**Queuing Strategies Documentation**

**Table of Contents**

1. Introduction

2. Design Choices

3. Setup Instructions

4. Usage Instructions

5. Analysis of Queuing Strategies

6. Conclusion

**Introduction**

Welcome! In the world of distributed systems, how we manage tasks is super important. It can make or break performance and efficiency. This project dives into three queuing strategies: **Clustering**, **Priority Queue**, and **Round** **Robin**. Each has its own perks and challenges, suited for different situations.

**Design Choices**

**1. Clustering**

**Goal:** Make the most of CPU power.

**Why it Matters:** Clustering spreads tasks across multiple CPU cores. This boosts responsiveness and handles multiple requests simultaneously. Great for tasks that need a lot of processing power! **2. Priority Queue**

**Goal:**Tackle tasks based on importance.

**Why it Matters:** Sometimes, some tasks are more urgent than others. A Priority Queue manages tasks efficiently. High-priority tasks get handled first, improving the user experience. Think of it as giving VIP access to important guests.

**3. Round Robin**

**Goal:** Share tasks evenly across servers.

**Why it Matters:** Round Robin is a straightforward way to balance the load. It treats each server equally, so no single server gets overwhelmed. This is perfect for tasks that are similar in complexity.

**Setup Instructions**

**Prerequisites**

Before we dive in, make sure you have:

• **Node.js:**You’ll need Node.js installed. Grab it from the [Node.js official website](https://nodejs.org/).

• **Project Files:** Have all the project files ready in your working directory.

**Project Structure**

Here’s a quick look at what your project folder should look like:

/BackendTask

├── /backend

|  |  |  |  |
| --- | --- | --- | --- |
| │ | ├── app.js | | # Clustering setup |
| │ | ├── server.js | | # Express server for clustering |
| │ ├── data/ | | |  |
| │ | │ | └── todos.json | # Data store for todos |
| │ | ├── priorityQueue.js | | # Priority queue implementation |
| │ | ├── queueManager.js | | # Priority queue management |
| │ |  | ├── roundRobin.js |  | # Round Robin load balancer |
| │ |  | ├── process1.js |  | # Backend server 1 |
| │ |  | ├── process2.js |  | # Backend server 2 |
| │ |  | ├── process3.js |  | # Backend server 3 |
| ├── /frontend | |  |  |  |
| │ |  | ├── dist/ |  | # Compiled files |
| │ |  | ├── index.html |  | # Main HTML file |
| │ |  | ├── node\_modules/ |  | # Dependencies |
| │ |  | ├── package.json |  | # Project metadata |
| │ |  | ├── package-lock.json |  | # Dependency lock file |
| │ |  | ├── src/ |  | # Source files |
| │ │ ├── App.css | |  | # Styles for the app |  |
| │ │ ├── App.jsx | |  | # Main React component |  |
| │ │ └── main.jsx | |  | # Entry point for React app |  |
| │ └── vite.config.js | |  | # Vite configuration file |  |

**Installing Dependencies**

Let’s get everything set up:

Navigate to your project directory: cd BackendTask

Move to the backend folder: cd backend

Install necessary packages (if any): npm install

Now, head to the frontend directory: cd ../frontend

Install frontend dependencies: npm install

**Usage Instructions**

**Running the Clustering Approach**

To kick off the clustering server, just follow these steps:

|  |  |
| --- | --- |
| 1. | Open your terminal. |

Go to the backend folder: cd BackendTask/backend

Run this command: node app.js

**Running the Priority Queue Approach**

To start the priority queue server, do this:

|  |  |
| --- | --- |
| 1. | Open your terminal. |

Navigate to the backend folder: cd BackendTask/backend

Execute the command: node priorityQueue.js

**Running the Round Robin Approach**

Setting up the Round Robin load balancer is easy! Here’s how:

1. Open three terminal windows.

In the **first terminal**, start the Round Robin load balancer: cd BackendTask/backend

node roundRobin.js

In the **second terminal**, start the first backend process: cd BackendTask/backend

node process1.js

In the **third terminal**, start the second backend process: cd BackendTask/backend

node process2.js

In the **fourth terminal**, start the third backend process: cd BackendTask/backend

node process3.js

In the **fifth terminal**, run the frontend to send requests: cd BackendTask/frontend

npm run dev # or use the command to start the Vite server

**Analysis of Queuing Strategies**

**Clustering**

• **Pros:**

◦ Uses all CPU cores efficiently.

◦ Boosts performance during heavy loads.

• **Cons:**

◦ Might be too much for small applications.

◦ Can be tricky to set up and manage resources.

**Priority Queue**

• **Pros:**

◦ Flexibly handles tasks based on urgency.

◦ Shortens wait times for critical tasks.

• **Cons:**

◦ Low-priority tasks might be delayed.

◦ Needs careful management to avoid bottlenecks.

**Round Robin**

• **Pros:**

◦ Simple and effective for load balancing.

◦ Distributes tasks fairly, stopping server overload.

• **Cons:**

◦ Doesn’t consider task complexity; simpler tasks might take longer.

◦ May leave some servers idle while others work hard.

**Conclusion**

In summary, the right queuing strategy depends on your app's needs.

• Use **Clustering** for CPU-heavy tasks.

• Go for a **Priority Queue** when some tasks need that extra push.

• Choose **Round Robin** for even load distribution.

Now, get coding! Happy building!