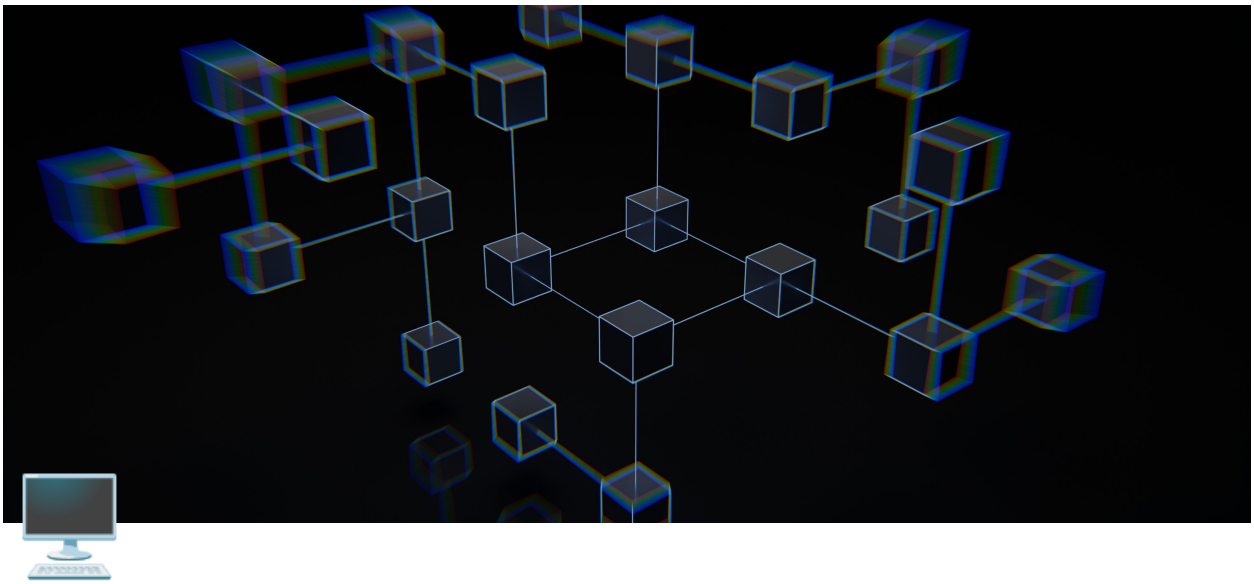

A3 - Sistemas Distribuídos e Mobile

Componentes:

- Arthur Oliveira Passos - 1272022772
- Filipe Souza Alves - 1272023233
- Kaled Freire Barreto - 1272023144

Requerimentos:

Os requerimentos necessários para a execução da aplicação é o Node.js utilizando o JavaScript.

Como rodar o projeto?

O primeiro passo para começar a rodar o projeto é o clone do projeto, o primeiro passo a se fazer no terminal é rodar o comando `npm i`, que assim irá instalar todos os pacotes necessários para rodar o projeto.

Após a instalação dos pacotes, será necessário abrir o arquivo `nodejs.bat` com um duplo clique (Windows), e ele automaticamente irá abrir seis terminais, onde um deles

será o **servidor principal**, dois serão o ***seller***, dois serão o ***manager*** e os outros dois restantes será com uma cargo aleatório, na qual cada um deles tentará iniciar uma instância em uma das portas disponíveis.

Caso queira adicionar um cargo específico ao iniciar, basta apenas passar como parâmetro, uma dos seguir: "*seller*" ou "*manager*", como por exemplo: `node index.js seller`.

Caso o servidor principal seja finalizado, para que outro suba novamente como principal, deve ser rodado o comando `node index.js main_server`

Detalhamento do projeto:

As bibliotecas abaixo são utilizadas no ambiente Node.js para diversas finalidades:

1. `axios`: É uma biblioteca para realizar requisições HTTP. Ela facilita a comunicação com servidores e APIs, permitindo enviar requisições e receber respostas de forma simples e eficiente.
2. `express`: É um framework web minimalista e flexível para Node.js. Ele facilita a criação de aplicativos e APIs web, fornecendo recursos para lidar com rotas, middlewares, gerenciamento de sessões e muito mais.
3. `fs` (File System): É um módulo integrado do Node.js que fornece métodos para interagir com o sistema de arquivos. Com ele, é possível ler, escrever, atualizar e excluir arquivos, criar diretórios, entre outras operações relacionadas a arquivos e pastas.
4. `path`: É um módulo integrado do Node.js que lida com caminhos de arquivos e diretórios de forma eficiente e multiplataforma. Ele fornece métodos para manipular caminhos, resolver caminhos relativos, extrair informações sobre caminhos, entre outras funcionalidades úteis.
5. `sqlite3`: É uma biblioteca para integração com bancos de dados SQLite no Node.js. Ela permite a criação, manipulação e consulta de bancos de dados SQLite de forma simples e eficiente.
6. `@faker-js/faker`: É uma biblioteca que oferece recursos para geração de dados fictícios, como nomes, endereços, números de telefone, textos, entre outros. É útil para criar dados de teste ou preencher informações em bancos de dados durante o desenvolvimento.

7. `uuidv4` (UUID Version 4): É uma biblioteca para geração de identificadores únicos universalmente. Ela permite a criação de identificadores únicos baseados no padrão UUID v4, que são amplamente utilizados em várias aplicações para garantir a singularidade de identificadores.

Todas as instâncias são configuradas para escutar em uma porta específica, que é determinada pelo método `start()`. Esse método invoca a função `getAll()`, responsável por buscar em uma lista (no formato JSON) todas as supostas instâncias disponíveis. O mesmo percorre essa lista, verificando qual porta está disponível para uso.

Iniciando na porta de início, a instância itera sobre as portas até encontrar uma porta livre. Quando uma porta é encontrada, o servidor é criado e o processo continua. No entanto, se todas as portas estiverem ocupadas, é emitida uma mensagem: **Não há portas disponíveis**.

Além de verificar a disponibilidade da porta, a instância também recebe o cargo como parâmetro. Exemplos de papéis incluem **vendedor**, **gerente** e **servidor**. Após criar o servidor com a porta disponível, a instância registra na listagem a sua presença.

Ao receber um dos cargos é removido todas as funcionalidades que ele pode ter possuído anteriormente e em seguida, o servidor identifica o cargo atribuído e atribui funções específicas com base nessa informação. No código, isso pode ser verificado através da condição `type === 'seller'` ou outros cargos definidos.

Atualmente, no contexto em que nos encontramos, é adotado um modelo de armazenamento centralizado conhecido como "Single Source of Truth" (Única Fonte de Verdade) para lidar com múltiplas instâncias. Esse modelo nos permite garantir que todas as instâncias estejam perfeitamente alinhadas, compartilhando os mesmos dados essenciais. Para alcançar esse alinhamento, cada instância consulta um arquivo denominado `servers.json` para identificar as demais instâncias presentes na rede.

Quando uma nova instância é adicionada à rede, ela se registra nesse arquivo central, permitindo que o servidor principal tenha conhecimento de sua existência. Por outro lado, caso o servidor perceba que uma instância não está respondendo, essa instância é removida da lista, mantendo a integridade do sistema.

O servidor principal é iniciado automaticamente, a menos que já tenha sido explicitamente desativado anteriormente. Caso nenhum servidor esteja ocupando essa função, o servidor principal é designado com uma porta específica e um ID distinto. No caso de uma falha ou desligamento do servidor principal, uma eleição é realizada entre

as instâncias ativas para determinar qual delas assumirá temporariamente o papel de servidor principal.

Antes de cada requisição, é requisitado ao servidos uma simples conexão e caso ele não se torne responsivo, é iniciada uma eleição, em caso de eleição, o algoritmo busca na listagem centralizada de instâncias e realiza um teste de conectividade baseado nela. Portanto, caso o servidor não esteja responsivo, ele sairá da listagem. No entanto, assim que um servidor se tornar responsivo, ele será eleito como servidor temporário.

É importante ressaltar que o servidor temporário, além de executar as funções normais de um servidor, verifica se o servidor principal já foi inicializado novamente. Além disso, é possível iniciar o servidor principal em um terminal diferente do que foi utilizado para iniciar o sistema, basta utilizar o comando apropriado de inicialização já explicado anteriormente. Se o servidor principal estiver disponível novamente, quem foi eleito retorna ao seu cargo anterior e devolve o cargo de “servidor” ao servidor principal.

Para manter a comunicação entre servidores, antes de cada requisição a instância envia um *ping* para o servidor, aguardando a resposta do *pong* por meio do método `getServer()`.

A arquitetura da instância, com sua capacidade de receber *pings*, responder *pongs*, gerenciar portas disponíveis e desempenhar diferentes papéis, oferece uma base sólida para sistemas distribuídos e de comunicação entre servidores, promovendo a eficiência e confiabilidade nas interações entre os componentes.