1. **HTML and CSS Webpage:**

**Create a simple webpage that showcases your favorite hobby. Use HTML to structure the content and CSS to style the page, including adding colors, fonts, and images.**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Hiking Hobby</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f0f0f0;

margin: 0;

padding: 0;

}

header {

background-color: #333;

color: #fff;

text-align: center;

padding: 20px;

}

h1 {

font-size: 36px;

margin: 0;

}

main {

max-width: 800px;

margin: 20px auto;

padding: 20px;

background-color: #fff;

border-radius: 5px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);

}

p {

font-size: 18px;

line-height: 1.5;

}

img {

max-width: 100%;

height: auto;

border-radius: 5px;

}

footer {

background-color: #333;

color: #fff;

text-align: center;

padding: 10px;

}

</style>

</head>

<body>

<header>

<h1>My Hiking Hobby</h1>

</header>

<main>

<h2>About Hiking</h2>

<p>Hiking is one of my favorite hobbies. It allows me to connect with nature, stay active, and explore new places. There's nothing quite like the feeling of reaching a summit after a challenging hike.</p>

<img src="hiking.jpg" alt="Hiking in the mountains">

<h2>Favorite Hiking Spots</h2>

<p>Some of my favorite hiking spots include:</p>

<ul>

<li>Rocky Mountain National Park, Colorado</li>

<li>Zion National Park, Utah</li>

<li>The Appalachian Trail, Eastern United States</li>

</ul>

</main>

<footer>

&copy; 2023 My Hiking Hobby

</footer>

</body>

</html>

1. **JavaScript Form Validation:**

**Develop a web form with fields for name, email, and password. Implement JavaScript validation to ensure that all fields are filled correctly before submitting the form.**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Form Validation</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

label {

display: block;

margin-bottom: 8px;

}

input {

width: 100%;

padding: 8px;

margin-bottom: 16px;

box-sizing: border-box;

}

button {

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

</style>

</head>

<body>

<form id="myForm">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="button" onclick="validateForm()">Submit</button>

</form>

<script>

function validateForm() {

var name = document.getElementById('name').value;

var email = document.getElementById('email').value;

var password = document.getElementById('password').value;

// Simple validation

if (name.trim() === '') {

alert('Name cannot be empty');

return;

}

if (email.trim() === '') {

alert('Email cannot be empty');

return;

}

// Basic email validation using a regular expression

var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailRegex.test(email)) {

alert('Invalid email format');

return;

}

if (password.trim() === '') {

alert('Password cannot be empty');

return;

}

// If all validations pass, submit the form (in this case, just show an alert)

alert('Form submitted successfully!');

}

</script>

</body>

</html>

1. **Node.js Server with Express:**

**Build a basic server using Node.js and Express. Create routes to handle HTTP requests like GET and POST and respond with simple JSON data.**

const express = require("express");

const app = express();

const PORT = process.env.PORT || 3000;

app.use(express.json());

//JSON file instead of DB

const data = [

{id: 1, name: "Abhi"},

{id: 2, name: "Bhuvan"},

{id: 3, name: "Chetan"},

{id: 4, name: "Deeksha"},

];

app.get("/",(req,res)=>{

res.send("Home");

});

app.get("/api/data", (req,res)=>{

res.json(data);

});

app.post('/api/data', (req, res) => {

const nItem = req.body;

data.push(nItem);

res.json(data);

});

app.listen(PORT, (err)=>{

if(err) console.log(err);

console.log("Server running on port: ",PORT);

});

1. **Database Integration:**

**Extend the previous Node.js server by integrating a database (e.g., SQLite or MongoDB). Implement endpoints to perform CRUD operations on a dataset.**

// Import necessary modules

const express = require("express"); // Express framework for building web applications

const mongoose = require("mongoose"); // Mongoose for MongoDB object modeling

const app = express(); // Create an instance of Express application

const PORT = process.env.PORT || 3000; // Set the port for the server, use environment variable or default to 3000

// Middleware to parse incoming JSON requests

app.use(express.json());

// Connect to MongoDB

mongoose.connect("mongodb://localhost:27017/MCE2", { // Connect to the MongoDB database named "MCE2"

useNewUrlParser: true, // Use new URL parser

useUnifiedTopology: true, // Use new Server Discovery and Monitoring engine

})

.then(()=>{

console.log("Connection to DB Successful !"); // Log success message if connection is successful

})

.catch((err)=>{

console.log("Connection to DB Failed !"); // Log error message if connection fails

})

// Define a mongoose schema for the data

const dataSchema = new mongoose.Schema({

name: String,

usn: String,

sem: String

});

// Create a mongoose model based on the schema

const Data = mongoose.model("Data", dataSchema);

// Define a route for the root endpoint

app.get("/", (req, res) => {

res.send("Home"); // Respond with "Home" when the root endpoint is accessed

});

// Retrieve all data from MongoDB

app.get("/api/data", async (req, res) => {

try {

const allData = await Data.find(); // Retrieve all data entries from the "Data" collection in MongoDB

res.json(allData); // Respond with the retrieved data in JSON format

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Create a new data entry in MongoDB

app.post('/api/data', async (req, res) => {

try {

const nItem = req.body; // Get the data from the request body

const newData = new Data(nItem); // Create a new instance of the Data model with the received data

await newData.save(); // Save the new data entry to the MongoDB collection

res.json(newData); // Respond with the saved data in JSON format

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Update data by ID in MongoDB (This part is missing in the provided code)

// Start the server and listen on the specified port

app.listen(PORT, (err) => {

if (err) console.log(err); // Log any errors that occur while starting the server

console.log("Server running on port: ", PORT); // Log a message indicating that the server is running on the specified port

});

1. **RESTful API:**

**Design and implement a RESTful API using Node.js, Express, and a database of your choice. Define endpoints for managing resources, such as creating, reading, updating, and deleting data.**

// Import required modules

const express = require("express"); // Import the Express framework

const mongoose = require("mongoose"); // Import Mongoose for MongoDB object modeling

const app = express(); // Create an instance of the Express application

const PORT = process.env.PORT || 3000; // Set the port for the server, use environment variable or default to 3000

app.use(express.json()); // Middleware to parse incoming JSON requests

// Connect to MongoDB

mongoose.connect("mongodb://localhost:27017/dataDB\_1", {

useNewUrlParser: true, // Use new URL parser

useUnifiedTopology: true, // Use new Server Discovery and Monitoring engine

})

.then(()=>{

console.log("Connection to DB Successful !"); // Log success message if connection is successful

})

.catch((err)=>{

console.log("Connection to DB Failed !"); // Log error message if connection fails

})

// Define a Mongoose schema for the data

const dataSchema = new mongoose.Schema({

id: Number,

name: String,

});

// Create a Mongoose model based on the schema

const Data = mongoose.model("Data", dataSchema);

// Define a route for the root endpoint

app.get("/", (req, res) => {

res.send("Home"); // Respond with "Home" when the root endpoint is accessed

});

// Retrieve all data from MongoDB

app.get("/api/data", async (req, res) => {

try {

const allData = await Data.find(); // Retrieve all data entries from the "Data" collection in MongoDB

res.json(allData); // Respond with the retrieved data in JSON format

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Create a new data entry in MongoDB

app.post('/api/data', async (req, res) => {

try {

const nItem = req.body; // Get the data from the request body

const newData = new Data(nItem); // Create a new instance of the Data model with the received data

await newData.save(); // Save the new data entry to the MongoDB collection

res.json(newData); // Respond with the saved data in JSON format

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Update data by ID in MongoDB

app.put('/api/data/:id', async (req, res) => {

try {

const id = req.params.id; // Get the ID from the request parameters

const updatedItem = req.body; // Get the updated data from the request body

await Data.findOneAndUpdate({ id }, updatedItem); // Find and update the corresponding document in MongoDB

res.json(updatedItem); // Respond with the updated data in JSON format

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Delete data by ID from MongoDB

app.delete('/api/data/:id', async (req, res) => {

try {

const id = req.params.id; // Get the ID from the request parameters

await Data.findOneAndDelete({ id }); // Find and delete the corresponding document in MongoDB

res.json({ message: "Data deleted" }); // Respond with a JSON message indicating successful deletion

} catch (err) {

console.error(err);

res.status(500).json({ message: "Internal Server Error" }); // Handle errors and respond with an internal server error

}

});

// Start the server and listen on the specified port

app.listen(PORT, (err) => {

if (err) console.log(err); // Log any errors that occur while starting the server

console.log("Server running on port: ", PORT); // Log a message indicating that the server is running on the specified port

});

1. **React Component Library:**

**Create a library of reusable React components. Build components like buttons, cards, and modals and use them in a sample React application.**

1. **Setting Up Your React Project:** Start by creating a new React project using Create React App.

npx create-react-app react-components-library

cd react-components-library

1. **Creating the Components:** Inside the **src** folder, create a new folder named **components**. Inside this folder, create three files for each component: **Button.js**, **Card.js**, and **Modal.js**.

**Button.js**

// Import the React library

import React from 'react';

// Import PropTypes for defining the types of component props

import PropTypes from 'prop-types';

// Define a functional component named Button that takes onClick and label as props

const Button = ({ onClick, label }) => {

// Return JSX for rendering a button element

return (

<button onClick={onClick} className="button">

{label}

</button>

);

};

// Define PropTypes for the Button component to enforce prop types

Button.propTypes = {

onClick: PropTypes.func.isRequired, // onClick prop should be a function and is required

label: PropTypes.string.isRequired, // label prop should be a string and is required

};

// Export the Button component for use in other parts of the application

export default Button;

**Card.js**

// Import the React library

import React from 'react';

// Import PropTypes for defining the types of component props

import PropTypes from 'prop-types';

// Define a functional component named Card that takes title and content as props

const Card = ({ title, content }) => {

// Return JSX for rendering a div with title and content

return (

<div className="card">

<h2>{title}</h2>

<p>{content}</p>

</div>

);

};

// Define PropTypes for the Card component to enforce prop types

Card.propTypes = {

title: PropTypes.string.isRequired, // title prop should be a string and is required

content: PropTypes.string.isRequired, // content prop should be a string and is required

};

// Export the Card component for use in other parts of the application

export default Card;

**Modal.js**

// Import the React library

import React from 'react';

// Import PropTypes for defining the types of component props

import PropTypes from 'prop-types';

// Define a functional component named Modal that takes isOpen, onClose, and children as props

const Modal = ({ isOpen, onClose, children }) => {

// Return JSX for rendering a modal overlay if isOpen is true

return (

isOpen && (

<div className="modal-overlay">

<div className="modal">

<button onClick={onClose} className="close-button">

X

</button>

{children}

</div>

</div>

)

);

};

// Define PropTypes for the Modal component to enforce prop types

Modal.propTypes = {

isOpen: PropTypes.bool.isRequired, // isOpen prop should be a boolean and is required

onClose: PropTypes.func.isRequired, // onClose prop should be a function and is required

children: PropTypes.node.isRequired, // children prop can be any node (e.g., JSX, string) and is required

};

// Export the Modal component for use in other parts of the application

export default Modal;

1. **Using Components in App:** Replace the content of **src/App.js** with the following code:

import React, { useState } from 'react';

import './App.css';

import Button from './components/Button';

import Card from './components/Card';

import Modal from './components/Modal';

const App = () => {

const [isModalOpen, setIsModalOpen] = useState(false);

return (

<div className="app">

<Button label="Open Modal" onClick={() => setIsModalOpen(true)} />

<Card title="Sample Card" content="This is a sample card component." />

<Modal isOpen={isModalOpen} onClose={() => setIsModalOpen(false)}>

<h2>Modal Content</h2>

<p>This is the content of the modal.</p>

</Modal>

</div>

);

};

export default App;

1. **Styling:** Create a simple CSS file for styling. Create a new file named **App.css** inside the **src** folder:

.app {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

height: 100vh;

}

.button {

padding: 10px 20px;

margin-bottom: 20px;

}

.card {

border: 1px solid #ccc;

padding: 20px;

margin-bottom: 20px;

width: 300px;

text-align: center;

}

.modal-overlay {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: rgba(0, 0, 0, 0.5);

display: flex;

align-items: center;

justify-content: center;

}

.modal {

background: white;

padding: 20px;

width: 400px;

border-radius: 8px;

position: relative;

}

.close-button {

position: absolute;

top: 10px;

right: 10px;

cursor: pointer;

background: none;

border: none;

font-size: 18px;

}

1. **Run your App:** Start your React app:

**npm start**

Open your browser and navigate to **http://localhost:3000** to see your sample React application using the reusable components.