaration
    - Document Scope

- **List Element**
    - Ordered List
        - List Item
            1. Definition
            2. Syntax
            3. Tag Name
            4. Type of Tag
            5. Purpose of the element
    - Unordered List
        - List Item
            1. Definition
            2. Syntax
            3. Tag Name
            4. Type of Tag
            5. Purpose of the element
    - Description List
        - Data Term
            1. Definition
            2. Syntax
            3. Tag Name
            4. Type of Tag
            5. Purpose of the element
        - Data Definition
            1. Definition
            2. Syntax
            3. Tag Name

4. Type of Tag
5. Purpose of the element

- **Attributes in HTML**
  - Purpose
  - Syntax
  - rules
  - Example
- **Attributes of Ordered element**
  - Type
  - Start
  - Reversed
- **Attributes of Unordered Element**
  - Type
- **Markers of Ordered and Unordered List**
  - List of Marker
  - Changing the Marker
- **Image Element**
  - Purpose
  - Syntax
  - Example
- **Attributes of Image Element**
  - Width
  - Height
  - Title
  - Alt
  - Src
  - Id
  - Class
  - Name
- **Division Element**
- **Table Element**
  - Table Header

    - Table Body
    - Table Footer
    - Table Head
    - Table Row
    - Table Cell
    - Caption
- **Attributes of Table Element**
    - Frame
    - Rules
    - Width
    - Height
    - Align
    - Valign
    - Colspan
- **Anchor Element**
    - Creating Link
    - Inter navigation
    - Intra Navigation
- **Forms Element**
    - Input element
    - Select
    - Textarea
    - Select
    - Label
    - Optgroup
    - Option
    - Data List
- **Attributes of Input Element**
    - Type
    - Placeholder
    - Required
    - Name

    - Value
    - minLength
    - maxLength
    - Multiple
    - Checked
    - Disabled
    - Readonly
    - Pattern
    - Size
- **Media Elements**
    - Audio
    - Video
    - Source
- **Semantic Elements**
    - Header
    - Main
    - Footer
    - Nav
    - Menu
    - Section
    - Aside
    - Details
    - Summary

CSS

- **Introduction to CSS**
    - **What is CSS?**
        - Definition and purpose
        - Brief history and evolution

    - **Why Learn CSS?**
        - Importance in web design and development
        - Enhancing user experience
    - **How CSS Works**
        - Role of the browser
        - CSS rendering process
- **Purpose of CSS**
    - **Separation of Content and Design**
        - HTML for structure, CSS for style

      Advantages of Using CSS

        - Reusability and consistency
        - Better accessibility and performance

      CSS in Modern Web Development

        - Integration with JavaScript frameworks
        - Responsive and adaptive design

Properties

- **Definition of CSS Properties**
    - Basic syntax: property: value;
- **Categories of Properties**
    - Text-related properties (e.g., color, font-size, text-align)
    - Box model properties (e.g., margin, padding, border, width, height)
    - Background and border properties (e.g., background-color, border-radius)

    - Display and positioning properties (e.g., display, position, z-index)
- **Commonly Used Properties**
    - Examples and practical use cases

Format of Properties

- **Structure of a CSS Declaration**
    - Selector, property, value
- **CSS Syntax Rules**
    - Importance of semicolons and curly braces
- **Best Practices**
    - Readability and maintainability
    - Using comments and indentation

Approach in CSS

- **Inline CSS**
    - Syntax and use cases
    - Pros and cons
- **Internal CSS**
    - <style> tag in HTML
    - Scope and applications
- **External CSS**
    - Linking external stylesheets
    - Benefits of modular CSS
- **Comparison of CSS Approaches**
    - When to use each approach

Selectors

- **Overview of CSS Selectors**
  - Purpose and importance
- **Universal Selector**
  - Syntax: *
  - Use cases
- **Group Selector**
  - Syntax: selector1, selector2
  - Combining multiple selectors
- **Id Selector**
  - Syntax: #id
  - Specificity and uniqueness
- **Class Selector**
  - Syntax: .class
  - Reusability across elements
- **Tag/Element Selector**
  - Syntax: tagName
  - Applying styles to specific HTML tags
- **Descendant Selector**
  - Syntax: ancestor descendant
  - Targeting nested elements
- **Child Selector**
  - Syntax: parent > child
  - Direct child elements only
- **Examples and Exercises**
  - Practical tasks for each selector

- **Box Model**
  - Border
    - Border-width
    - Border-style

        - Border-Color
        - Border-radius
    - Margin
        - Margin-Top
        - Margin-Bottom
        - Margin-Left
        - Margin-Right
    - Padding
        - Padding-top
        - Padding-Bottom
        - Padding-Left
        - Padding-Right
    - Box Shadow
- **Flex Concept**
    - Display
    - Flex Direction
        - Row
        - Column
        - Row-Reverse
        - Column-Reverse
    - Flex Wrap
    - Justify Content
        - Start
        - Flex-start
        - Center
        - End
        - Flex-End
        - Space Between
        - Space Evenly
        - Space Around
    - Align Items
        - Flex-Start

        - Start
        - Center
        - End
        - Flex-End
    - Orders
    - Flex-basis
- **Grid Concept**
    - Display
    - Grid-template-columns
    - Grid-template-rows
    - Gap
    - Grid Lines
    - Grid column
    - Grid Row

- **Position in CSS**
    - Static
    - Relative
    - Absolute
    - Fixed
    - Sticky

- **Media Queries**
    - Component of Media Queries
        - Media Types
            1. All
            2. Screen
            3. Print
        - Media Features
            1. Width
            2. Height
            3. Max-width

        4. Max-height
        5. Min-width
        6. Min height
        7. Orientation
    - Logical Operators
        1. And
        2. Not
        3. Only
        4. Or

- **CSS Backgrounds**
    - background-color
    - background-image
    - background-repeat
    - background-position
    - background
- **CSS Colors**
    - Color name
    - RGB
    - RGBA
    - HEX
- **CSS Transform**
    - Translate
    - Rotate
    - Scale
- **CSS Transition**
    - Transition Delay
    - Transition Duration
    - Transition Property

Bootstrap

- **Introduction to Bootstrap? and its Features**
- **Grid System**
- **Components**
    - Basic Typography
        - h1
        - h2
        - h3
        - h4
        - h5
        - h6
        - display-1
        - display-2
        - display-3
        - display-4
        - display-5
        - Text-c
        - enter
    - Color
        - Text-Primary
        - Text-Secondary
        - Text-Warning
        - Text-Danger
        - Text-Info
        - Text-Success
        - Text-Dark
        - Text-Light
    - **Background**
        - bg-primary
        - bg-secondary
        - bg-warning

        - bg-danger
        - bg-success
        - bg-dark
        - bg-light
        - bg-info
    - **Button**
        - Btn
        - Btn-primary
        - Btn-secondary
        - Btn-dark
        - Btn-warning
        - Btn-danger
        - Btn-outline-primary
        - Btn-outline-secondary
        - Btn-outline-dark
        - Btn-outline-warning
        - Btn-outline-danger
        - Btn-sm
        - Btn-lg
        - Btn-md
        - Btn-link

    - **Card**
        - Card-header
        - Card-body
        - Card-footer
        - Card-title
        - Card-Subtitle
    - **Forms**
        - Form-Group
        - Form-Control
        - Form-Check

        - Form-select
    - **Pagination**
        - Page-item
        - Pagination
    - **Carousel**
        - Carousel-inner
        - Carousel-item
        - Carousel
        - Slide
        - Fade
        - Data-bs-ride
        - active
        - carousel-caption
        - Carousel-control-prev
        - Carousel-control-next
        - Carousel-control-next-icon
        - Carousel-control-prev-icon
    - **Modal**
        - Modal-dialog
        - Modal
        - Modal-content
        - Modal-header
        - Modal-body
        - Modal-footer
        - Data-bs-dismiss
        - Btn-close
        - Modal-lg
        - Modal-sm
        - Modal-md
    - **Table**
        - Table
        - Table-hover

        - Table-dark
        - Table-Primary
        - Table-secondary
        - Table-dark
        - Table-warning
        - Table-Striped

Javascript

- **Introduction to Javascript**
    - What is JS
    - Purpose of JS
    - Features of JS
    - Program
    - Execution
- **Installation of NodeJS**
- **Variables**
    - Declaration
    - Assignment
    - Initialization
    - Scope Statement
    - Var
    - Let
    - Const
- **Data Types**
    - Number
    - Boolean
    - String
    - Null
    - Undefined
    - BigInt

    - Symbol
    - Object
- **Functions**
    - What is Function
    - Purpose of Function
    - Syntax to create Function
    - Function Definition
    - Function Scope
    - Function Block
    - Function Call
    - Types of Function
        - Normal function
        - Parameters
        - Arguments
        - Return
        - Callback
        - Function Expression
        - Anonymous
        - Arrow function
        - Async Function
        - Higher Order function
    - Syntax
    - Purpose
    - Examples
- **Object**
    - What is Object
    - Purpose
    - Creation of Object
    - Properties
    - Crud Operation on Object
        - Read
        - Insert

        - Update
        - Delete
    - Object Method
        - Seal
        - Assign
        - Key
- **Array**
    - Syntax
    - Purpose
    - Literal Notation
    - Index
- **Array Methods**
    - Push
    - Pop
    - Shift
    - Unshift
    - ForEach
    - Map
    - Filter
    - Splice
    - Slice
    - Includes
    - indexOf
- **Selection Statement and Loops**
    - If
    - Else
    - Else if
    - Switch
    - For
    - For of
    - For in

- **Spread**
    - How to copy properties from one object into another object.
    - How to copy elements from one array into another array.
- **Rest**
    - Rules of parameter
    - Order of parameter
- **Destructuring**
    - Object
    - Array
- **Scopes**
    - Global Scope
    - Function Scope
    - Block Scope
    - Lexical Scope
    - Var
    - Let
    - Const
    - Difference between var, let and const
- **This keyword**
    - Browser Context
        - Arrow function
        - Named Function
        - Object
    - Node context
        - Arrow Function
        - Named Function
        - Object
- **Call**
    - What is call
    - Syntax
    - Purpose
    - How to work with it

- **Apply**
    - What is call
    - Syntax
    - Purpose
    - How to work with it
- **Bind**
    - What is call
    - Syntax
    - Purpose
    - How to work with it
- **Difference between call(), apply() and Bind()**
- **Clousers**
    - What is Closures
    - How to create it
    - Syntax
    - Purpose
- **Promises**
    - Creation of Promise
    - Resolve
    - Reject
    - States of Promise
        - Pending
        - Rejected
        - Fulfilled
    - Accessing the data from Promise
        - Then catch
        - Async Await
        - Try catch
- **DOM and DOM Manipulation**
    - What is DOM
    - Dom Objects
    - How to access the DOM Object

    - How to manipulate HTML with DOM
    - getElementById
    - getElementsByClass
    - QuerySelector
    - innerText
    - innerHTML
    - Append
    - Appnedchild
- **CreateElement**
    - How to create a DOM Element
    - How to add content in it
    - How to add DOM element in the DOM
    - How to remove Dom Element
- **Fetch API**
    - Get request
- **OOPs (Recorded Form)**
    - Class
    - Object
    - Constructor
    - Inheritance
    - Polymorphism with overriding
- **Module Concept in JS**
    - What is module concept
    - Why we use it
    - Named export
    - Default export
    - Importing named export
    - Importing default export

React JS

Introduction to React JS

- What is React JS
- Purpose of React JS
- Why do we required React JS
- Features of React JS
- Drawback of HTML and JS to create UI

React Elements

- What is React Element
- How to create React Element
- Integration of HTML and React
- How to add Inline, Internal and external CSS
- Detailed Understanding About the React.createElement()
- How to create User Interface with React Element

ReactDOM

- What is React DOM
- How to integrate React DOM with HTML
- How ReactDOM is used to add React Element in the DOM
- Understanding of ReactDOM.render()
- Virtual DOM
- How Virtual DOM works

React Element with Functions Concept

- React Element inside the function
- Parameters and arguments
- New Way of Calling the Functions

JSX

- What is JSX
- Syntax of JSX
- How JSX is different from HTML
- Rules of JSX
- Integrating babel with Html
- Creating UI with JSX
- Advantages of JSX
- How JSX simplifies Creation of UI in React JS
- What is babel
- Integration of babel with HTML

React Components

- What is React Component
- Advantage of React Component
- How to create React Component
- Types of React Component
- Introduction to Functional Component
- Creating the Functional Component
- Introduction to Class Component
- Creating the class Components
- How components can be used for Reusability

Props

- What is Props
- Purpose of the props
- How to use Props in Functional Component and Class Component
- How to pass Props
- Access the Props
- Pass the different types of data as a props
- Props types

Vite Tool

- What is Vite Tool
- How Vite is used to create Basic React Application
- How to run and stop React Application
- Accessing the React Application
- Understanding the Folder Structure of React Application
- NPM
- What is Node Package Manager
- How to install different Packages using NPM

Functional Components

- Understanding of Functional Component in Detail
- How to create Functional Component
- Why Functional Components are used over class Components
- How to render the Functional Component
- Sequence of Calling the Functional Component
- Flow of React Application

Class Components (Recorded Form)

- Understanding of Class Components
- How to create Class Component
- How Class Components are Different Functional Component
- Rendering of the class Component

Introduction to Hooks

- Introduction to the Hooks
- Why hooks are introduced
- Rules of using the hooks
- How to import the Hooks and use it
- Listing the important hooks

State and setState

- What is State
- Why do we required state
- How state can be used to create Dynamic User Interface
- Creation of State
- Introduction to first hook useState()
- Understanding of useState()
- How setState() is used and purpose of it
- Understanding in depth of setState and its Working
- Implementing the Counter App
- Implementing the Dynamic Card with Dark and Light Theme
- Implement Theme feature

How to Integrate CSS with React

- How to use Inline Css using style attribute in JSX
- How to integrate External CSS
- Problem with Css
- Using className attribute

Rendering the List using Map()

- What is map() in JS
- How it Works
- Understanding in details about map()
- How map() used to create UI
- Iterating Through Map
- Keys and List.

Axios

- What is Axios
- Installing and Integrating Axios with React App
- How to do get() Request with Axios
- Handling the Promise with then and catch

- Handling the Promise with async await

Form Management and Controlled Components

- How to create Form
- Managing the Form using React JS
- onChange event
- Managing the Form using State Concept
- Controlled Form Components

Json Server

- How to create Json Server
- Add the Data in the JSON Server
- Understanding About Client Server Architecture
- Fetching the data From Server using get request
- Understanding of POST, PUT and DELETE Request
- CRUD Operation using JSON Server

Interaction between Components

- Understanding the Relationship between Components
- Parent Child Relation
- Sharing the Data From Parent Component to Child Component using Props
- Props Drilling
- Problems With Props Drilling
- Introduction to Context

Context API

- Introduction to Context API
- Why Context API
- How Context API solves the Problems of Props Drilling
- Limitation of Context API
- How to Create the Context
- How to access Provider Component
- Understanding or Provider Component
- Storing the Data in Context

- How to make Available the context data to Child Components
- useContext() hook
- Purpose of useContext() hook
- How to access data from context using the useContext() hook

React Routing

- What is Routing
- How to implement routing in React App
- Installing and Configuring the react-router-dom
- BrowserRouter
- Routes
- Route
- Link
- Navigate
- Outlet
- useParams() hook
- useNavigate() hook
- Nested Routing

useRef() hook

- What is useRef() hook
- How it works
- Purpose of the useRef() hook
- Syntax of useRef() hook
- How useRef() hook used to manage the data or store the data
- Difference between useRef() hook and useState() hook
- DOM Manipulation using useRef() hook

DOM Manipulation and UnControlled Components

- How to manipulate the DOM using useRef() hook
- Change the Content of JSX Element
- Change the Style of JSX Element
- Managing the Form using useRef() hook

- What is UnControlled Components

useEffect() hook (Recording will be Provided)

- What are sideEffects in the React
- Pure functions in JS
- Lifecycle of Components
- Phases of Lifecycle
- What is Mounting and Unmounting
- Mount Phase
- Unmount Phase
- Update Phase
- How useEffect can be used to perform sideEffects in Different Phases of component
- Understanding How useEffect() hook works

useReducer() hook

- Understanding of useReducer() hook
- How to manage complex state operation in the reducer
- Reducer
- Dispatch
- Action object
- Types
- Difference between useState() and useReducer() hook

Lazy Loading

- What is Lazy Loading
- Benefits of Lazy Loading
- How to Lazy Loading will improve Performance
- Implementation of Lazy Loading

Redux with Functional Component

- What is Redux

- Why Redux
- How redux will help in state management
- Installing and Configuring redux
- Store
- Dispatch
- Reducer
- How to combine Multiple reducers
- Configuring Reducers with Redux Store
- Action
- ActionCreator
- Action Types
- Redux Pattern
- useSelecto() hook
- useDispatch() hook
- Implementing Redux in React Application

Backend Development with

Node JS, Express JS, and MongoDB

- **Introduction to Node JS**
  - What is NodeJS
  - Javascript Runtime
  - Open Source
  - Purpose
- **Recap of Javascript**
  - Object
  - Array
  - Callback

- **Module Concept using Common JS Module Pattern**
    - What is common js
    - How to export Functions and variables
    - How to import functions and variables
- **Callbacks**
    - Introduction to Callbacks
    - Definition of a Callback
    - Why Callbacks are Important in JavaScript
    - Passing Functions as Arguments
    - Writing a Callback Function
    - Simulating Asynchronous Behavior with Callbacks
    - Refactoring Callback Hell into Promises or async/await

- **Callback hell**
    - What is Callback hell
    - How to implement it
    - Problems with Callback hell
    - 2 Use Cases of callback hell
- **Promises in JS**
    - Creation of Promise
    - Resolve
    - Reject
    - States of Promise
        - Pending
        - Rejected
        - Fulfilled
    - Accessing the data from Promise

        - Then catch
        - Async Await
        - Try catch

- **Difference between callback hell and Promises**
- **Node Module System**
    - File
        - Readfile
        - Writefile
        - Rename file
        - Delete file
        - Creating new file
    - Path
        - path.basename()
        - path.dirname()
        - path.extname()
        - path.join()
        - path.resolve()
        - path.normalize()
        - path.relative()
        - path.parse()
        - path.format()
        - path.isAbsolute()
        -
    - Os
        - os.arch()
        - os.cpus()
        - os.endianness()
        - os.freemem()
        - os.homedir()
        - os.hostname()
        - os.loadavg()

        - os.networkInterfaces()
        - os.platform()
        - os.totalmem()

    - Http
        - http.createServer()
        - http.request()
        - http.get()
        - http.Server.listen()
        - http.Server.close()
        - http.Server.on()
        - http.IncomingMessage
        - http.ServerResponse
        - http.setHeader()
        - http.getHeader()

- **Asynchronous Nature of Node JS**
    - Javascript Engine
    - Execution Context
    - Callstack
    - Libuv
    - Thread Pool
    - Task Queue
    - Microtask Queue
    - Event Loop
- **Web Server**
        - Definition and Role
        - Difference Between Client and Server
        - Why Use Node.js as a Web Server?
        - Comparison with Traditional Servers
        - Setting Up a Basic Node.js Web Server
        - Installing Node.js

    - Writing and Running Your First Web Server with http Module
    - Handling Basic HTTP Requests and Responses
- **Overview on How the Web Works**
    - Client-Server Architecture
    - Role of Client
    - Role of Server
    - HTTP Protocol Basics
    - Request Methods (GET, POST, PUT, DELETE, etc.)
    - Request and Response Structure (Headers, Body, Status Codes)
    - URL Breakdown (Protocol, Domain, Path, Query)
    - Lifecycle of an HTTP Request
    - Static vs Dynamic Content
    - What are Static Files?
    - Generating Dynamic Responses Using Node.js
    - Sending JSON Data as response
    - Sending File Content as response
    - Sending HTML File as a REsponse
    - Configuring CSS File
    - Sending Text Data as a Response
- **Routing in NodeJS**
    - Intro to Routing?
    - Definition and Importance
    - Role of Routing in Handling Different Endpoints
    - Basic Routing Using the http Module
    - Creating Routes for Different Paths
    - Handling Query Parameters and URL Parameters
- **Responses**
    - Web page as a response
    - Json as a response
    - Normal text as a response
    - Setting headers for a response

- **NPM**
    - Introduction to NPM
    - What is NPM?
    - Definition and Purpose
    - Role in Node.js Ecosystem
    - Why Use NPM?
    - Installing Node.js and NPM
        - Installation Steps (Windows, macOS, Linux)
        - Verifying Installation (node -v and npm -v)
        - Understanding NPM Versions
        - NPM CLI Versions
        - Updating NPM (npm install -g npm)

- **Packages**
    - What is an NPM Package?
    - Difference Between Local and Global Packages
- **Package Registry**
    - What is the NPM Registry?
    - Accessing the Registry via https://www.npmjs.com
- **Package Management Basics**
    - Installing, Updating, and Removing Packages
    - Difference Between Development and Production Depen
    - Working with package.json
    - What is package.json?
        - Purpose and Structure
        - Key Fields (name, version, dependencies)
        - Creating a package.json File
        - Using npm init and npm init -y
        - Manually Editing package.json

    - Understanding Dependency Versioning
    - Semantic Versioning (SemVer) Explained (^, ~, *)
    - Installing Packages
    - npm install <package> (Local Installation)
    - npm install -g <package> (Global Installation)
    - Removing Packages
    - npm uninstall <package>
    - Updating and Checking Dependencies
    - npm update
    - npm outdated
    - Lock Files (package-lock.json)
    - Alternatives to NPM
        - Yarn
        - PNPM
        - Differences Between NPM, Yarn, and PNPM
        - Creating a New Project with npm init
        - Installing and Using a Package (e.g., lodash)

Introduction to Express JS

- What is Express JS?
    - Definition and Purpose
    - Why Use Express for Web Applications
    - Comparison with Vanilla Node.js (Simplifies Routing and Middleware)
- Setting Up Express
    - Installing Express (npm install express)
    - Creating a Basic Express Server
    - Writing and Running Your First Express App
- Key Features of Express
    - Lightweight and Flexible Framework
    - Middleware Support
    - Simplified Routing

    - Integration with Other Tools and Libraries
- **Understanding of Web API**
    - What is a Web API?
    - Definition and Role in Web Development
- **Types of Web APIs**
    - REST APIs
    - SOAP APIs
    - GraphQL APIs

- **Components of a Web API**
    - Endpoints and Resources
    - HTTP Methods and Status Codes
    - Input and Output (Request Body, Query Parameters, and Responses)

- **Why Use Web APIs?**
    - Enabling Communication Between Systems
    - Supporting Multiple Platforms (Web, Mobile, IoT)

- **REST API Principles**
    - What is REST?
    - Definition (Representational State Transfer)
    - Characteristics of Restful APIs

- **REST Principles**
  - Statelessness
  - Client-Server Architecture
  - Uniform Interface
  - Resource-Based URLs

- **Best Practices for REST APIs**
  - Use Meaningful Resource Names
  - Handle Errors Gracefully
  - Version Your API
  - Secure the API (Authentication and Authorization)

HTTP Methods in REST APIs

- Overview of HTTP Methods
- Definition and Role
- Mapping CRUD Operations to HTTP Methods

GET

- Purpose (Retrieve Data)
- Examples of GET Endpoints
- Handling Query Parameters

POST

- Purpose (Create New Resources)
- Sending Data in the Request Body
- Validating Input Data

PUT

- Purpose (Update or Replace Resources)
- Differences Between PUT and PATCH

DELETE

- Purpose (Delete Resources)
- Handling Deletion and Response Codes

- **Building REST APIs with Express**
  - Setting Up Routes
  - Defining Routes for Different HTTP Methods
  - Using Route Parameters and Query Strings
  - Working with Middleware
  - Using Built-in Middleware (express.json(), express.urlencoded())
  - Creating Custom Middleware
- **Sending Responses**
  - JSON Responses (res.json)
  - Handling Errors (res.status, next)
- **Organizing Code**
  - Separating Routes, Controllers, and Middleware
  - Using Router Instances for Modularization

- **HTTP Status Codes**
  - **Overview**
    - Categories of Status Codes (1xx, 2xx, 3xx, 4xx, 5xx)
  - **Commonly Used Status Codes**
    - 200 (OK)
    - 201 (Created)
    - 400 (Bad Request)

        - 404 (Not Found)
        - 500 (Internal Server Error)
    - **How to Send Status Codes in Express**
        - res.status().send()

- **Hands-On Exercises**
    - Setting Up a Basic Express Server
    - Creating RESTful Endpoints for a Sample Application (e.g., To-Do List, Library System)
    - Implementing CRUD Operations Using GET, POST, PUT, and DELETE
    - Sending Proper Status Codes and Responses
    - Testing API Endpoints Using Tools like Postman or cURL

- **Understanding of Middleware**
    - **What is Middleware?**
        - Definition and Role in Express
        - Middleware as a Function Intercepting Requests/Responses
    - **Types of Middleware**
        - Built-in Middleware (e.g., express.json, express.urlencoded)
        - Third-party Middleware (e.g., morgan, cors)
        - Custom Middleware
    - **Middleware Execution Flow**
        - Request-Response Lifecycle in Express
        - Chaining and Execution Order
    - **Common Use Cases**
        - Logging
        - Authentication and Authorization

        - Data Validation
        - Error Handling

- **2. Custom Middleware**
    - **What is Custom Middleware?**
        - User-defined Functions for Specific Tasks
    - **How to Create Custom Middleware**
        - Syntax and Structure of Middleware (req, res, next)
    - **Middleware Parameters**
        - req (Request Object)
        - res (Response Object)
        - next (Function to Pass Control to the Next Middleware)

- **Examples of Custom Middleware**
    - Logging Middleware
    - Authentication Middleware
- **Placing and Using Middleware**
    - app.use for Global Middleware
    - Route-Specific Middleware
    - Middleware Priority and Execution Order

- **Routing in Express**
    - **What is Routing?**
        - Definition and Purpose
        - Routing as URL Mapping
    - **Setting Up Routes in Express**
        - app.get, app.post, app.put, app.delete
        - Route Parameters (req.params)

        - Query Strings (req.query)
    - **Dynamic Routing**
        - Capturing Parameters in Routes
    - **Router Instances**
        - Creating and Using express.Router()
        - Modularizing Routes into Separate Files
        - Combining Multiple Routers
- **Middleware in Routing**
    - Applying Middleware to Specific Routes
    - Grouping Middleware with Routers

- **Error Handling**
    - **What is Error Handling?**
        - Capturing and Managing Errors in Express
        - Importance of Consistent Error Responses
- **Custom Error-Handling Middleware**
    - Structure (err, req, res, next)
    - Creating a Centralized Error Handler
- **Handling 404 Errors**
    - Setting Up a Default Route for Unmatched Paths
- **Environment Variables**
    - **What are Environment Variables?**
        - Definition and Purpose
        - Storing Configuration Data (e.g., API Keys, Database Credentials)
    - **Using Environment Variables in Node.js**
        - Accessing Variables with process.env

        - Example: Setting Up a PORT Variable
    - **Configuring Environment Variables**
        - .env Files
        - Installing and Using dotenv Package

- **Introduction to MongoDB**
    - **What is MongoDB?**
        - Definition and Features
        - Comparison with Relational Databases
        - Use Cases for MongoDB (e.g., Big Data, IoT, Real-Time Applications)
    - **Why Choose MongoDB?**
        - Schema-less Structure
        - High Performance and Scalability
        - Flexible Data Model

- **Installation of MongoDB**
    - **Downloading MongoDB**
        - Supported Platforms (Windows, macOS, Linux)
        - Choosing the Right Version (Community vs Enterprise)
    - **Installing MongoDB**
        - Step-by-Step Installation Guide for Different Operating Systems
        - Setting Up MongoDB as a Service (Optional)
    - **Verification**
        - Running MongoDB Server (mongod)
        - Verifying Installation with Mongo Shell

- **Installation of Mongo Shell**
  - **What is Mongo Shell?**
    - Definition and Purpose
    - Interaction with MongoDB Server
  - **Installing Mongo Shell**
    - Standalone Installation (if required)
    - Using the Shell with MongoDB Tools

- **Connecting Mongo Shell with MongoDB Server**
  - **Starting the MongoDB Server**
    - Running the mongod Command
  - **Connecting to the Server via Mongo Shell**
    - Starting Mongo Shell (mongo)
    - Default Connection to localhost and Port 27017

- **Creating the Database**
  - **Overview of MongoDB Databases**
    - How Databases are Created Dynamically
  - **Creating a Database**
    - Using use <database-name>
    - Verifying Created Databases with show dbs

- **Collections**
  - **What is a Collection?**
    - Collections vs Tables in Relational Databases

    - **Creating Collections**
        - Dynamic Creation on Data Insertion
        - Using db.createCollection()
    - **Listing and Dropping Collections**
        - Commands (show collections, db.collection.drop())

- **BSON Format**
    - **What is BSON?**
        - Definition and How it Differs from JSON
        - Binary-Encoded JSON for Efficient Storage
    - **Key Features of BSON**
        - Support for Data Types Like Date, Binary, ObjectId

- **CRUD Operations**
    - **Create**
        - Inserting Documents (db.collection.insertOne, db.collection.insertMany)
    - **Read**
        - Retrieving Data with find() and Query Filters
    - **Update**
        - Modifying Documents with updateOne, updateMany, and $set
    - **Delete**
        - Removing Documents with deleteOne and deleteMany

- **Data Types in MongoDB**
  - **Overview of Supported Data Types**
    - Common Types: String, Number, Boolean, Array, Object
    - Special Types: ObjectId, Date, Binary, Null
  - **Examples of Storing Data Using Various Types**

- **Embedded Documents**
  - **What are Embedded Documents?**
    - Storing Related Data Within a Single Document
  - **Use Cases for Embedded Documents**
    - Benefits (Performance and Simplicity)
  - **Examples**
    - Nested Objects and Arrays

- **Relations in MongoDB**
  - **Types of Relations**
    - One-to-One,
    - One-to-Many,
    - Many-to-Many
  - **Modeling Relationships**
    - Embedded vs Referenced Approach
  - **Examples of Each Relation Type**

- **Operators in MongoDB**
  - **Query Operators**
    - $eq — Matches values equal to a specified value.

        - $ne — Matches values not equal to a specified value.
        - $gt — Matches values greater than a specified value.
        - $gte — Matches values greater than or equal to a specified value.
        - $lt — Matches values less than a specified value.
        - $lte — Matches values less than or equal to a specified value.
        - $in — Matches values in an array of specified values.
        - $nin — Matches values not in an array of specified values.
    - **Update Operators**
        - $set, $unset, $inc, $push
    - **Aggregation Operators**
        - $sum, $avg, $group, $match
    - **Logical Operators**
        - $and,
        - $or,
        - $not,
        - $nor

    - **Element Operator**
        - $exists — Matches documents where the field exists or does not exist.
        - $type — Matches documents where the field is of a specified BSON data type.
    - Evaluation Operators
        - $text — Performs text search on indexed fields.
        - $where — Matches documents that satisfy a JavaScript expression.
    - **Array Operator**
        - $all — Matches arrays containing all specified elements.

        - $elemMatch — Matches documents where at least one array element satisfies specified conditions.
        - $size — Matches arrays with a specified number of elements.
    - **Projection Operators (Optional Query Enhancement)**
        - $slice — Limits the number of elements returned from an array.
        - $meta — Includes metadata in query results (e.g., text search scores).
        - $elemMatch — Projects matching array elements.
    - **Hands-On Exercises**
        - Installing and Setting Up MongoDB
        - Creating a Database and Adding Collections
        - Performing CRUD Operations on Sample Data
        - Modeling Embedded Documents and Relationships
        - Writing Queries with Operators

- **Indexes in MongoDB**
    - **Introduction to Indexes**
        - What are Indexes?
        - Purpose of Indexes (Improved Query Performance)
        - Impact of Indexes on Read and Write Operations
    - **Single Field Index**
        - Definition and Use Case
        - Creating a Single Field Index
        - Querying with Single Field Indexes
        - Examples and Hands-On Practice
    - **Compound Field Index**
        - Definition and When to Use
        - Creating Compound Indexes
        - Syntax: db.collection.createIndex({ field1: 1, field2: -1 })

        - Importance of Index Order in Compound Indexes
        - Use Cases: Sorting and Filtering
        - Examples and Practice
    - **Multikey Index**
        - What is a Multikey Index?
        - Indexing Array Fields in MongoDB
        - Limitations of Multikey Indexes
        - Not Allowed on Fields with Both Arrays and Other Indexed Types
        - Examples of Querying with Multikey Indexes
    - **Text Index**
        - Overview of Text Indexes
        - Full-Text Search Capabilities in MongoDB
        - Creating Text Indexes
        - Using db.collection.createIndex({ field: "text" })
        - Querying with Text Indexes

- **Introduction to Aggregation in MongoDB**
    - **What is Aggregation?**
        - Definition and Purpose
        - Difference Between Aggregation Framework and Query Language
        - Common Use Cases (Data Transformation, Grouping, Analysis)
    - **Aggregation Pipeline**
        - Definition and Components
        - Pipeline Stages Overview
            - $lookup
            - $match (Filter Data)
            - $group (Group Documents)
            - $project (Transform Output Fields)

        - $sort (Order Documents)
        - $limit
- **Mongoose**
    - Introduction to Mongoose
        - What is Mongoose?
        - Benefits of Using Mongoose with MongoDB
        - Schema vs. Collection vs. Document
    - Defining Schemas
        - Creating a Schema
        - Adding Field Types and Validation
        - Using Schema Methods and Statics
    - Working with Models
        - Creating a Model from a Schema
        - CRUD Operations with Models
        - create(),
        - find(),
        - findById(),
        - updateOne(),
        - deleteOne(),
- **Authentication and Authorization using JWT**
    - **What is JWT (JSON Web Token)?**
        - Overview and Structure of JWT (Header, Payload, Signature)
        - Benefits of Using JWT for Authentication
    - **Implementing Authentication with JWT**
        - Setting Up Registration and Login Endpoints
        - Generating JWT Tokens
        - Storing Tokens on Client (Cookies or Local Storage)
    - **Authorization with JWT**
        - Protecting Routes Using Middleware
        - Verifying Tokens on Protected Routes
    - **Refreshing Tokens**

        - Why Token Expiry is Important
        - Implementing Refresh Tokens
- **Integration of Node.js, Express, and MongoDB**
    - Setting Up the Environment
        - Installing Dependencies (express, mongoose, dotenv)
        - Configuring MongoDB Connection with mongoose.connect()
    - **Building an Express Server**
        - Creating Routes for CRUD Operations
        - Middleware for Parsing JSON and Handling Errors
        - Connecting with MongoDB
        - Defining and Using Mongoose Models in Routes
        - Handling Query Results (e.g., find, save, update)
        - Using Try-Catch for Route Handlers
        - Postman or cURL for API Testing

- **Integration with React**
    - **Overview of MERN Stack**
        - Why Use React with Node.js, Express, and MongoDB?
        - Architecture of a Full-Stack MERN Application
    - **Connecting Frontend and Backend**
        - Setting Up Proxy in React for API Requests
        - Using axios or fetch for HTTP Requests
    - **Managing State in React**
        - Storing Fetched Data in State
        - Using Context API or Redux for Global State Management
        - Authentication with React and JWT
        - Storing JWT in Cookies or Local Storage
        - Using JWT for Protected Routes in React
        - Implementing Login and Logout Features