TYPESCRIPT ASSIGNMENT

1.

interface Student{

    name:string;

    rollNumber:number;

    course:string;

    phoneNumber:string;

}

function printStudentDetails(student:Student){

    console.log(`Name: ${student.name}`);

    console.log(`Roll Number: ${student.rollNumber}`);

    console.log(`Course: ${student.course}`);

    console.log(`Phone Number: ${student.phoneNumber}`);

    console.log('-------------------------');

}

const student1: Student = { name: 'Riya', rollNumber: 101, course: 'TypeScript', phoneNumber: '1234567890' };

const student2: Student = { name: 'Arjun', rollNumber: 102, course: 'Angular', phoneNumber: '9876543210' };

printStudentDetails(student1);

printStudentDetails(student2);

2.

class Book {

    title: string;

    author: string;

    price: number;

    constructor(title: string, author: string, price: number) {

      this.title = title;

      this.author = author;

      this.price = price;

    }

    getDetails(): string {

      return `Book: ${this.title} by ${this.author} - ₹${this.price}`;

    }

  }

  class PremiumBook extends Book {

    deliveryCharge: number;

    constructor(title: string, author: string, price: number, deliveryCharge: number) {

      super(title, author, price);

      this.deliveryCharge = deliveryCharge;

    }

    // Override getDetails to include delivery charge

    getDetails(): string {

      const totalPrice = this.price + this.deliveryCharge;

      return `Book: ${this.title} by ${this.author} - ₹${totalPrice} (Includes Delivery)`;

    }

  }

  // Sample Data

  const normalBook = new Book("Clean Code", "Robert C. Martin", 500);

  const premiumBook = new PremiumBook("Design Patterns", "Erich Gamma", 700, 50);

  // Display details

  console.log(normalBook.getDetails());

  console.log(premiumBook.getDetails());

3.

import express from "express";

import { createServer } from "http";

import WebSocket, { WebSocketServer } from "ws";

import path from "path";

const app = express();

const port = 3000;

// Serve static files (our client HTML)

app.use(express.static(path.join(\_\_dirname, "public")));

const server = createServer(app);

// Set up WebSocket server

const wss = new WebSocketServer({ server });

wss.on("connection", (ws: WebSocket) => {

  console.log("Client connected");

  ws.on("message", (data: WebSocket.RawData) => {

    const message = data.toString().trim();

    if (!message) return; // Ignore empty messages

    // Broadcast to all clients

    wss.clients.forEach(client => {

      if (client.readyState === WebSocket.OPEN) {

        client.send(message);

      }

    });

  });

  ws.on("close", () => {

    console.log("Client disconnected");

  });

});

server.listen(port, () => {

  console.log(`Server started at http://localhost:${port}`);

});

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <title>Simple Chat Client</title>

  <style>

    body { font-family: Arial, sans-serif; max-width: 600px; margin: 20px auto; }

    #messages { border: 1px solid #ccc; height: 300px; overflow-y: scroll; padding: 10px; margin-bottom: 10px; }

    #messageInput { width: 80%; padding: 8px; }

    #sendBtn { padding: 8px 12px; }

  </style>

</head>

<body>

  <h2>Chat</h2>

  <div id="messages"></div>

  <input type="text" id="messageInput" placeholder="Type your message" />

  <button id="sendBtn">Send</button>

  <script>

    const ws = new WebSocket(`ws://${window.location.host}`);

    const messagesDiv = document.getElementById("messages");

    const input = document.getElementById("messageInput");

    const sendBtn = document.getElementById("sendBtn");

    ws.onopen = () => {

      appendMessage("Connected to chat server.");

    };

    ws.onmessage = event => {

      appendMessage(event.data);

    };

    ws.onclose = () => {

      appendMessage("Disconnected from chat server.");

    };

    sendBtn.onclick = sendMessage;

    input.onkeydown = (e) => {

      if (e.key === "Enter") sendMessage();

    };

    function appendMessage(message) {

      const p = document.createElement("p");

      p.textContent = message;

      messagesDiv.appendChild(p);

      messagesDiv.scrollTop = messagesDiv.scrollHeight;

    }

    function sendMessage() {

      const msg = input.value.trim();

      if (!msg) return alert("Please enter a non-empty message");

      ws.send(msg);

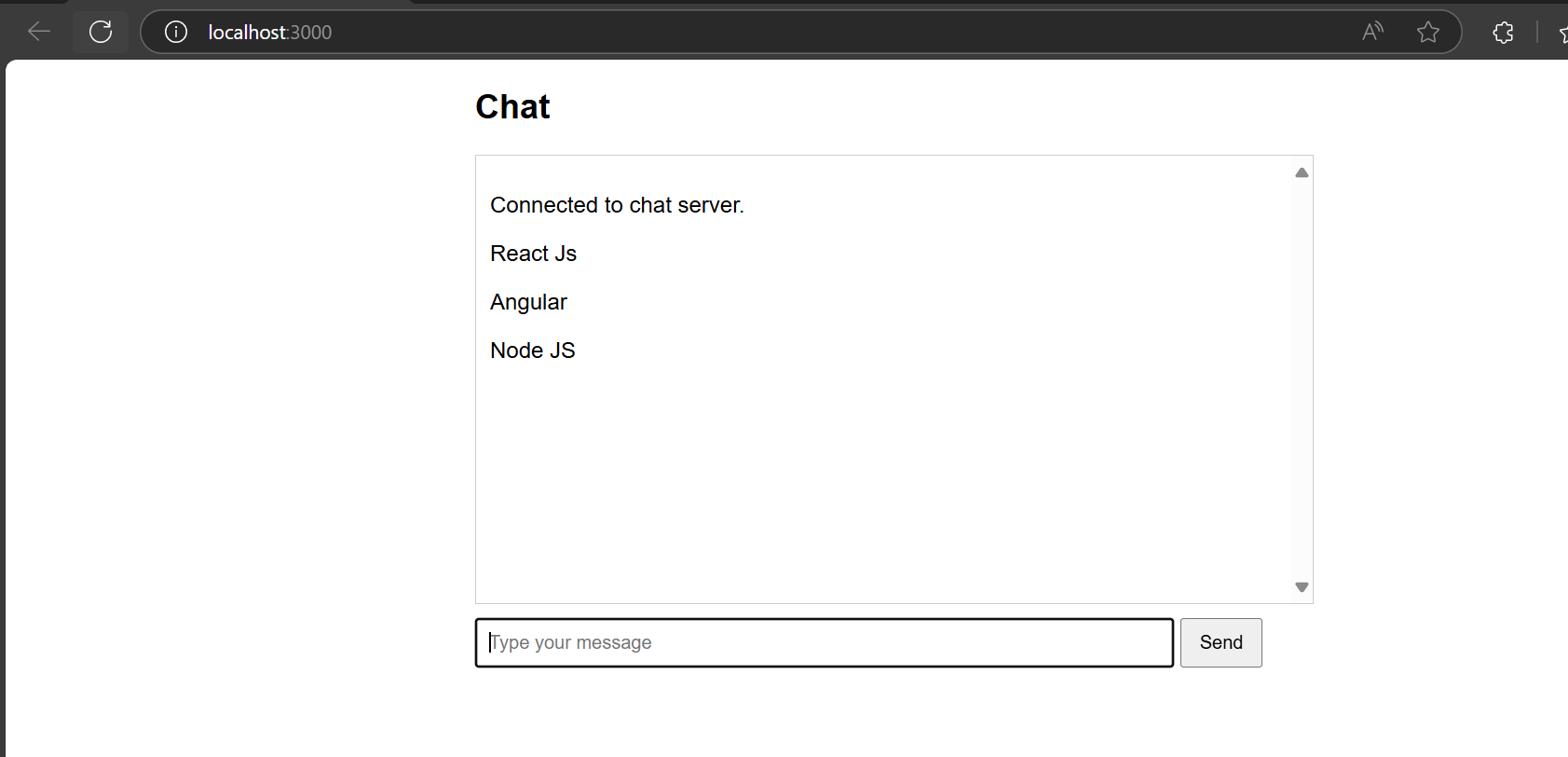
      input.value = "";

    }

  </script>

</body>

</html>



4.

interface Article {

  title: string;

  content: string;

}

// Create 15 dummy articles

const articles: Article[] = [

  { title: "TypeScript Basics", content: "Introduction to TypeScript, its benefits, and basic types." },

  { title: "Understanding Interfaces", content: "How interfaces define object shapes in TypeScript." },

  { title: "Advanced Types", content: "Exploring union, intersection, and mapped types." },

  { title: "TypeScript and DOM", content: "Manipulating the DOM safely with TypeScript." },

  { title: "Generics in TypeScript", content: "Writing reusable components with generics." },

  { title: "TypeScript Enums", content: "Defining enums for better code clarity." },

  { title: "Modules and Namespaces", content: "Organizing code with modules and namespaces." },

  { title: "Async Programming", content: "Handling async operations using Promises and async/await." },

  { title: "Decorators in TypeScript", content: "Meta-programming with decorators." },

  { title: "Type Assertions", content: "Overriding inferred types using assertions." },

  { title: "Working with Classes", content: "OOP concepts in TypeScript with classes and inheritance." },

  { title: "TypeScript Configuration", content: "Customizing compiler options in tsconfig.json." },

  { title: "Error Handling", content: "Catching and managing errors in TypeScript apps." },

  { title: "Testing TypeScript", content: "Writing tests using Jest or Mocha." },

  { title: "Deploying TypeScript Apps", content: "Best practices for deployment and bundling." }

];

// Constants for pagination

const ARTICLES\_PER\_LOAD = 5;

const container = document.getElementById('articles-container') as HTMLDivElement;

let currentIndex = 0;

let isLoading = false;

function renderArticles(): void {

  if (currentIndex >= articles.length) {

    // No more articles to load

    return;

  }

  isLoading = true;

  // Simulate async loading delay

  setTimeout(() => {

    const nextArticles = articles.slice(currentIndex, currentIndex + ARTICLES\_PER\_LOAD);

    nextArticles.forEach(article => {

      const articleDiv = document.createElement('div');

      articleDiv.className = 'article';

      const title = document.createElement('h2');

      title.textContent = article.title;

      const content = document.createElement('p');

      content.textContent = article.content;

      articleDiv.appendChild(title);

      articleDiv.appendChild(content);

      container.appendChild(articleDiv);

    });

    currentIndex += ARTICLES\_PER\_LOAD;

    isLoading = false;

  }, 500); // Delay for simulation

}

// Debounce helper function

function debounce(func: () => void, wait: number) {

  let timeout: number | undefined;

  return () => {

    if (timeout !== undefined) {

      clearTimeout(timeout);

    }

    timeout = window.setTimeout(() => {

      func();

    }, wait);

  };

}

function handleScroll(): void {

  if (isLoading) return;

  // Check if user scrolled near bottom (100px threshold)

  const scrollTop = window.scrollY;

  const viewportHeight = window.innerHeight;

  const fullHeight = document.documentElement.scrollHeight;

  if (scrollTop + viewportHeight >= fullHeight - 100) {

    renderArticles();

  }

}

// Initial render

renderArticles();

// Add scroll listener with debounce

window.addEventListener('scroll', debounce(handleScroll, 200));

index.html

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <title>Blog Article Viewer - Infinite Scroll</title>

  <link rel="stylesheet" href="styles.css" />

</head>

<body>

  <h1>Blog Articles</h1>

  <div id="articles-container"></div>

  <script src="script.js"></script>

</body>

</html>

Styles.css

body {

    font-family: Arial, sans-serif;

    max-width: 700px;

    margin: 0 auto;

    padding: 20px;

    background: #f7f7f7;

  }

  h1 {

    text-align: center;

    margin-bottom: 30px;

  }

  #articles-container {

    display: flex;

    flex-direction: column;

    gap: 20px;

  }

  .article {

    background: white;

    padding: 20px;

    border-radius: 6px;

    box-shadow: 0 2px 5px rgba(0,0,0,0.1);

  }

  .article h2 {

    margin: 0 0 10px;

    font-size: 1.4em;

  }

  .article p {

    margin: 0;

    line-height: 1.5;

    color: #333;

  }