PROJECT REPORT

**COLLABORATORS:**

***Anjali Kadyan (12311971) [61]***

***Mohammad Aatif Khan (12310148) [60]***

***Vaibhav Verma (12305069) [59]***

Course: CSE316

Date: 14/04/2025

**Project Overview**

**Project Title**

**“Comprehensive Inter-Process Communication Framework”**

**Project Objective**

The primary objective of this project is to design and develop a comprehensive inter-process communication (IPC) framework that ensures efficient, secure, and reliable data exchange between processes. The project aims to:

Facilitate seamless communication between processes using various IPC methods such as pipes, message queues, and shared memory.

Provide a unified and flexible architecture that supports multiple IPC mechanisms based on system requirements.

Incorporate robust security features to prevent unauthorized access and ensure data integrity during inter-process communication.

Enhance system modularity and performance by streamlining process coordination and data sharing.

**Scope of Work**

The project encompasses the design and development of a robust inter-process communication (IPC) framework that supports efficient and secure data exchange between processes using various IPC mechanisms. The key deliverables include:

1. **Modular IPC Framework with Multi-Method Support**:  
   This core component of the project provides seamless communication between processes using multiple IPC techniques such as pipes, message queues, and shared memory. It includes abstraction layers that allow easy integration of different IPC mechanisms based on system needs, while ensuring reliable data transmission and minimal communication overhead.
2. **Security and Access Control Module**:  
   A critical part of the framework that implements comprehensive security measures to prevent unauthorized data access and tampering. This module includes authentication, permission settings, and data encryption where necessary, ensuring secure communication between processes in multi-user environments.
3. **Monitoring and Diagnostics Interface**:  
   A tool for developers and system administrators to monitor IPC activity, detect communication bottlenecks, and trace data flow between processes. This interface provides logs, metrics, and diagnostics to help with debugging and performance tuning.
4. **Comprehensive Documentation**:  
   Detailed documentation covering installation, configuration, usage, and extension of the IPC framework. It also includes explanations of the supported IPC methods, security architecture, and integration examples, aimed at both novice and experienced developers.

**INCLUSION**

The inclusions in this project are the essential components that enable the IPC framework to facilitate secure and efficient communication between processes. These include:

1. **Support for Multiple IPC Methods**:  
   Built-in support for standard inter-process communication techniques such as pipes, message queues, and shared memory. Each method is implemented with configurable options to suit different use cases and performance requirements.
2. **Data Serialization and Buffer Management**:  
   Mechanisms for packaging and unpackaging data to ensure compatibility and integrity during transmission. Includes dynamic buffer management to handle variable message sizes and prevent data loss or corruption.
3. **Security and Access Control Features**:  
   Authentication and authorization systems that regulate process access to shared resources. This includes user permission checks, data encryption (where applicable), and protection against unauthorized IPC attempts.
4. **Error Handling and Recovery Protocols**:  
   Robust error detection and recovery strategies to handle communication failures gracefully. This ensures system resilience and minimizes the impact of failed transmissions or interrupted connections.

**EXCLUSION**

The exclusions from this project are features and functionalities that are beyond the current scope. These include:

1. **Cross-Network Communication Support**:  
   The IPC framework is designed for communication between processes on the same system and does not support distributed communication across networked systems or different machines.
2. **Integration with Non-Unix Platforms**:  
   While the framework is compatible with Unix-based systems (such as Linux and macOS), it does not currently provide full support or optimization for Windows or other non-Unix environments.
3. **Graphical Interface for IPC Management**:  
   The project focuses on backend functionality and does not include a GUI for configuring or visualizing IPC connections. All configuration and monitoring are done via command-line tools or logs.
4. **Machine Learning-Based Communication Optimization**:  
   The framework does not employ machine learning or AI techniques for dynamic optimization of communication patterns or predictive resource allocation.

**Who It’s For**

* Primary Users: System developers and operating system engineers who need efficient, secure inter-process communication tools.
* Secondary Users: Researchers, educators, and advanced Python developers exploring low-level system design and IPC strategies.

**Module-Wise Breakdown**

1. **Pipe Communication Module**  
   This is the direct phone line between processes. It allows two related processes to send and receive data in a linear, one-way or two-way stream. Great for simple communication with minimal setup—think of it as passing notes between friends.
2. **Message Queue System**  
   A post office for processes. This module lets processes drop messages into a queue and retrieve them asynchronously. It supports ordered delivery and allows multiple producers and consumers, perfect for systems where timing and order matter.
3. **Shared Memory Interface**  
   The communal whiteboard. Shared memory lets multiple processes access the same block of memory, making data exchange fast and efficient. With this module, processes can collaborate using shared space—ideal for high-performance systems where speed is key.
4. **Security & Access Control Module**  
   The framework’s gatekeeper. This module ensures only authorized processes can access IPC resources. It includes authentication checks, permissions, and optional data encryption—think of it as the bouncer at the club, keeping everything secure.
5. **Monitoring & Diagnostics Utility**  
   The eyes and ears of the system. It logs communication stats, detects failures, and provides real-time data on IPC traffic. Great for debugging and fine-tuning performance—like a black box recorder for process interaction.

**Functionalities**

* **Multi-Method IPC Support**  
  Supports pipes, message queues, and shared memory—each optimized for specific communication needs, so developers can choose what fits best.
* **Secure Data Exchange**  
  Built-in security protocols to ensure that only trusted processes can communicate, protecting data from unauthorized access or leaks.
* **Flexible Configuration**  
  Easily switch between IPC methods or tune settings based on system requirements. It's like having interchangeable parts in a high-performance machine.
* **Real-Time Monitoring**  
  Track process communication status and metrics in real time. Know which processes are talking, how often, and what’s being shared.
* **Error Handling & Recovery**Automatic detection and graceful handling of IPC failures. Think of it as the safety net that ensures your system doesn’t crash when something goes wrong.

**Script.js**

This is the core implementation of the data management, communication, and resource allocation. It:

* **Modular Design** :Each action (create, terminate, route, read/write shared memory) is cleanly abstracted into its own function.
* **UI Simulation Layer:** The sendMessage() function elegantly bridges user interaction with backend message simulation—smart approach for interactive demos.
* **Security Handling:** Token validation is centralized and consistently applied, which is critical in IPC-like simulations.
* **Logging System:** Real-time logs to both console and UI with severity levels. Makes debugging and educational usage intuitive.
* Peer Awareness: Workers are informed of peer changes, which reflects a true IPC environment.
* Shared Memory Sync: Every valid write updates the memory and propagates changes—this is a really faithful emulation of shared memory management.

**Worker.js**

**The worker.js file provides:**

* **Worker Initialization**: The worker is initialized with a unique workerId, a list of peer workers, and a local shared memory cache, which are provided by the main thread.
* **Message Handling**: The worker listens for messages from the main thread and processes different types.
* **Security Check**: The worker uses a simple token validation system to ensure that only authorized messages are processed.
* **Logging**: Logs are sent back to the main thread for display, which helps track the worker's actions and message processing.
* **Shared Memory Interaction**: The worker can request shared memory read or write operations via the main thread, using secure tokens for validation.
* **Peer Communication**: The worker can request the main thread to route messages to other peer workers, ensuring secure message passing between workers.
* **Autonomous Actions (Optional)**: After initialization, the worker may autonomously request shared memory reads, write key-value pairs to shared memory, or send messages to peers.
* **Error Handling**: The worker provides error handling for invalid tokens, unknown message types, and other issues, ensuring robust communication in the IPC system.
* **Modularity**: The script is designed to be modular and can be easily adapted for different IPC systems, supporting direct and broadcast communication between workers.

**README.md**

**The documentation file explains the project's purpose, functionality, and usage:**

* **Project Overview**: Introduces the concept of inter-process communication and outlines how the framework simulates IPC mechanisms (like shared memory and message queues) using Web Workers and JavaScript in a secure and interactive environment.
* **Key Features**: Showcases core functionalities including direct and broadcast messaging, shared memory emulation, token-based access control, real-time communication logs, and dynamic peer updates.
* **Getting Started**: Offers step-by-step instructions on launching the application in a browser, spawning workers, sending messages, and interacting with shared memory.
* **Educational Value**: Establishes the framework as a hands-on learning platform for students and developers to explore IPC concepts, secure message handling, and process synchronization in a visual and engaging way.

**Technology Used**

Programming Languages

* JavaScript: The primary programming language used for implementing the worker script, which handles inter-process communication (IPC) between workers and the main thread.

Libraries and Tools

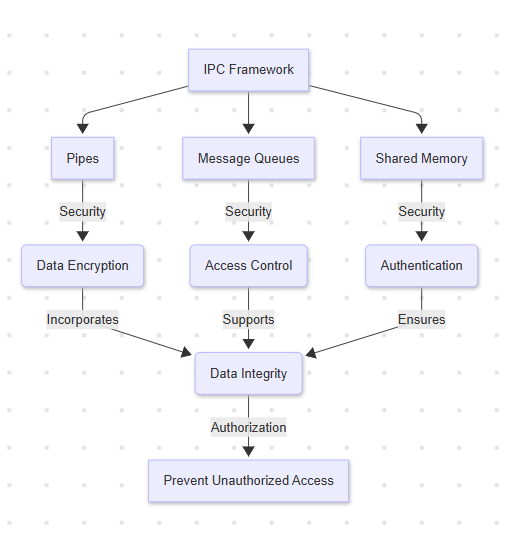
* Web Workers: A browser feature that allows for running JavaScript code in the background on a separate thread, enabling efficient parallel processing in the IPC simulation.
* Structured Cloning: A JavaScript API used for sending complex data types (like objects and arrays) between the main thread and workers, enabling safe communication without the need for serialization.
* console: Used for logging messages and debugging information during the worker's execution, helping developers track worker behavior and interactions.

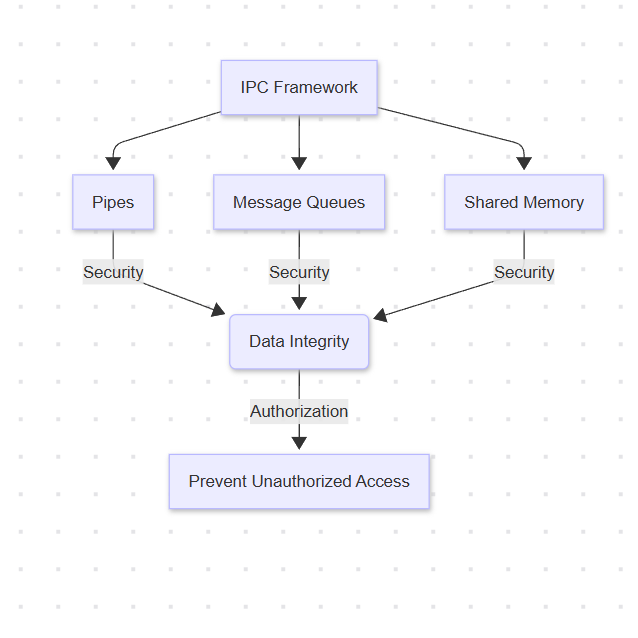
Other Tools

**GitHub**: Used for version control to manage the project's source code, track changes, and collaborate with team members.

**Flow Diagram**

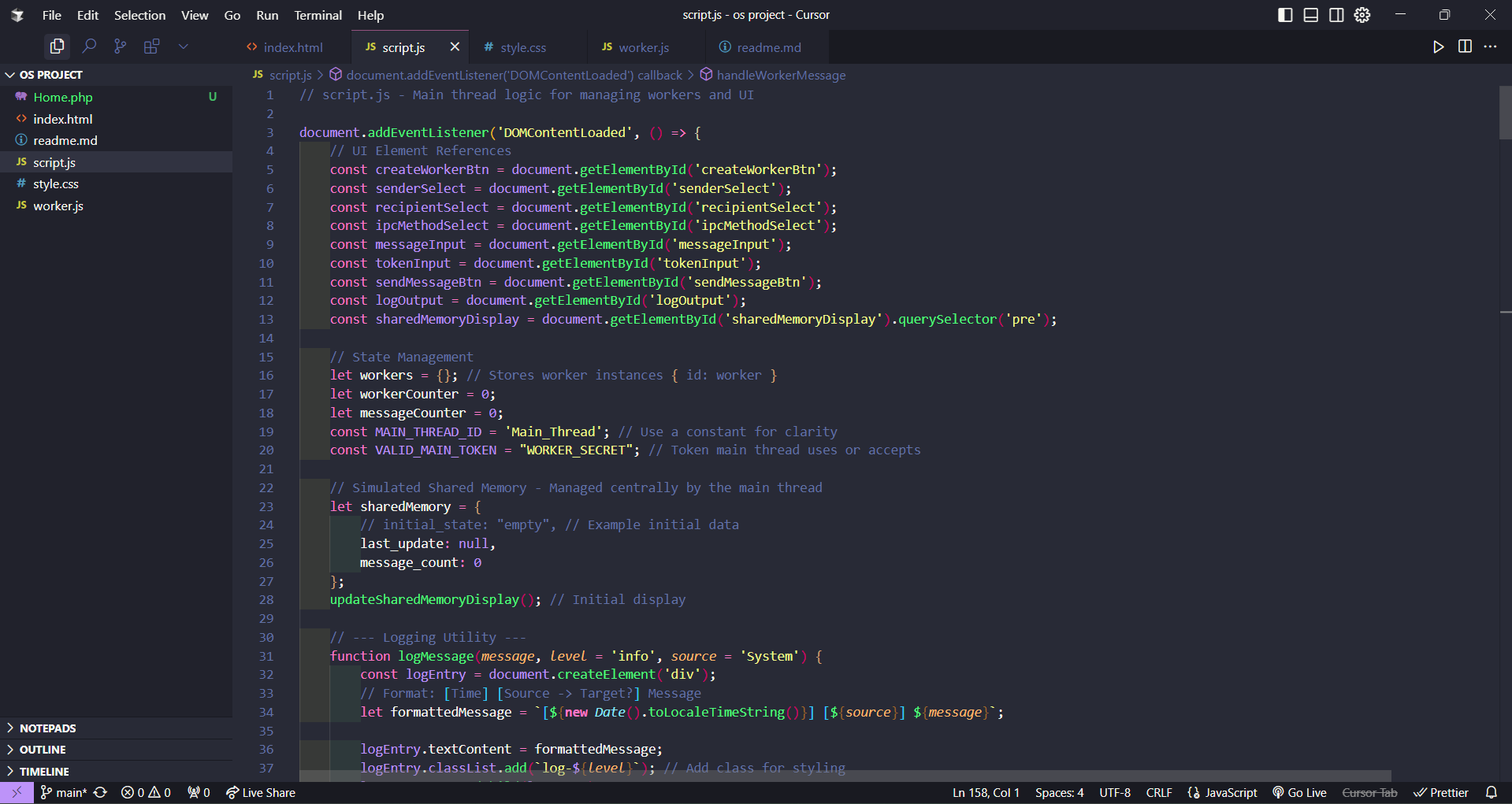
Mermaid Chart Link: <https://www.mermaidchart.com/app/projects/03dc0439-5468-4794-b286-329e61d3af67/diagrams/3b411e63-8223-41ad-8dae-260da23de571/version/v0.1/whiteboard>





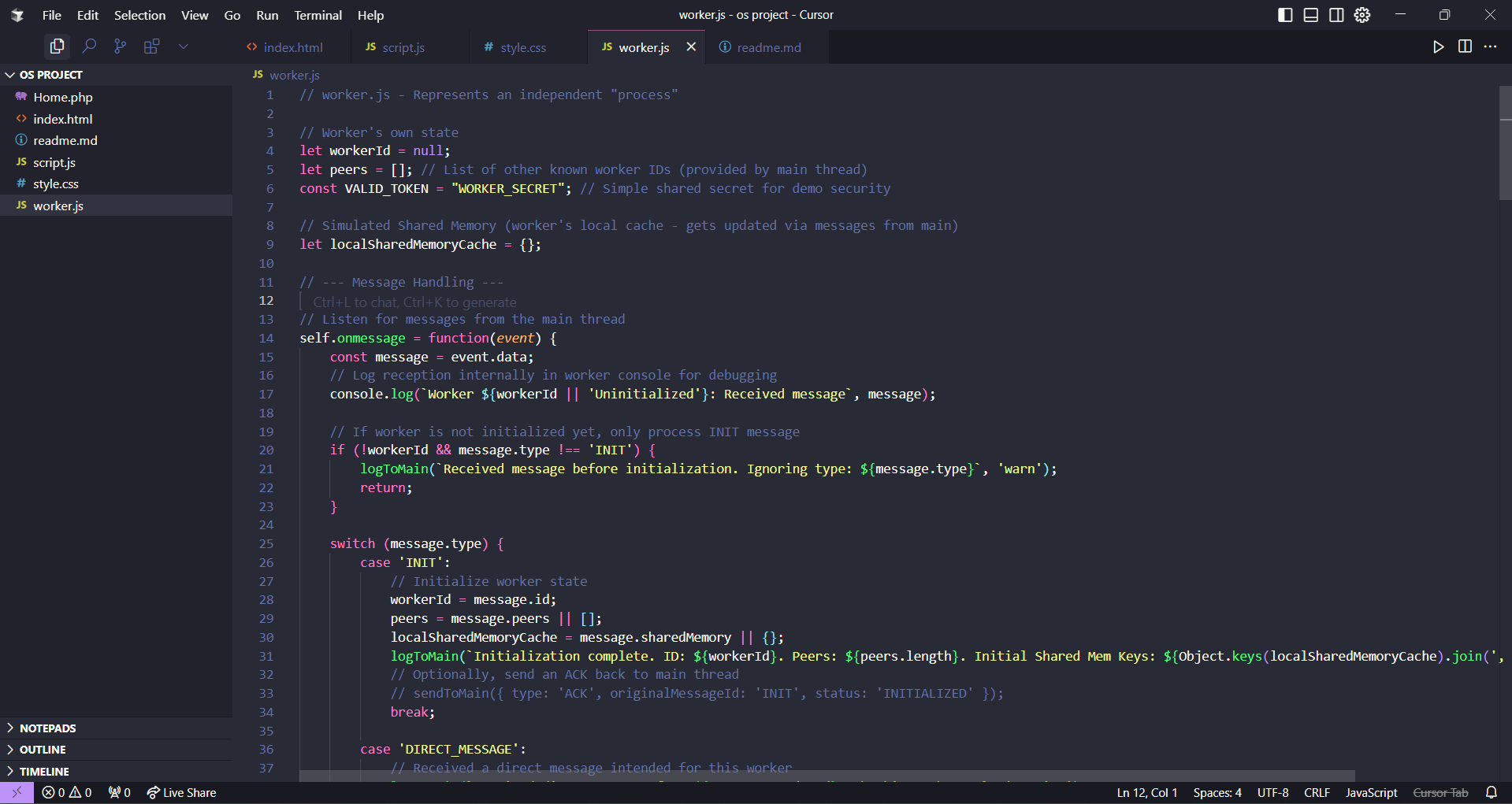
**Screenshots**

**Script.js**

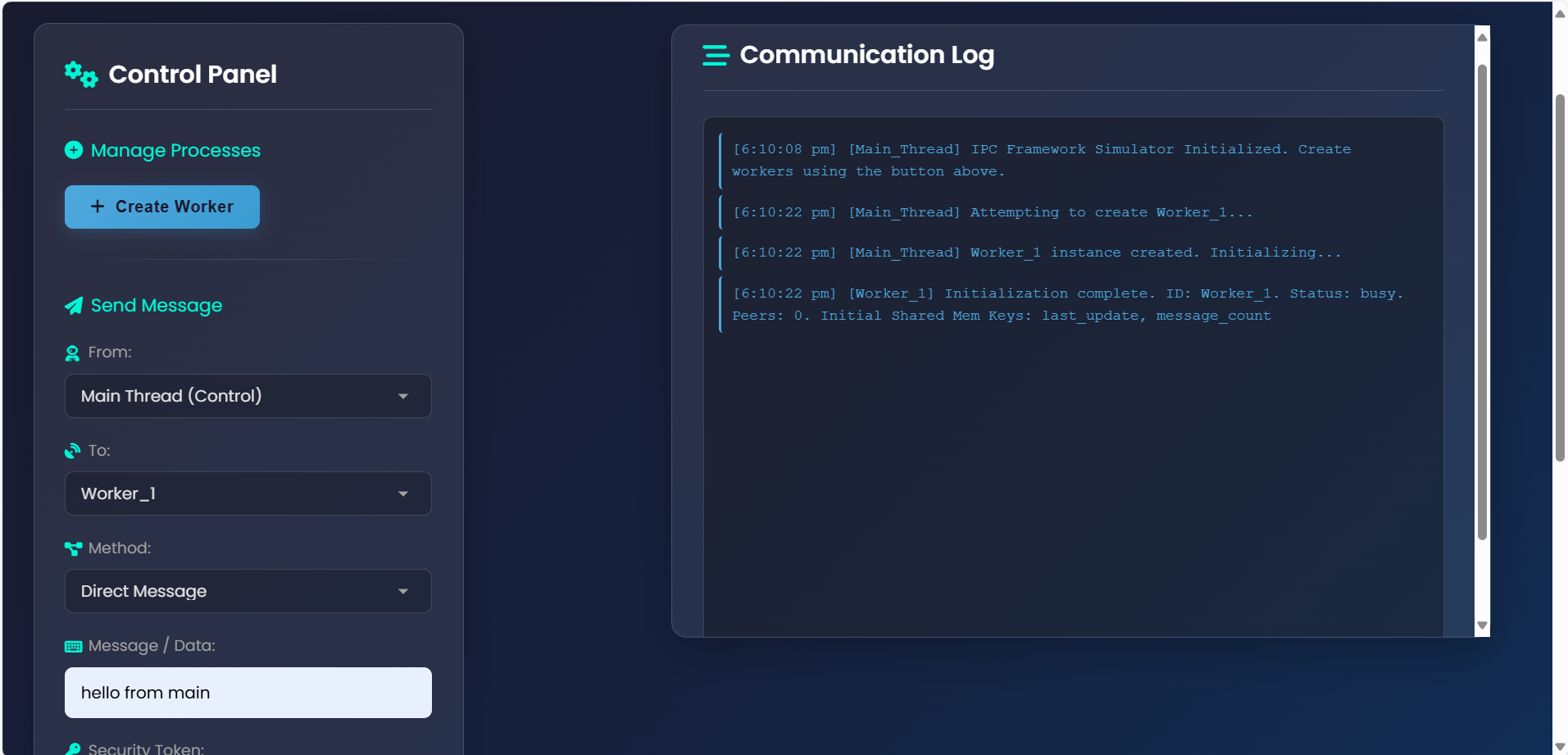


**Simple\_load\_balancer\_gui.py**

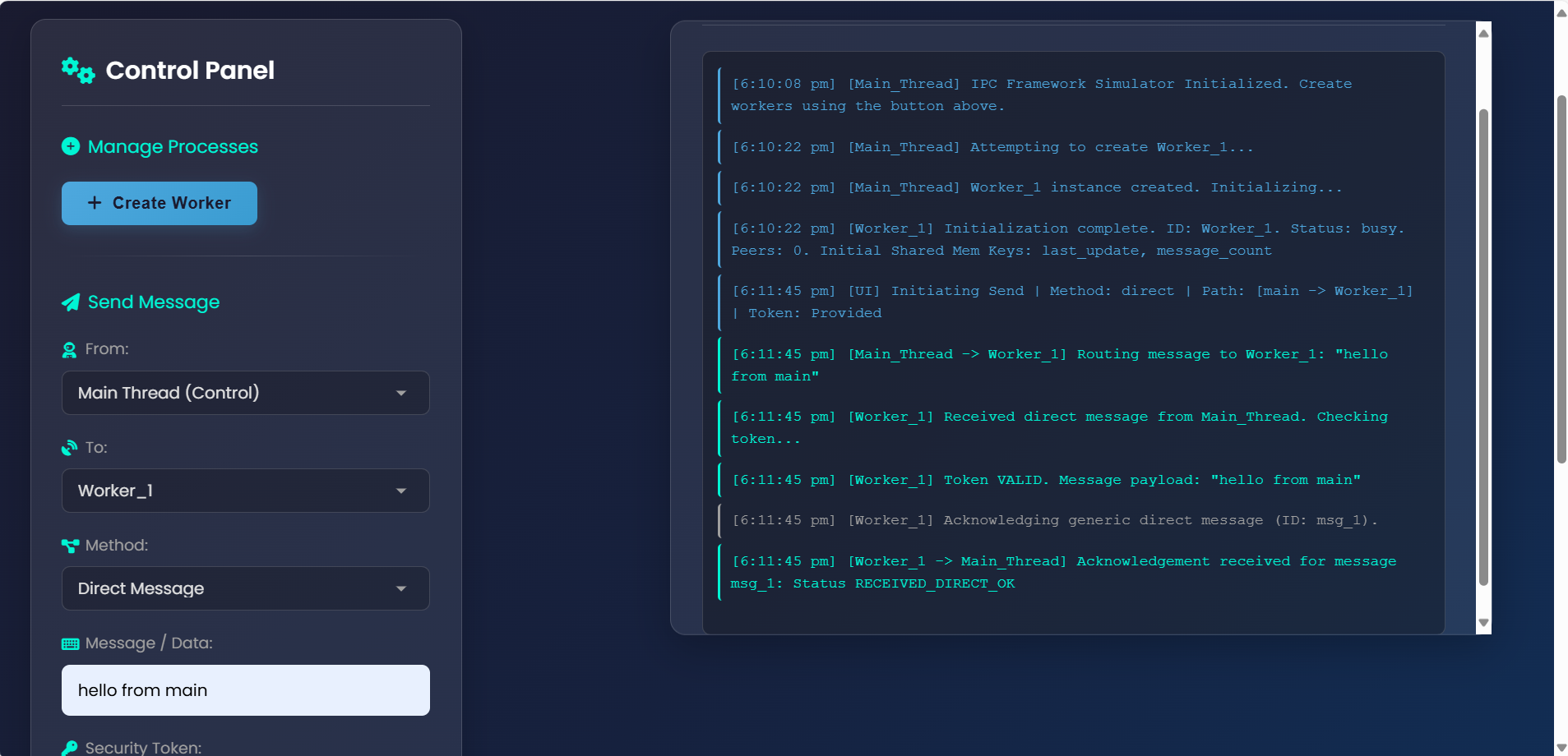
**Worker.js**



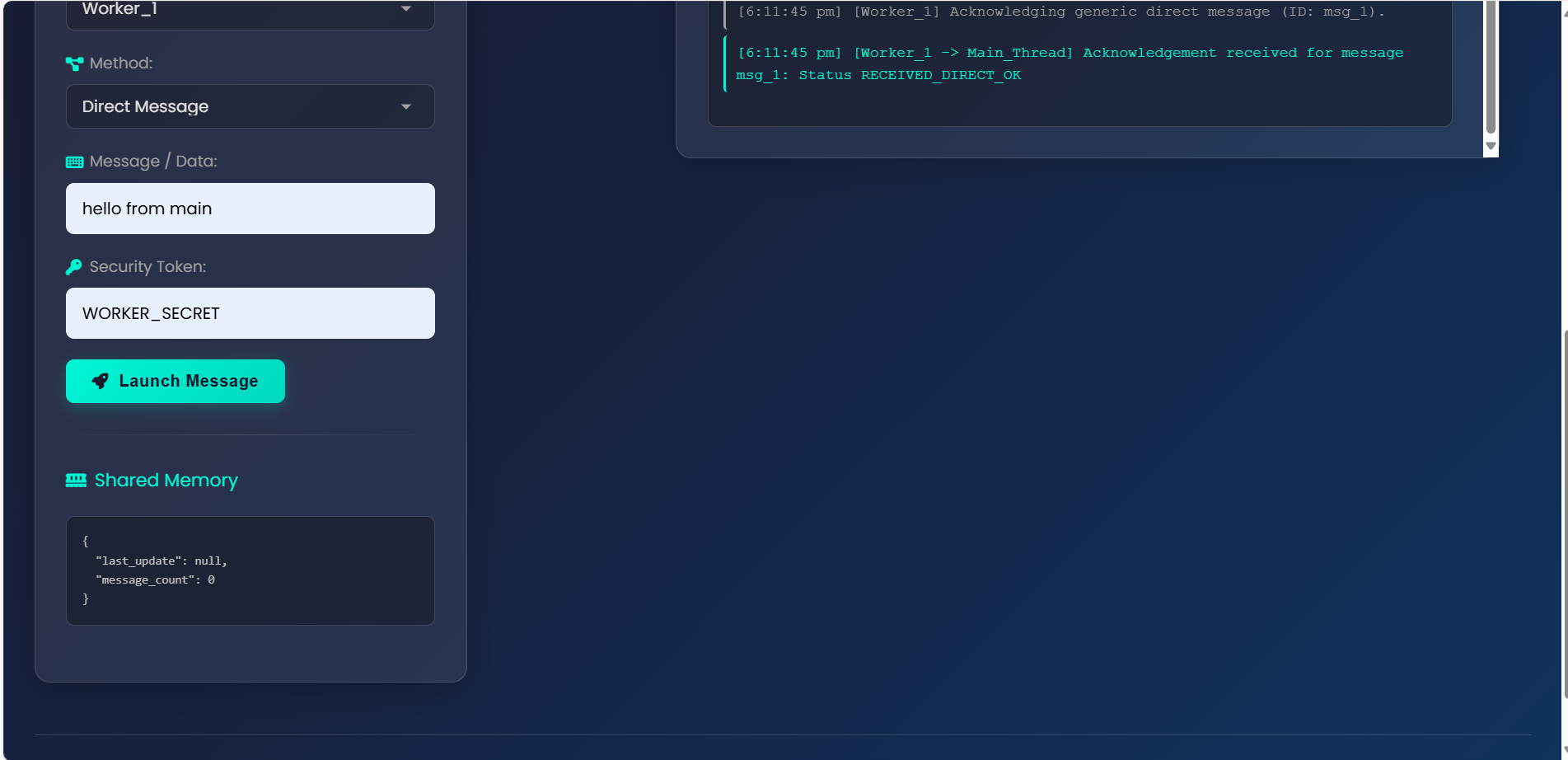
**Running**

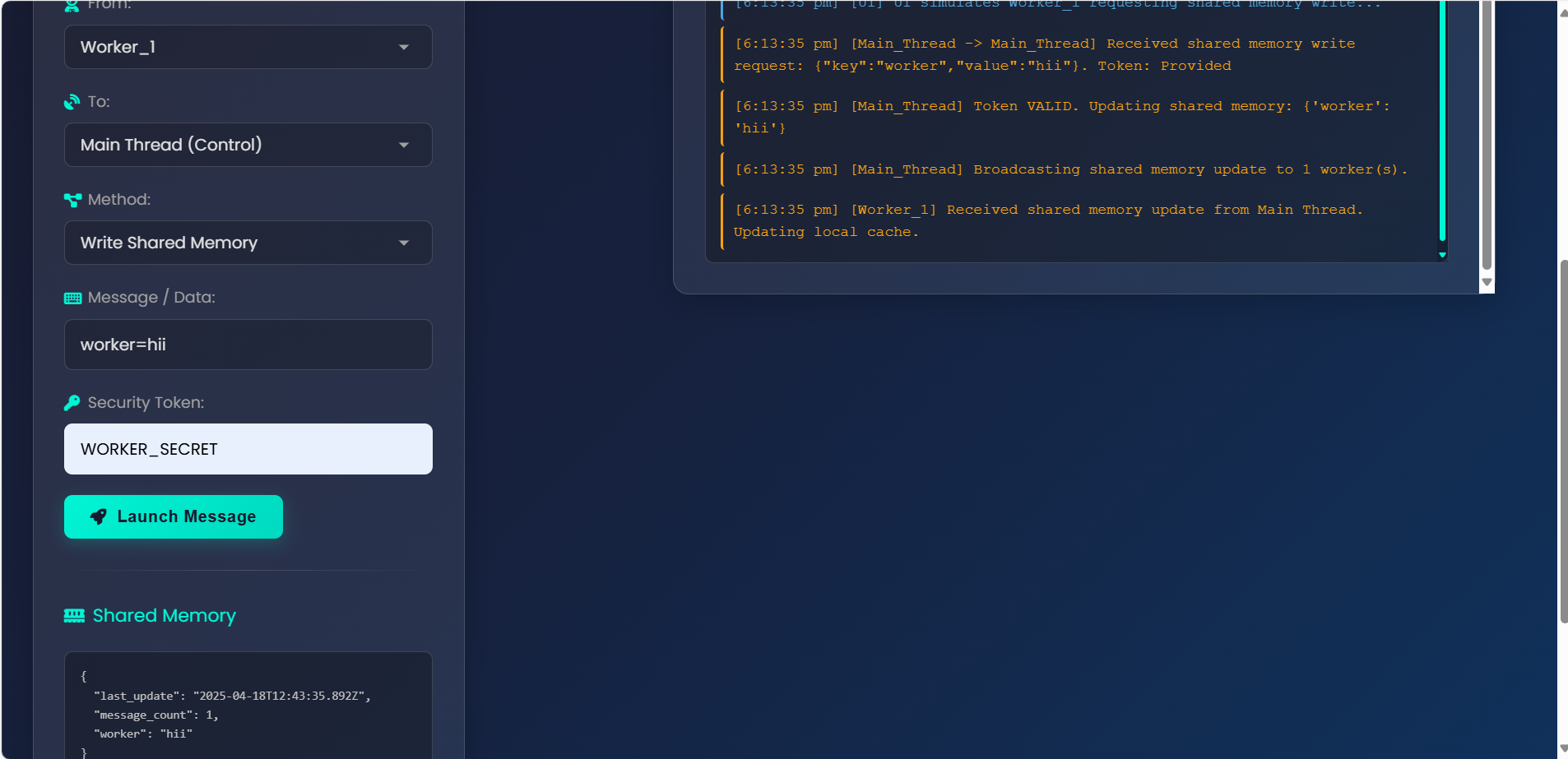


**Final Output**



**GUI Screenshot**





**Revision Tracking on GitHub**

**Anjali Kadyan**

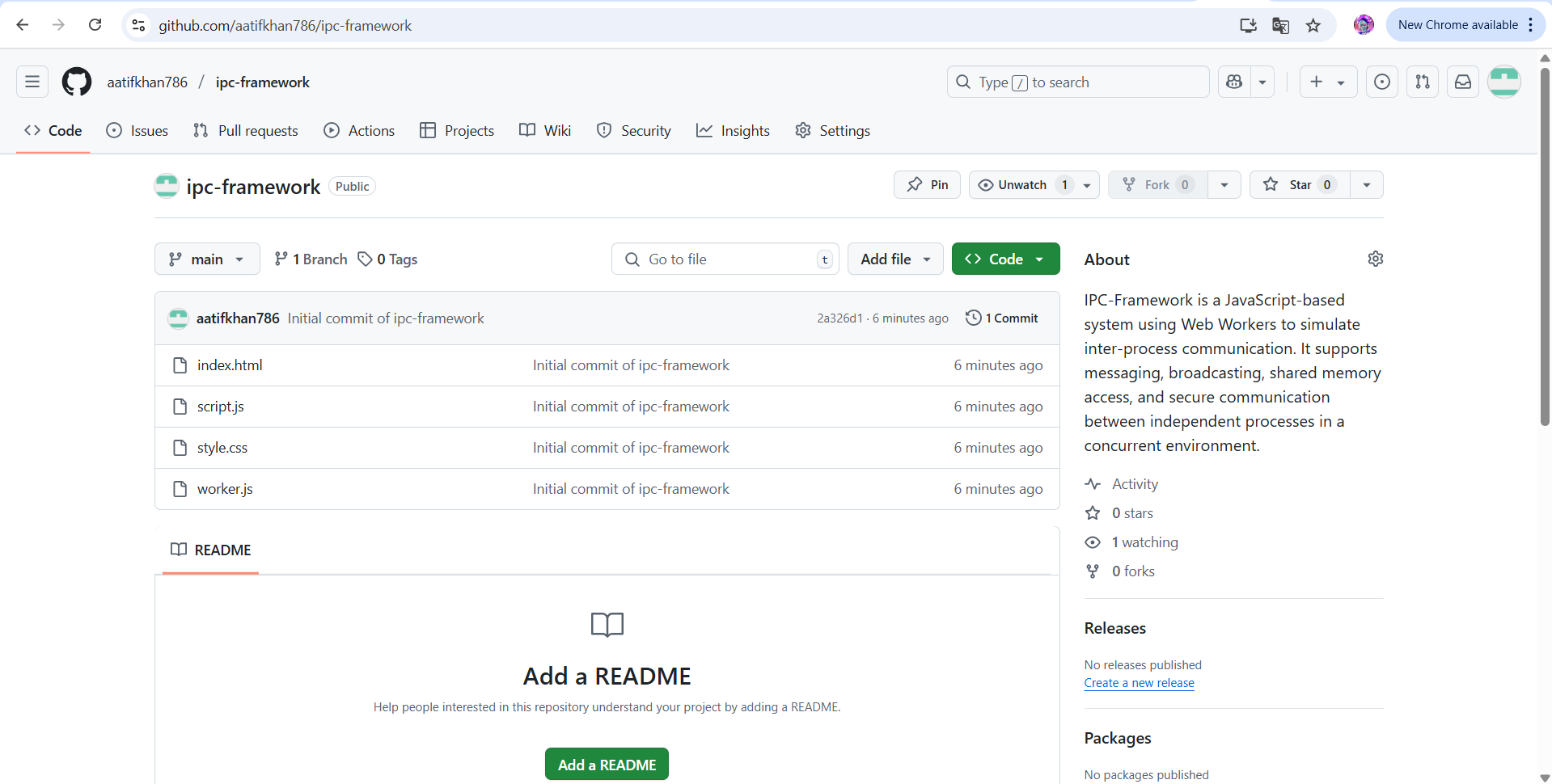
• Repository Name: **Comprehensive-Inter-Process-Communication-Framework**

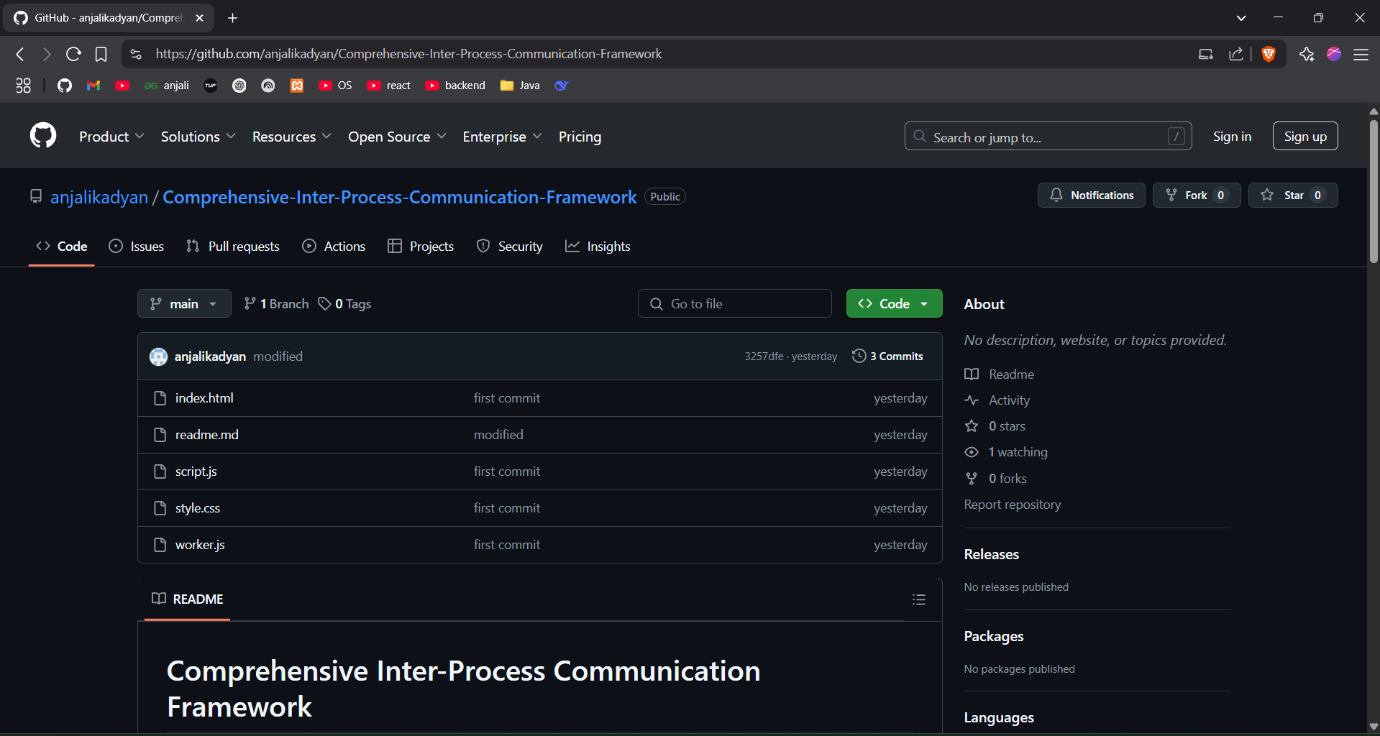
• GitHub Link: <https://github.com/anjalikadyan/Comprehensive-Inter-Process-Communication-Framework>

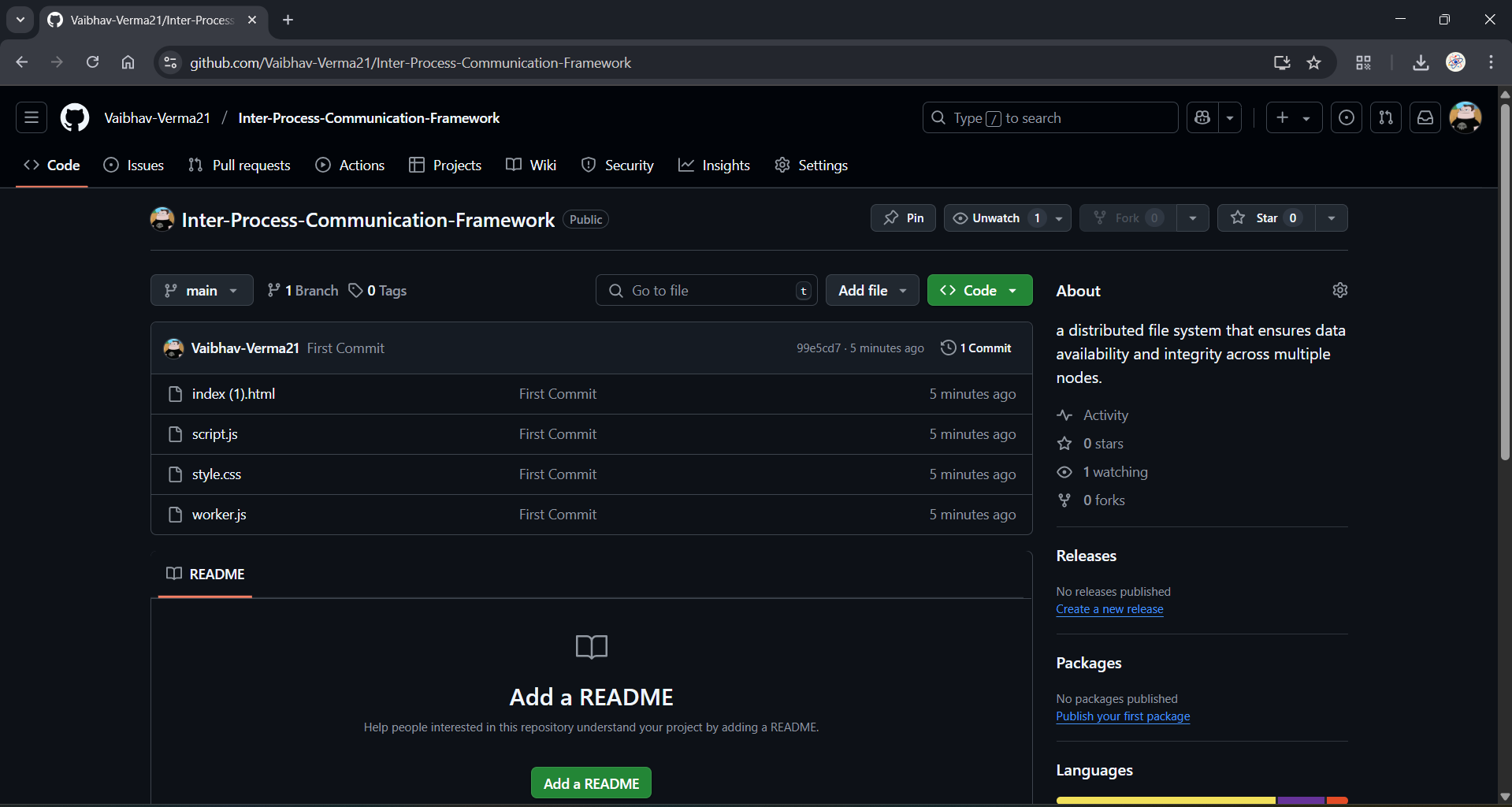
* Repository Name: ipc-framework
* GitHub Link: <https://github.com/aatifkhan786/ipc-framework>

Vaibhav Verma

* Repository Name:
* GitHub Link: <https://github.com/Vaibhav-Verma21/Inter-Process-Communication-Framework>







**Conclusion & Future Scope**

The Comprehensive Inter-Process Communication (IPC) Framework provides a robust and scalable solution for seamless communication between multiple processes, leveraging methods like pipes, message queues, and shared memory. This framework ensures the secure exchange of data between processes, incorporating token-based validation and message routing to maintain data integrity and prevent unauthorized access. By simulating a real-time IPC system, it demonstrates the power of efficient data handling in modern computing environments.

Looking ahead, there’s great potential to expand and enhance this project. One future direction could involve integrating more sophisticated encryption and access control mechanisms to further secure inter-process communication, making the system suitable for enterprise-level applications. Additionally, exploring distributed systems where multiple frameworks can communicate across networks or using advanced algorithms to optimize message routing and load balancing could improve the framework’s efficiency. The addition of a user-friendly interface or a visualization system would also make it easier for users to monitor and manage communication states across processes. The framework’s design paves the way for a wide range of applications, and it is exciting to think about how this project can evolve to support more complex, real-world computing scenarios.

**What's Next?**

* Enhanced security features for enterprise-level deployments
* Integration of distributed systems for cross-network IPC
* Advanced routing algorithms for load balancing and message optimization
* User interface for monitoring and managing IPC states
* Visualizations of message flow and performance metrics

**References**

* [psutil Documentation](https://psutil.readthedocs.io/en/latest/)
* [matplotlib Documentation](https://matplotlib.org/stable/contents.html)
* [tkinter Documentation](https://docs.python.org/3/library/tkinter.html)

**YOUTUBE CHANNELS**

ByteByteGo <https://youtu.be/dBmxNsS3BGE?si=zEWcXbBdZx5_yBxq>

5 Minutes Engineering <https://youtu.be/TO3yGao_Vjs?si=9sFlOzEXzYnJMO4Q>

Tec2Check <https://youtu.be/SdzAOanwrF0?si=CLlAIAoMiuhinfeI>

Anton Putra <https://youtu.be/gqb7LmmXuyw?si=364NdMnfvIXwf4Hl>

Concept && Coding – by Shrayansh <https://youtu.be/vJYycNWAYZU?si=O-X7JGnBOzeqd__O>

Zeto <https://youtu.be/c2zhbf1gZHc?si=b_Jtpxbp6n-AWtkV>

Zeto <https://youtu.be/XFeI_5RIAuY?si=PrBIm1nb3WKta5cu>

**Appendix**

**A. Comprehensive Inter-Process Communication (IPC) Framework Elaboration/Breakdown Report**The Comprehensive IPC Framework facilitates secure and efficient communication between multiple processes using methods like pipes, message queues, and shared memory. The framework ensures that data is transmitted securely between processes, preventing unauthorized access through token-based validation and routing mechanisms. Designed with scalability and flexibility in mind, this system simulates real-time IPC to optimize data sharing in complex computing environments.

**Key Components:**

1. Message Routing & Management: Handles the communication between processes by ensuring that messages are correctly routed and managed, including validation through tokens.
2. Shared Memory Management: Simulates shared memory where data is accessed and updated by different processes, ensuring synchronization and integrity of data.
3. Security Features: Implements token-based validation to ensure that only authorized processes can access or modify shared memory, preventing unauthorized data access.
4. Process Communication Protocols: Uses methods such as direct messages and broadcast messages to facilitate communication between processes, ensuring reliable data delivery.

**Technologies Used:**

* JavaScript (ES6+): The primary programming language used to implement the IPC framework.
* Web Workers: For simulating independent processes and their communication in a web environment.
* Shared Memory Emulation: Simulates shared memory access and synchronization in a browser-based environment.
* Token-based Access Control: For ensuring security and preventing unauthorized access.
* HTML5/JavaScript: For building the interactive UI that manages and visualizes IPC operations.

**B. Problem Statement**In modern computing systems, inter-process communication is crucial for ensuring that processes can share data and synchronize their operations efficiently. However, managing communication between multiple processes in a secure and organized way can be challenging. Inefficient IPC can lead to data corruption, unauthorized access, and synchronization issues, which degrade system performance and security. The challenge is to create a flexible and secure IPC framework that can handle communication across multiple processes while ensuring data integrity and preventing unauthorized access.

**C. Solution/Code**

Script.js

// script.js - Main thread logic for managing workers and UI

document.addEventListener('DOMContentLoaded', () => {

    // UI Element References

    const createWorkerBtn = document.getElementById('createWorkerBtn');

    const senderSelect = document.getElementById('senderSelect');

    const recipientSelect = document.getElementById('recipientSelect');

    const ipcMethodSelect = document.getElementById('ipcMethodSelect');

    const messageInput = document.getElementById('messageInput');

    const tokenInput = document.getElementById('tokenInput');

    const sendMessageBtn = document.getElementById('sendMessageBtn');

    const logOutput = document.getElementById('logOutput');

    const sharedMemoryDisplay = document.getElementById('sharedMemoryDisplay').querySelector('pre');

    // State Management

    let workers = {}; // Stores worker instances { id: worker }

    let workerCounter = 0;

    let messageCounter = 0;

    const MAIN\_THREAD\_ID = 'Main\_Thread'; // Use a constant for clarity

    const VALID\_MAIN\_TOKEN = "WORKER\_SECRET"; // Token main thread uses or accepts

    // Simulated Shared Memory - Managed centrally by the main thread

    let sharedMemory = {

        // initial\_state: "empty", // Example initial data

        last\_update: null,

        message\_count: 0

    };

    updateSharedMemoryDisplay(); // Initial display

    // --- Logging Utility ---

    function logMessage(*message*, *level* = 'info', *source* = 'System') {

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

        // Format: [Time] [Source -> Target?] Message

        let formattedMessage = `[${**new** Date().toLocaleTimeString()}] [${source}] ${message}`;

        logEntry.textContent = formattedMessage;

        logEntry.classList.add(`log-${level}`); // Add class for styling

        logOutput.appendChild(logEntry);

        // Auto-scroll to the bottom

        logOutput.scrollTop = logOutput.scrollHeight;

        // Also log to console for debugging

        console.log(`[${level.toUpperCase()}] [${source}] ${message}`);

    }

    // --- Worker Management ---

    function createWorker() {

        workerCounter++;

        const workerId = `Worker\_${workerCounter}`;

        logMessage(`Attempting to create ${workerId}...`, 'system', MAIN\_THREAD\_ID);

        try {

            const worker = **new** Worker('worker.js');

            workers[workerId] = worker;

            // Add worker to UI selectors

            addWorkerToSelect(senderSelect, workerId);

            addWorkerToSelect(recipientSelect, workerId);

            // Handle messages \*FROM\* the worker

            worker.onmessage = (*event*) => handleWorkerMessage(workerId, event.data);

            worker.onerror = (*error*) => {

                logMessage(`Critical error in ${workerId}: ${error.message} at ${error.filename}:${error.lineno}. Terminating worker.`, 'error', workerId);

                terminateWorker(workerId); // Terminate on critical error

            };

            // Initialize the worker: send its ID, current peers, and shared memory state

            const peerIds = Object.keys(workers).filter(*id* => id !== workerId);

            worker.postMessage({

                type: 'INIT',

                id: workerId,

                peers: peerIds,

                sharedMemory: sharedMemory, // Send initial shared memory state

                // No token needed for INIT as it's a system setup message

            });

            logMessage(`${workerId} instance created. Initializing...`, 'system', MAIN\_THREAD\_ID);

            // Inform \*existing\* workers about the new peer (if any exist)

            const allWorkerIds = Object.keys(workers);

            if (allWorkerIds.length > 1) {

                broadcastSystemUpdate({ type: 'PEER\_UPDATE', peers: allWorkerIds });

                logMessage(`Notified other workers about new peer: ${workerId}`, 'system', MAIN\_THREAD\_ID);

            }

        } catch (e) {

             logMessage(`Failed to create ${workerId}: ${e.message}`, 'error', MAIN\_THREAD\_ID);

             workerCounter--; // Rollback counter if creation failed

        }

    }

    // Helper to add worker options to dropdowns

    function addWorkerToSelect(*selectElement*, *workerId*) {

        const option = document.createElement('option');

        option.value = workerId;

        option.textContent = workerId;

        selectElement.appendChild(option);

    }

    // Helper to remove worker options from dropdowns

    function removeWorkerFromSelect(*selectElement*, *workerId*) {

         const options = *selectElement*.options;

         for (let i = options.length - 1; i >= 0; i--) { // Iterate backwards when removing

             if (options[i].value === *workerId*) {

*selectElement*.remove(i);

                 break;

             }

         }

    }

    // Function to terminate a worker (e.g., on error or manual request)

    function terminateWorker(*workerId*) {

        if (workers[*workerId*]) {

            logMessage(`Terminating ${*workerId*}...`, 'system', MAIN\_THREAD\_ID);

            workers[*workerId*].terminate();

            delete workers[*workerId*];

            removeWorkerFromSelect(senderSelect, *workerId*);

            removeWorkerFromSelect(recipientSelect, *workerId*);

            logMessage(`${*workerId*} terminated successfully.`, 'system', MAIN\_THREAD\_ID);

            // Inform remaining workers about the removed peer

            broadcastSystemUpdate({ type: 'PEER\_UPDATE', peers: *Object*.keys(workers) });

        } else {

             logMessage(`Attempted to terminate non-existent worker: ${*workerId*}`, 'error', MAIN\_THREAD\_ID);

        }

    }

    // --- Message Handling ---

    /\*\*

     \* Handles messages received \*FROM\* workers.

     \* @param *{string}* *workerId* - The ID of the worker sending the message.

     \* @param *{object}* *message* - The message object received from the worker.

     \*/

    function handleWorkerMessage(*workerId*, *message*) {

        console.log(`Main: Received from ${*workerId*}`, *message*); // Keep raw console log

        switch (*message*.type) {

            case 'LOG':

                // Display log messages sent from workers

                logMessage(*message*.payload, *message*.level || 'info', *workerId*);

                break;

            case 'ACK':

                 // Acknowledge received message (optional)

                 logMessage(`Acknowledgement received for message ${*message*.originalMessageId}: Status ${*message*.status}`, 'receive', `${*workerId*} -> ${MAIN\_THREAD\_ID}`);

                break;

            case 'ERROR\_REPORT': // More specific error type from worker

                 logMessage(`Worker reported error: ${*message*.error} (Regarding: ${*message*.context || 'N/A'})`, 'error', *workerId*);

                 break;

            case 'ROUTE\_MESSAGE\_REQUEST':

                // Worker wants the main thread to route a message to another worker

                logMessage(`Received request to route message from ${*workerId*} to ${*message*.recipientId}`, 'system', `${*workerId*} -> ${MAIN\_THREAD\_ID}`);

                handleDirectMessage(*workerId*, *message*.recipientId, *message*.payload, *message*.token, `routed\_${messageCounter++}`);

                break;

            case 'SHARED\_MEM\_READ\_REQUEST\_FROM\_WORKER':

                 // Worker requests the current state of shared memory

                 logMessage(`Received shared memory read request. Token: ${*message*.token ? 'Provided' : 'Missing'}`, 'shared', `${*workerId*} -> ${MAIN\_THREAD\_ID}`);

                 if (isValidToken(*message*.token)) {

                     logMessage(`Token VALID. Sending current shared memory state.`, 'shared', `${MAIN\_THREAD\_ID} -> ${*workerId*}`);

                     workers[*workerId*]?.postMessage({

                         type: 'SHARED\_MEM\_UPDATE', // Use update type to refresh worker's cache

                         payload: sharedMemory,

                         token: 'MAIN\_INTERNAL\_TOKEN' // Use a distinct token for internal comms if needed

                     });

                 } else {

                      logMessage(`Token INVALID. Denying shared memory read request.`, 'error', `${MAIN\_THREAD\_ID} -> ${*workerId*}`);

                      workers[*workerId*]?.postMessage({ type: 'ERROR\_REPORT', context: 'Shared Mem Read', error: 'Invalid Token' });

                 }

                break;

             case 'SHARED\_MEM\_WRITE\_REQUEST\_FROM\_WORKER':

                 // Worker requests to write to shared memory

                 logMessage(`Received shared memory write request: ${JSON.stringify(*message*.payload)}. Token: ${*message*.token ? 'Provided' : 'Missing'}`, 'shared', `${*workerId*} -> ${MAIN\_THREAD\_ID}`);

                 if (isValidToken(*message*.token)) {

                     const { key, value } = *message*.payload;

                     if (key !== undefined && value !== undefined) {

                         logMessage(`Token VALID. Updating shared memory: {'${key}': '${value}'}`, 'shared', MAIN\_THREAD\_ID);

                         // Update the central shared memory

                         sharedMemory[key] = value;

                         sharedMemory.last\_update = **new** *Date*().toISOString();

                         sharedMemory.message\_count++;

                         updateSharedMemoryDisplay();

                         // Broadcast the update to ALL workers (including the sender) to sync their caches

                         broadcastSharedMemoryUpdate();

                         // Optionally send ACK back to requester

                         workers[*workerId*]?.postMessage({ type: 'ACK', originalMessageId: *message*.id, status: 'SHARED\_WRITE\_SUCCESS' });

                     } else {

                          logMessage(`Invalid payload format for write request from ${*workerId*}. Key or value missing.`, 'error', MAIN\_THREAD\_ID);

                          workers[*workerId*]?.postMessage({ type: 'ERROR\_REPORT', context: 'Shared Mem Write', error: 'Invalid Payload Format' });

                     }

                 } else {

                     logMessage(`Token INVALID. Denying shared memory write request.`, 'error', `${MAIN\_THREAD\_ID} -> ${*workerId*}`);

                     workers[*workerId*]?.postMessage({ type: 'ERROR\_REPORT', context: 'Shared Mem Write', error: 'Invalid Token' });

                 }

                 break;

            default:

                logMessage(`Received unknown message type '${*message*.type}' from worker.`, 'warn', *workerId*);

        }

    }

    /\*\*

     \* Initiates sending a message based on UI selections.

     \*/

    function sendMessage() {

        const senderId = senderSelect.value;

        const recipientId = recipientSelect.value;

        const ipcMethod = ipcMethodSelect.value;

        const payload = messageInput.value;

        const token = tokenInput.value;

        // Basic validation

        if (!payload && (ipcMethod === 'direct' || ipcMethod === 'shared\_write')) {

            logMessage("Input required: Please enter a message or data for 'Direct Message' or 'Write Shared Memory'.", 'error', 'UI');

            return;

        }

         if (!token) {

             logMessage("Input required: Please enter a security token.", 'error', 'UI');

             return;

         }

        messageCounter++;

        const messageId = `msg\_${messageCounter}`;

        logMessage(`Initiating Send | Method: ${ipcMethod} | Path: [${senderId} -> ${recipientId}] | Token: ${token ? 'Provided' : 'None'}`, 'system', 'UI');

        // --- Handle different IPC Methods ---

        if (senderId === 'main') {

            // Actions initiated directly from the Main Thread UI

            switch (ipcMethod) {

                case 'direct':

                    handleDirectMessage(MAIN\_THREAD\_ID, recipientId, payload, token, messageId);

                    break;

                case 'shared\_read':

                    handleSharedRead(MAIN\_THREAD\_ID, token); // Recipient doesn't matter for main reading its own memory

                    break;

                case 'shared\_write':

                    handleSharedWrite(MAIN\_THREAD\_ID, payload, token);

                    break;

                default:

                    logMessage(`Unknown IPC method selected: ${ipcMethod}`, 'error', 'UI');

            }

        } else {

            // Simulate a worker initiating an action via the UI

            logMessage(`Simulating action from ${senderId} via UI...`, 'system', 'UI');

            // Currently, workers initiate actions from their own code.

            // To truly simulate via UI, we'd need to send a message \*to\* the source worker

            // telling it to perform the action, which adds complexity.

            // Let's simplify: For UI actions \*selected\* as 'From Worker', we'll \*route\* them via main.

            switch (ipcMethod) {

                case 'direct':

                     // Route message via main thread

                    logMessage(`UI simulates ${senderId} sending direct message. Routing via Main Thread...`, 'system', 'UI');

                    handleDirectMessage(senderId, recipientId, payload, token, messageId);

                    break;

                 case 'shared\_read':

                     // Simulate worker requesting read (Main thread handles check)

                     logMessage(`UI simulates ${senderId} requesting shared memory read...`, 'system', 'UI');

                     // Send request to main thread's handler, mimicking worker request

                     handleWorkerMessage(MAIN\_THREAD\_ID, { // Mock message as if from worker

                         type: 'SHARED\_MEM\_READ\_REQUEST\_FROM\_WORKER',

                         token: token,

                         workerId: senderId // Indicate the intended source worker

                     });

                     break;

                 case 'shared\_write':

                      // Simulate worker requesting write (Main thread handles check)

                      logMessage(`UI simulates ${senderId} requesting shared memory write...`, 'system', 'UI');

                      // Parse payload for key-value

                       let key, value;

                       if (payload.includes('=')) {

                           [key, value] = payload.split('=', 2);

                           key = key.trim();

                           value = value.trim();

                       } else {

                           logMessage(`Invalid format for Shared Write from UI (expected key=value): "${payload}"`, 'error', 'UI');

                           return; // Stop if format is wrong

                       }

                       // Send request to main thread's handler

                       handleWorkerMessage(MAIN\_THREAD\_ID, { // Mock message

                            type: 'SHARED\_MEM\_WRITE\_REQUEST\_FROM\_WORKER',

                            payload: { key: key, value: value },

                            token: token,

                            workerId: senderId, // Indicate source

                            id: messageId // Include message ID if needed

                        });

                       break;

                default:

                    logMessage(`Unknown IPC method selected for worker simulation: ${ipcMethod}`, 'error', 'UI');

            }

        }

    }

    /\*\*

     \* Handles sending/routing direct messages or broadcasts.

     \* @param *{string}* *actualSenderId* - Who is actually sending (Main\_Thread or a Worker\_X ID)

     \* @param *{string}* *recipientId* - Target ('broadcast', 'Main\_Thread', or a Worker\_X ID)

     \* @param *{string}* *payload* - The message content.

     \* @param *{string}* *token* - Security token.

     \* @param *{string}* *messageId* - Unique ID for the message.

     \*/

    function handleDirectMessage(*actualSenderId*, *recipientId*, *payload*, *token*, *messageId*) {

        const message = {

            id: messageId,

            // Type depends on if it's broadcast or direct

            type: recipientId === 'broadcast' ? 'BROADCAST\_MESSAGE' : 'DIRECT\_MESSAGE',

            senderId: actualSenderId, // Mark who originated the message

            payload: payload,

            token: token

        };

        if (recipientId === 'broadcast') {

            logMessage(`Broadcasting message: "${payload}"`, 'send', `${actualSenderId} -> ALL\_WORKERS`);

            broadcastMessageToWorkers(message); // Use specific broadcast function

        } else if (recipientId === 'main' || recipientId === MAIN\_THREAD\_ID) {

             // Message intended for the main thread itself

             logMessage(`Received message intended for Main Thread: "${payload}". Token: ${token ? 'VALID' : 'INVALID'}`, 'receive', `${actualSenderId} -> ${MAIN\_THREAD\_ID}`);

             if (!isValidToken(token)) {

                  logMessage(`Received message for Main Thread with INVALID TOKEN from ${actualSenderId}. Ignoring payload.`, 'error', MAIN\_THREAD\_ID);

             }

             // TODO: Add actual processing logic for messages to main thread if needed

        } else if (workers[recipientId]) {

            // Route to a specific worker

            logMessage(`Routing message to ${recipientId}: "${payload}"`, 'send', `${actualSenderId} -> ${recipientId}`);

            workers[recipientId].postMessage(message);

        } else {

            logMessage(`Send failed: Recipient process '${recipientId}' not found or not running.`, 'error', actualSenderId);

        }

    }

    /\*\*

     \* Handles request to read shared memory (initiated by Main Thread UI)

     \*/

    function handleSharedRead(*requesterId*, *token*) {

         logMessage(`Processing Shared Memory Read request. Token: ${token ? 'Provided' : 'Missing'}`, 'shared', requesterId);

         if (!isValidToken(token)) {

            logMessage(`Shared Memory Read Request DENIED - Invalid Token.`, 'error', requesterId);

            return;

         }

         // If main thread requests read, it already has the data. Just log it.

         logMessage(`Read successful. Current Shared Memory: ${JSON.stringify(sharedMemory)}`, 'shared', requesterId);

         // No need to send message back to self (main thread)

    }

     /\*\*

      \* Handles request to write shared memory (initiated by Main Thread UI)

      \*/

     function handleSharedWrite(*requesterId*, *payload*, *token*) {

        logMessage(`Processing Shared Memory Write request. Payload: "${payload}". Token: ${token ? 'Provided' : 'Missing'}`, 'shared', requesterId);

         if (!isValidToken(token)) {

            logMessage(`Shared Memory Write Request DENIED - Invalid Token.`, 'error', requesterId);

            return;

         }

         // Parse "key=value" from payload

         let key, value;

         if (payload.includes('=')) {

             [key, value] = payload.split('=', 2);

             key = key.trim();

             value = value.trim();

         } else {

             logMessage(`Invalid format for shared memory write payload: "${payload}". Use "key=value".`, 'error', requesterId);

             return; // Stop processing if format is wrong

         }

         if (key) {

             logMessage(`Token VALID. Updating Shared Memory: {'${key}': '${value}'}`, 'shared', requesterId);

             sharedMemory[key] = value;

             sharedMemory.last\_update = **new** Date().toISOString();

             sharedMemory.message\_count++;

             updateSharedMemoryDisplay();

             // Broadcast the update to all workers

             broadcastSharedMemoryUpdate();

         }

         // else: Key parsing failed, error already logged

     }

    // --- Broadcasting Helpers ---

    /\*\* Broadcasts a generic message (like direct/broadcast) to all workers \*/

    function broadcastMessageToWorkers(*message*) {

        if (Object.keys(workers).length === 0) {

            logMessage("Broadcast failed: No active workers found.", 'warn', message.senderId || MAIN\_THREAD\_ID);

            return;

        }

        logMessage(`Broadcasting message type '${message.type}' to ${Object.keys(workers).length} worker(s).`, 'system', message.senderId || MAIN\_THREAD\_ID);

        Object.values(workers).forEach(*worker* => {

            // Send a structured clone of the message

             worker.postMessage(JSON.parse(JSON.stringify(message)));

        });

    }

    /\*\* Broadcasts shared memory updates specifically \*/

    function broadcastSharedMemoryUpdate() {

        const updateMessage = {

             type: 'SHARED\_MEM\_UPDATE',

             payload: sharedMemory,

             token: 'MAIN\_INTERNAL\_TOKEN' // Use a distinct token for internal comms

        };

         if (Object.keys(workers).length === 0) {

            logMessage("Shared memory updated, but no workers to notify.", 'info', MAIN\_THREAD\_ID);

            return;

        }

        logMessage(`Broadcasting shared memory update to ${Object.keys(workers).length} worker(s).`, 'shared', MAIN\_THREAD\_ID);

        Object.values(workers).forEach(*worker* => {

             worker.postMessage(JSON.parse(JSON.stringify(updateMessage)));

        });

    }

     /\*\* Broadcasts system updates (like peer changes) \*/

    function broadcastSystemUpdate(*message*) {

         // Add type if missing, maybe token if needed for system messages

         message.token = 'MAIN\_INTERNAL\_TOKEN'; // Assuming system updates are trusted

          if (Object.keys(workers).length === 0) return; // No one to notify

          logMessage(`Broadcasting system update type '${message.type}' to ${Object.keys(workers).length} worker(s).`, 'system', MAIN\_THREAD\_ID);

         Object.values(workers).forEach(*worker* => {

              worker.postMessage(JSON.parse(JSON.stringify(message)));

         });

    }

    // --- UI and Utility Helpers ---

    // Update the display of the shared memory object

    function updateSharedMemoryDisplay() {

        sharedMemoryDisplay.textContent = JSON.stringify(sharedMemory, null, 2); // Pretty print JSON

    }

    // Basic token validation (mirrors worker's logic for simplicity)

    function isValidToken(*token*) {

        // In a real app, main thread might have different/more complex validation

        return token === VALID\_MAIN\_TOKEN;

    }

    // --- Event Listeners ---

    createWorkerBtn.addEventListener('click', createWorker);

    sendMessageBtn.addEventListener('click', sendMessage);

    // --- Initial Log ---

    logMessage("IPC Framework Simulator Initialized. Create workers using the button above.", 'system', MAIN\_THREAD\_ID);

});

Worker.js

// worker.js - Represents an independent "process"

// Worker's own state

let workerId = null;

let peers = []; // List of other known worker IDs (provided by main thread)

const VALID\_TOKEN = "WORKER\_SECRET"; // Simple shared secret for demo security

// Simulated Shared Memory (worker's local cache - gets updated via messages from main)

let localSharedMemoryCache = {};

// --- Message Handling ---

// Listen for messages from the main thread

self.onmessage = function(*event*) {

    const message = *event*.data;

    // Log reception internally in worker console for debugging

    console.log(`Worker ${workerId || 'Uninitialized'}: Received message`, message);

    // If worker is not initialized yet, only process INIT message

    if (!workerId && message.type !== 'INIT') {

        logToMain(`Received message before initialization. Ignoring type: ${message.type}`, 'warn');

        return;

    }

    switch (message.type) {

        case 'INIT':

            // Initialize worker state

            workerId = message.id;

            peers = message.peers || [];

            localSharedMemoryCache = message.sharedMemory || {};

            logToMain(`Initialization complete. ID: ${workerId}. Peers: ${peers.length}. Initial Shared Mem Keys: ${*Object*.keys(localSharedMemoryCache).join(', ')}`, 'system');

            // Optionally, send an ACK back to main thread

            // sendToMain({ type: 'ACK', originalMessageId: 'INIT', status: 'INITIALIZED' });

            break;

        case 'DIRECT\_MESSAGE':

            // Received a direct message intended for this worker

            logToMain(`Received direct message from ${message.senderId}. Checking token...`, 'receive');

            if (isValidToken(message.token)) {

                logToMain(`Token VALID. Message payload: "${message.payload}"`, 'receive');

                // Process the payload here if needed

                // Example: Send an acknowledgement back

                sendToMain({ type: 'ACK', originalMessageId: message.id, status: 'RECEIVED\_DIRECT\_OK' });

            } else {

                logToMain(`Received direct message with INVALID TOKEN from ${message.senderId}. Message ignored.`, 'error');

                // Report error back

                sendToMain({ type: 'ERROR\_REPORT', context: `Direct Msg ${message.id}`, error: 'Invalid Token Received' });

            }

            break;

        case 'BROADCAST\_MESSAGE':

             // Received a broadcast message

             logToMain(`Received broadcast message from ${message.senderId}. Checking token...`, 'receive');

             if (isValidToken(message.token)) {

                logToMain(`Token VALID. Broadcast payload: "${message.payload}"`, 'receive');

                // Process broadcast payload

                sendToMain({ type: 'ACK', originalMessageId: message.id, status: 'RECEIVED\_BROADCAST\_OK' });

             } else {

                 logToMain(`Received broadcast message with INVALID TOKEN from ${message.senderId}. Broadcast ignored.`, 'error');

                 sendToMain({ type: 'ERROR\_REPORT', context: `Broadcast Msg ${message.id}`, error: 'Invalid Token Received' });

             }

            break;

        case 'SHARED\_MEM\_UPDATE':

            // Main thread sent an update to the shared memory state

            // Note: We might trust internal updates more, or still check token

            // if (isValidToken(message.token)) { // Optional token check for updates

                logToMain(`Received shared memory update from Main Thread. Updating local cache.`, 'shared');

                localSharedMemoryCache = message.payload;

                // Optional: Log the new state (can be verbose)

                // logToMain(`Local shared memory cache is now: ${JSON.stringify(localSharedMemoryCache)}`, 'info');

            // } else {

            //     logToMain(`Ignored shared memory update - INVALID TOKEN.`, 'error');

            //     sendToMain({ type: 'ERROR\_REPORT', context: 'Shared Mem Update', error: 'Invalid Token Received' });

            // }

            break;

        case 'PEER\_UPDATE':

            // Main thread sent updated list of active peers

            logToMain(`Received peer list update from Main Thread.`, 'system');

            peers = message.peers || [];

            logToMain(`Peer list updated. Current known peers: ${peers.filter(*p* => p !== workerId).join(', ') || 'None'}`, 'info');

            break;

         case 'ERROR\_REPORT': // Error reported by main thread (e.g., failed request)

             logToMain(`Received error report from Main Thread: ${message.error} (Context: ${message.context || 'N/A'})`, 'error');

             break;

        default:

            logToMain(`Received unknown message type: ${message.type}. Payload: ${JSON.stringify(message)}`, 'warn');

    }

};

// --- Helper Functions ---

/\*\*

 \* Sends a structured log message back to the main thread for display.

 \* @param *{string}* *logText* - The message to log.

 \* @param *{string}* *[level='info']* - The log level (info, error, warn, system, etc.).

 \*/

function logToMain(*logText*, *level* = 'info') {

    // Ensure workerId is available before sending

    if (workerId) {

        sendToMain({ type: 'LOG', payload: logText, level: level });

    } else {

        // Log locally if worker not initialized yet

        console.warn(`Worker (Uninitialized) Log: [${level}] ${logText}`);

    }

}

/\*\*

 \* Sends structured data back to the main thread.

 \* Automatically adds workerId for context if available.

 \* @param *{object}* *messageData* - The data object to send.

 \*/

function sendToMain(*messageData*) {

    // Add workerId for context if not already present and worker is initialized

    if (workerId && !messageData.workerId) {

        messageData.workerId = workerId;

    }

    // Use structured cloning to send the message object

    self.postMessage(messageData);

}

/\*\*

 \* Simulated security check for tokens received by the worker.

 \* @param *{string}* *token* - The token to validate.

 \* @returns *{boolean}* - True if the token is valid, false otherwise.

 \*/

function isValidToken(*token*) {

    // In a real scenario, this would involve more robust validation.

    return token === VALID\_TOKEN;

}

// --- Example Worker Actions (Could be triggered internally) ---

/\*\* Example: Worker requests to read shared memory from main thread \*/

function requestSharedMemoryRead() {

    if (!workerId) return; // Cannot request if not initialized

    logToMain(`Requesting shared memory read from Main Thread.`, 'shared');

    sendToMain({

        type: 'SHARED\_MEM\_READ\_REQUEST\_FROM\_WORKER',

        token: VALID\_TOKEN // Must provide valid token for the request

    });

}

/\*\* Example: Worker requests to write a key-value pair to shared memory via main thread \*/

function writeToSharedMemory(*key*, *value*) {

     if (!workerId) return; // Cannot request if not initialized

     logToMain(`Requesting write ('${key}':'${value}') to shared memory via Main Thread.`, 'shared');

     sendToMain({

        type: 'SHARED\_MEM\_WRITE\_REQUEST\_FROM\_WORKER',

        payload: { key: key, value: value },

        token: VALID\_TOKEN // Must provide valid token for the request

     });

}

/\*\* Example: Worker explicitly requests to send a message to another worker (routed via main) \*/

function sendMessageToPeer(*peerId*, *messagePayload*) {

     if (!workerId) return;

     if (!peers.includes(peerId)) {

         logToMain(`Cannot send message: Peer '${peerId}' not found in known peers.`, 'error');

         return;

     }

     logToMain(`Requesting Main Thread to route message to ${peerId}.`, 'send');

     sendToMain({

         type: 'ROUTE\_MESSAGE\_REQUEST', // Ask main thread to handle routing

         recipientId: peerId,

         payload: messagePayload,

         token: VALID\_TOKEN // Token required to initiate action

     });

}

// Optional: Add some autonomous behavior for demonstration after initialization

// setTimeout(() => {

//    if (workerId !== null) {

//        logToMain("Initiating autonomous actions...", 'system');

//        requestSharedMemoryRead();

//        writeToSharedMemory(`worker\_${workerId}\_timestamp`, Date.now());

//        // Try sending message to the first available peer (if any)

//        const firstPeer = peers.find(p => p !== workerId);

//        if (firstPeer) {

//             sendMessageToPeer(firstPeer, `Hello from ${workerId}!`);

//        }

//    }

// }, 7000 + Math.random() \* 5000); // Start after 7-12 seconds

logToMain("Worker script loaded and awaiting initialization...", 'system');

index.html

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>Creative IPC Simulator</title>

    <!-- Icons font (using Font Awesome for simplicity) -->

    <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">

    <link *rel*="stylesheet" *href*="style.css">

</head>

<body>

    <div *class*="app-container"> <!-- Main container for background/layout -->

        <h1 *class*="main-title">IPC Framework Simulator</h1>

        <div *class*="container">

            <div *class*="card controls">

                <h2><i *class*="fas fa-cogs"></i> Control Panel</h2>

                <div *class*="control-section">

                    <h3><i *class*="fas fa-plus-circle"></i> Manage Processes</h3>

                    <button *id*="createWorkerBtn" *title*="Create a new simulated process (Web Worker)">

                        <i *class*="fas fa-plus"></i> Create Worker

                    </button>

                </div>

                <hr>

                <div *class*="control-section">

                    <h3><i *class*="fas fa-paper-plane"></i> Send Message</h3>

                    <div *class*="form-group">

                        <label *for*="senderSelect"><i *class*="fas fa-user-astronaut"></i> From:</label>

                        <select *id*="senderSelect">

                            <option *value*="main">Main Thread (Control)</option>

                            <!-- Worker options added dynamically -->

                        </select>

                    </div>

                    <div *class*="form-group">

                        <label *for*="recipientSelect"><i *class*="fas fa-satellite-dish"></i> To:</label>

                        <select *id*="recipientSelect">

                            <option *value*="broadcast">Broadcast <i *class*="fas fa-broadcast-tower"></i></option>

                            <option *value*="main">Main Thread (Control)</option>

                            <!-- Worker options added dynamically -->

                        </select>

                    </div>

                    <div *class*="form-group">

                        <label *for*="ipcMethodSelect"><i *class*="fas fa-project-diagram"></i> Method:</label>

                        <select *id*="ipcMethodSelect" *title*="Direct (Phone Call), Shared Read (Check Board), Shared Write (Write on Board)">

                            <option *value*="direct">Direct Message</option>

                            <option *value*="shared\_read">Read Shared Memory</option>

                            <option *value*="shared\_write">Write Shared Memory</option>

                        </select>

                    </div>

                    <div *class*="form-group">

                        <label *for*="messageInput"><i *class*="fas fa-keyboard"></i> Message / Data:</label>

                        <input *type*="text" *id*="messageInput" *placeholder*="Message or key=value">

                    </div>

                    <div *class*="form-group">

                        <label *for*="tokenInput"><i *class*="fas fa-key"></i> Security Token:</label>

                        <input *type*="text" *id*="tokenInput" *placeholder*="'WORKER\_SECRET'">

                    </div>

                    <button *id*="sendMessageBtn" *title*="Send the configured message">

                         <i *class*="fas fa-rocket"></i> Launch Message

                    </button>

                </div>

                 <hr>

                 <div *class*="control-section">

                    <h3><i *class*="fas fa-memory"></i> Shared Memory</h3>

                    <div *id*="sharedMemoryDisplay" *class*="code-display">

                        <pre>{}</pre>

                    </div>

                 </div>

            </div>

            <div *class*="card logs">

                <h2><i *class*="fas fa-stream"></i> Communication Log</h2>

                <div *id*="logOutput" *class*="code-display">

                    <!-- Logs added dynamically -->

                </div>

            </div>

        </div>

        <footer>

            <p>A Creative IPC Simulation</p>

        </footer>

    </div> <!-- End app-container -->

    <script *src*="script.js"></script>

</body>

</html>

style.css

/\* Import Font \*/

@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap');

/\* Root Variables \*/

*:root* {

    --bg-gradient: linear-gradient(135deg, #1a1a2e, #16213e, #0f3460);

    --card-bg: rgba(255, 255, 255, 0.08); /\* Semi-transparent white \*/

    --card-blur: 15px;

    --border-color: rgba(255, 255, 255, 0.15);

    --text-color: #e0e0e0; /\* Light gray \*/

    --text-muted: #a0a0a0; /\* Medium gray \*/

    --title-color: #ffffff; /\* White \*/

    --accent-color: #00f5d4; /\* Neon Teal/Cyan \*/

    --accent-hover: #00d9c0;

    --error-color: #ff4d6d; /\* Neon Pink/Red \*/

    --success-color: #00f5d4; /\* Same as accent for success \*/

    --info-color: #4ea8de; /\* Soft Blue \*/

    --warn-color: #fca311; /\* Orange \*/

    --font-family: 'Poppins', sans-serif;

    --body-padding: 30px;

    --controls-width-fixed: 420px; /\* Fixed width for controls on large screens \*/

    --layout-gap: 40px;

}

/\* Base Styles \*/

body {

    font-family: var(--font-family);

    line-height: 1.6;

    margin: 0;

    padding: var(--body-padding);

    background: var(--bg-gradient);

    color: var(--text-color);

    min-height: 100vh;

    box-sizing: border-box;

    overflow-x: hidden;

}

*.app-container* {

    max-width: 100%; /\* Use full width available \*/

    margin: 0 auto;

    opacity: 0;

    transform: translateY(20px);

    animation: fadeInSlideUp 1s ease-out forwards;

}

@keyframes *fadeInSlideUp* {

    to {

        opacity: 1;

        transform: translateY(0);

    }

}

*.main-title* {

    text-align: center;

    font-size: 2.5em;

    color: var(--title-color);

    margin-bottom: 40px; /\* Space below title \*/

    font-weight: 600;

    text-shadow: 0 0 10px rgba(0, 245, 212, 0.3);

}

/\* --- Default Layout (Mobile First / Small Screens) --- \*/

*.container* {

    display: flex;

    flex-direction: column; /\* Stack vertically by default \*/

    gap: var(--layout-gap);

}

*.card* {

    background: var(--card-bg);

    backdrop-filter: blur(var(--card-blur));

*-webkit-backdrop-filter*: blur(var(--card-blur));

    border-radius: 15px;

    border: 1px solid var(--border-color);

    padding: 30px;

    box-shadow: 0 8px 32px 0 rgba(0, 0, 0, 0.2);

    width: 100%; /\* Take full width in column layout \*/

    box-sizing: border-box;

    transition: transform 0.3s ease, box-shadow 0.3s ease;

    height: auto; /\* Height based on content \*/

    overflow-y: visible; /\* Let page scroll \*/

    position: static; /\* Ensure it's not fixed by default \*/

}

*.card:hover* {

    transform: translateY(-5px);

    box-shadow: 0 12px 40px 0 rgba(0, 0, 0, 0.25);

}

/\* --- Layout for Larger Screens (using @media query) --- \*/

@media (min-width: 1024px) {

*.container* {

        flex-direction: row; /\* Side-by-side layout \*/

        align-items: flex-start; /\* Align items to the top \*/

        gap: 0; /\* Remove gap, handle spacing with margins/fixed positioning \*/

        position: relative; /\* Needed for absolute/fixed children context potentially \*/

    }

*.controls* {

        width: var(--controls-width-fixed); /\* Fixed width for controls \*/

        flex-shrink: 0; /\* Prevent controls from shrinking \*/

        margin-right: var(--layout-gap); /\* Create space between controls and logs area \*/

        height: auto; /\* Allow controls to grow taller than viewport \*/

        position: static; /\* Stays in normal flow \*/

        overflow-y: visible; /\* Scrolls with the page \*/

    }

*.logs* {

        position: fixed;

        /\* Calculate top based on title height and padding \*/

        /\* Adjust '110px' if your title height is different \*/

        top: 110px;

        right: var(--body-padding); /\* Align with body padding \*/

        /\* Calculate width: viewport - controls width - gap - body paddings (left+right) \*/

        width: calc(100% - var(--controls-width-fixed) - var(--layout-gap) - (2 \* var(--body-padding)));

        /\* Calculate height: viewport height - top offset - bottom padding \*/

        height: calc(100vh - 110px - var(--body-padding));

        overflow-y: auto; /\* Enable internal scrolling for the logs panel \*/

        flex-shrink: 1; /\* Allow shrinking if needed, although width is calculated \*/

         max-width: 800px; /\* Optional: Limit max width on very wide screens \*/

    }

     /\* Make sure the code display area inside logs scrolls \*/

*.logs* *.code-display* {

         height: 100%; /\* Use the full calculated height of the .logs panel \*/

         max-height: none; /\* Override any previous max-height \*/

     }

}

/\* --- General Styles (Remain mostly the same) --- \*/

/\* Headings within cards \*/

h2 {

    color: var(--title-color);

    margin-top: 0;

    margin-bottom: 25px;

    font-weight: 600;

    border-bottom: 1px solid var(--border-color);

    padding-bottom: 15px;

    display: flex;

    align-items: center;

    gap: 10px;

}

h2 i { color: var(--accent-color); font-size: 1.1em; }

h3 {

    color: var(--accent-color);

    margin-bottom: 20px;

    font-weight: 400;

    font-size: 1.1em;

    display: flex;

    align-items: center;

    gap: 8px;

}

h3 i { font-size: 1em; }

*.control-section* { margin-bottom: 25px; }

*.form-group* { margin-bottom: 20px; }

label {

    display: flex;

    align-items: center;

    gap: 8px;

    margin-bottom: 8px;

    font-weight: 400;

    font-size: 0.95em;

    color: var(--text-muted);

}

label i { color: var(--accent-color); width: 15px; text-align: center; }

input[*type*="text"], select {

    width: 100%;

    padding: 12px 15px;

    border: 1px solid var(--border-color);

    border-radius: 8px;

    font-size: 1em;

    box-sizing: border-box;

    background-color: rgba(0, 0, 0, 0.2);

    color: var(--text-color);

    font-family: var(--font-family);

    transition: border-color 0.2s ease, box-shadow 0.2s ease;

}

input[*type*="text"]*::placeholder* { color: var(--text-muted); opacity: 0.7; }

input[*type*="text"]*:focus*, select*:focus* {

    border-color: var(--accent-color);

    box-shadow: 0 0 0 3px rgba(0, 245, 212, 0.15);

    outline: none;

    background-color: rgba(0, 0, 0, 0.3);

}

select {

    appearance: none;

    background-image: url('data:image/svg+xml;utf8,<svg fill="%23a0a0a0" height="24" viewBox="0 0 24 24" width="24" xmlns="http://www.w3.org/2000/svg"><path d="M7 10l5 5 5-5z"/><path d="M0 0h24v24H0z" fill="none"/></svg>');

    background-repeat: no-repeat;

    background-position: right 15px center;

    padding-right: 40px;

}

/\* Button Styles \*/

button {

    background: linear-gradient(135deg, var(--accent-color), var(--accent-hover));

    color: #1a1a2e;

    padding: 12px 25px;

    border: none;

    border-radius: 8px;

    cursor: pointer;

    font-size: 1em;

    font-weight: 600;

    transition: all 0.3s ease;

    display: inline-flex;

    align-items: center;

    gap: 10px;

    box-shadow: 0 4px 15px rgba(0, 245, 212, 0.2);

    letter-spacing: 0.5px;

}

button*:hover* {

    transform: translateY(-2px) scale(1.02);

    box-shadow: 0 6px 20px rgba(0, 245, 212, 0.3);

}

button*:active* {

    transform: translateY(0px) scale(1);

    box-shadow: 0 2px 10px rgba(0, 245, 212, 0.2);

}

*#createWorkerBtn* {

    background: linear-gradient(135deg, var(--info-color), #3a9cd1);

    box-shadow: 0 4px 15px rgba(78, 168, 222, 0.2);

}

*#createWorkerBtn:hover* { box-shadow: 0 6px 20px rgba(78, 168, 222, 0.3); }

*#createWorkerBtn:active* { box-shadow: 0 2px 10px rgba(78, 168, 222, 0.2); }

hr {

    border: 0;

    height: 1px;

    background-image: linear-gradient(to right, transparent, var(--border-color), transparent);

    margin: 30px 0;

}

/\* Code Display Areas \*/

*.code-display* {

    background-color: rgba(0, 0, 0, 0.3);

    border: 1px solid var(--border-color);

    padding: 15px;

    border-radius: 8px;

    font-family: 'Courier New', Courier, monospace;

    font-size: 0.9em;

    /\* Default max-height for small screens \*/

    max-height: 400px;

    overflow-y: auto;

    scrollbar-width: thin;

    scrollbar-color: var(--accent-color) rgba(0,0,0,0.2);

}

*.code-display::-webkit-scrollbar* { width: 8px; }

*.code-display::-webkit-scrollbar-track* { background: rgba(0,0,0,0.2); border-radius: 4px; }

*.code-display::-webkit-scrollbar-thumb* { background-color: var(--accent-color); border-radius: 4px; border: 2px solid rgba(0,0,0,0.2); }

*.code-display::-webkit-scrollbar-thumb:hover* { background-color: var(--accent-hover); }

*#logOutput* div {

    padding: 6px 10px;

    margin-bottom: 6px;

    border-radius: 4px;

    border-left: 3px solid;

    transition: background-color 0.3s ease;

    line-height: 1.5;

    word-break: break-word;

}

*#logOutput* div*:hover* { background-color: rgba(255, 255, 255, 0.05); }

/\* Log message type styling \*/

*.log-info* { border-color: var(--text-muted); color: var(--text-muted); }

*.log-send* { border-color: var(--accent-color); color: var(--accent-color); }

*.log-receive* { border-color: var(--success-color); color: var(--success-color); }

*.log-error* { border-color: var(--error-color); color: var(--error-color); font-weight: 600; }

*.log-system* { border-color: var(--info-color); color: var(--info-color); }

*.log-shared* { border-color: var(--warn-color); color: var(--warn-color); }

*#sharedMemoryDisplay* pre {

    color: #c0c0c0;

    margin: 0;

    padding: 0;

    background: none;

    border: none;

    max-height: none;

    overflow: visible;

}

footer {

    text-align: center;

    margin-top: 50px;

    padding-top: 20px;

    border-top: 1px solid var(--border-color);

    color: var(--text-muted);

    font-size: 0.9em;

}