**Memory Block Allocation (Garbage Collection):**

**Simulate dynamic memory block allocation and deallocation using a singly linked list, where each node represents a memory block. Implement garbage collection to identify and release unreferenced blocks periodically**

<html lang="en">

<head>

<title>Memory Block Allocation</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: #f5f5f5;

}

.memory-blocks {

display: flex;

justify-content: center;

flex-wrap: wrap;

gap: 10px;

margin: 20px 0;

}

.block {

width: 60px;

height: 60px;

display: flex;

align-items: center;

justify-content: center;

font-size: 14px;

color: #fff;

border-radius: 5px;

background-color: #4CAF50;

position: relative;

}

.block.unreferenced {

background-color: #ff6b6b;

}

.controls {

margin-top: 20px;

}

button {

padding: 10px 20px;

margin: 5px;

font-size: 16px;

cursor: pointer;

}

</style>

</head>

<body>

<h1>Memory Block Allocation Simulation</h1>

<div class="memory-blocks" id="memoryBlocks"></div>

<div class="controls">

<button onclick="allocateMemory()">Allocate Block</button>

<button onclick="deallocateMemory()">Deallocate Block</button>

<button onclick="runGarbageCollection()">Run Garbage Collection</button>

</div>

<script>

class MemoryBlock {

constructor(id) {

this.id = id;

this.next = null;

this.referenced = true; // Default to true (active reference)

}

}

class MemoryManager {

constructor() {

this.head = null;

this.idCounter = 1;

}

allocate() {

const newBlock = new MemoryBlock(this.idCounter++);

newBlock.referenced = Math.random() > 0.3; // Randomize active reference status

if (!this.head) {

this.head = newBlock;

} else {

let current = this.head;

while (current.next) {

current = current.next;

}

current.next = newBlock;

}

this.render();

}

deallocate() {

if (!this.head) return;

this.head = this.head.next;

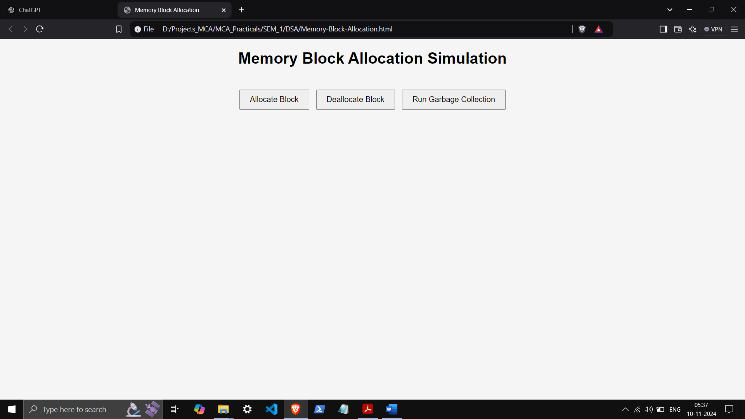
this.render();

}

garbageCollect() {

let current = this.head;

let prev = null;

 while (current) {

if (!current.referenced) {

if (prev) {

prev.next = current.next;

} else {

this.head = current.next;

}

} else {

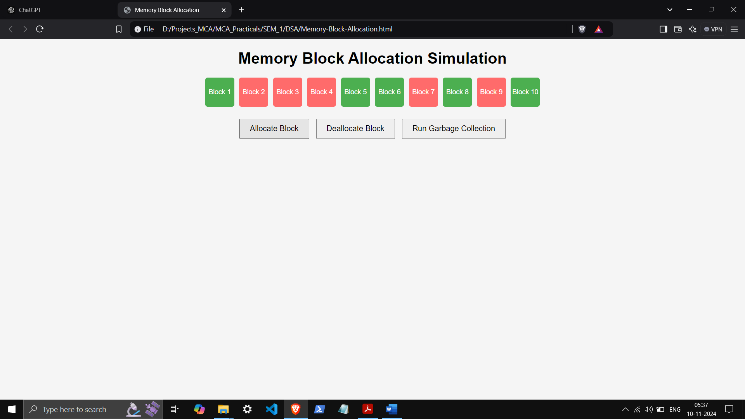
prev = current;

}

current = current.next;

}

this.render();

 }

render() {

const container = document.getElementById("memoryBlocks");

container.innerHTML = '';

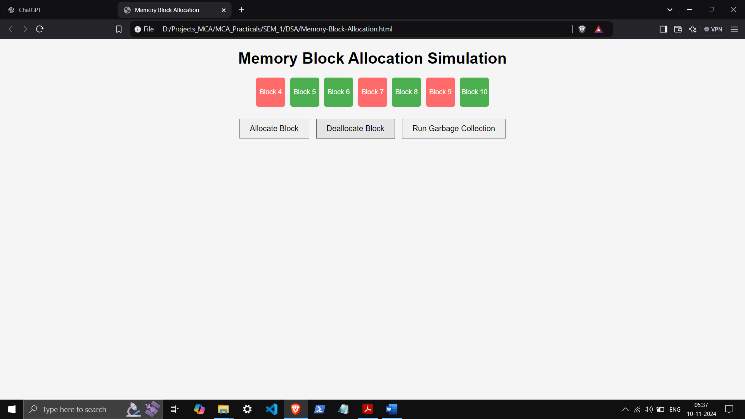
let current = this.head;

while (current) {

const block = document.createElement("div");

block.className = "block";

if (!current.referenced) block.classList.add("unreferenced");

 block.innerText = `Block ${current.id}`;

container.appendChild(block);

current = current.next;

}

}

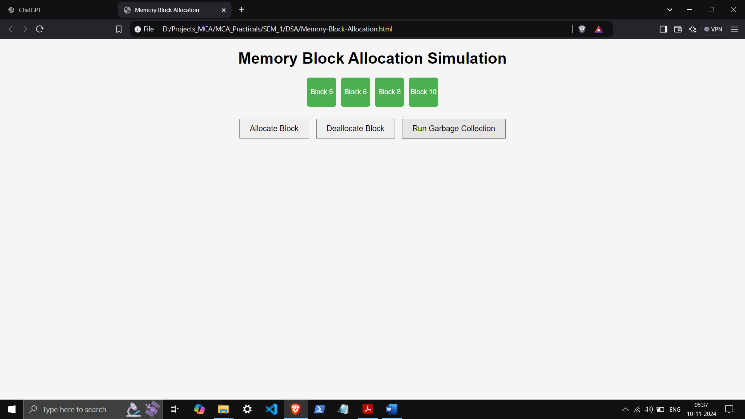
}

const memoryManager = new MemoryManager();

function allocateMemory() {

memoryManager.allocate();

}



function deallocateMemory() {

memoryManager.deallocate();

}

function runGarbageCollection() {

memoryManager.garbageCollect();

}

</script>

</body>

</html>