Construção de uma base de dados descentralizada em contexto Smart City

João Nuno Rodrigues Pratinha de Araújo



Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

Junho de 2021

Agradecimentos

Gostaria de agradecer à Câmara da Maia e ao Engenheiro Pedro Pimenta pela oportunidade de trabalhar no projecto da cidade virtual que estão a desenvolver e pela oportunidade de desenvolver técnicas de programação, assim como, a possibilidade de individualmente, planear e implementar um sistema que cumprisse os objectivos propostos.

Também gostaria de agradecer ao professor na Faculdade de Ciências da Universidade do Porto e coordenador de estágio José Paulo Leal por me ajudar a perceber falhas no sistema que estava a planear permitindo a criação de versões melhoradas.

Finalmente quero agradecer ao aluno da Faculdade de Engenharia da Universidade do Porto, Tiago Filipe Lima Rocha, com quem pude sempre contar sempre que me surgiram dúvidas na implementação do projecto.

Índice

1 [Introdução](#_937kowza3w5) ……………………………………………………………….. 4

2 [Estado de Arte](#_wcsnsnp52u9h) ………………………………………………………….. 6

[2.1 Protocolo de coordenação de máquinas](#_kvetj81b6ez2) ………………………… 6

[2.1.1 Simple Network Management Protocol (SNMP)](#_xc5suznx9vkd) ……………………. 6

[2.2 Base de dados descentralizada](#_k75mup26ylej) ………………………………………. 7

[2.2.1 BlockChain](#_pebslxa3przq) ………………………………………………………………………….. 7

[2.2.2 NGROK](#_d79qnsem8rah) ………………………………………………………………………………... 7

3 [Descrição do Trabalho Realizado](#_q8kc6lljh7qb) ………………………………. 8

[3.1 Integração na equipa e descrição do problema](#_8awrjyrekxg5) ……………….. 8

[3.2 Protocolo de coordenação de máquinas](#_tyh10f18qgjh) …………………………. 9

[3.2.1 Versão 1](#_q6f7ragdz9d8) ……………………………………………………………………………… 9

[3.2.2 Versão 2](#_93js0nvogbxj) ……………………………………………………………………………. 11

[3.2.3 Versão 3](#_2pgmwoe7k3b) ……………………………………………………………………………. 13

[3.2.3.1 Gestor](#_ytsuszvrdzic) …………………………………………………………………………….. 14

[3.3 Pesquisa de ASPs gratuitos](#_809otzx48996) ………………………………………….. 16

[3.4 Base de dados descentralizada](#_1ydwdt1z7vcn) …………………………………….. 16

[3.4.1 Sincronizador](#_3ne3x37u915i) ……………………………………………………………………. 16

[3.4.2 Tunneling](#_gf2ivoyprzwe) ………………………………………………………………………….. 18

4 [Conclusão](#_qjnhbvb3uhge) ………………………………………………………………..19

[5 Apêndices](#_k2n4pdiratz4) ...……………………………………………………………...21

Lista de figuras

Figura 1 Esquema do funcionamento SNMP …………………………………………..10

Figura 2 Transferência de informação: *Polling* vs. *Trap*…………………………. 10

Figura 3 Esquema do funcionamento da versão 2 (1ªparte) …………………...12

Figura 4 Esquema do funcionamento da versão 2 (2ªparte) …………………...12

Figura 5 Esquema do funcionamento da versão 2 (3ªparte) …………………...13

Figura 6 *Print Screen* do funcionamento do gestor …....……....…………………...15

Figura 7 Esquema do funcionamento do gestor ……………………………………..17

Figura 8 *Print Screen* do uso do Ngrok …....……………………………………………..18

Capítulo 1

# Introdução

Relatório do projecto “Desenvolvimento de base de dados descentralizada em contexto Smart City”, realizado remotamente no âmbito da unidade curricular de Estágio, parte da Licenciatura de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto. Proposto pela entidade acolhedora, Câmara Municipal da Maia, como parte do projecto de sustentabilidade “BaZe” (Projecto Balanço Zero) que visa transformar a cidade da Maia numa cidade inteligente (*Smart City)*.

As cidades inteligentes são actualmente uma prioridade de várias cidades à procura de modernizar. São áreas urbanas caracterizadas pelo uso de sensores e outros dispositivos para a recolha de informação em múltiplos pontos da cidade. Essa informação pode ser usada para melhorar operações, serviços e gestão de recursos em toda a cidade, aumentando a eficácia de todos os sectores.

O BaZe é um projecto que pretende utilizar a recolha de dados em grande escala para poder implementar medidas e novas iniciativas de descarbonização da cidade, que considera um fator determinante para a sustentabilidade na Maia. O projecto nasce da ambição da Câmara Municipal da Maia em colocar a sua cidade como a primeira “Net Zero Carbon City” a nível nacional.

O objectivo do projecto descrito neste relatório foi implementar uma base de dados descentralizada para dados recolhidos por computadores dispersos pela cidade, criando uma rede de computadores sem pontos de falha. O projecto foi dividido em duas fases, conceber e implementar um protocolo para a coordenação de uma rede de computadores na recolha de dados, e a criação de um sistema que construa e descentralize uma base de dados para os dados recolhidos.

O protocolo desenvolvido foi baseado no protocolo para gestão de redes “*Simple Network Management Protocol”*, implementado em *Node Js* usando a linguagem *JavaScript* e inspirado em tecnologia *BlockChain* na criação do sistema de descentralização da base de dados.

Para a realização deste projecto foi necessário o estudo de protocolos de administração de redes para usar como base na idealização do sistema, assim como o estudo e familiarização com tecnologias WEB e técnicas *JavaScript* para implementação do sistema.

Capítulo 2

# Estado de Arte

No desenvolvimento deste projecto foi necessário adquirir prática e novos conhecimentos programação com a linguagem *JavaScript* em *Node js* e foram estudadas diferentes tecnologias em que foram baseados o protocolo de coordenação de máquinas e a base de dados descentralizada.

## 2.1 Protocolo de coordenação de máquinas

### 2.1.1 *Simple Network Management Protocol* (SNMP)

De forma a coordenar as máquinas que fazem as leituras de informação, decidi criar um protocolo baseado no protocolo de redes para recolher e organizar informação SNMP. Este é um protocolo orientado à informação que permite a monitorização de um grande número de equipamentos diferentes de uma rede com o uso de notificações síncronas e assíncronas. Também oferece grande flexibilidade para a gestão de redes através do acesso e modificação das variáveis que configuram os dispositivos conectados.

Neste projecto foi implementada uma versão do SNMP simplificada porque apenas foi necessário coordenar apenas computadores que realizam a mesma função e contém variáveis semelhantes, mas a implementação pode facilmente ser expandida para incluir um maior número de máquinas capazes de realizar todo o tipo de função, seguindo a metodologia do SNMP.

## 2.2 Base de dados descentralizada

### 2.2.1 *BlockChain*

Para a implementação da base de dados foi utilizado um sistema inspirado em tecnologia *BlockChain*. Esta tecnologia geralmente associada a criptomoedas, é utilizada para guardar, por exemplo, informação sobre todas as transacções de uma criptomoeda entre um grande número de pessoas. Como a informação é mantida de forma descentralizada, qualquer modificação dos dados pode ser identificada comparando com a versão dos dados dos outros elementos envolvidos nas transacções, garantindo a segurança e a fiabilidade da informação guardada.

Neste projecto, o mesmo sistema foi implementado para descentralizar a base de dados relativa aos dados recolhidos nas leituras.

### 2.2.2 *NGROK*

Para permitir a comunicação entre máquinas localizadas em redes diferentes, foi usado o sistema de *tunneling* “ngrok” que faculta uma *cloud* que disponibiliza endereços públicos e, através de um túnel, redireciona o tráfego para o *localhost*.[1]

Este serviço foi usado para conectar servidores à internet na fase de teste do produto final do projecto.

Capítulo 3

# Descrição do Trabalho Realizado

As tarefas realizadas ao longo do estágio desde a integração na equipa criada pelo Engenheiro Pedro Pimenta, até à conclusão do projecto, serão descritas neste capítulo de forma cronológica.

## 3.1 Integração na equipa e descrição do problema

O início do estágio ocorreu em pleno confinamento causado pela pandemia “Covid-19” o que alterou drasticamente a tradicional integração no mercado de trabalho. Todo o estágio foi realizado remotamente.

As primeiras semanas foram dedicadas à apresentação das etapas e objectivos do projecto e à integração na equipa de estagiários, coordenada pelo Engenheiro Pedro Pimenta. A equipa era constituída por dez elementos, quatro engenheiros de dados, quatro cientistas de dados, o coordenador Pedro Pimenta e eu. O objectivo da equipa era usar os quatro meses de estágio curricular para progredir no projecto de longo prazo da Câmara Municipal da Maia de construir uma *Smart City*. Foi explicado que o meu trabalho iria ser realizado a solo enquanto os restantes oito estagiários trabalhavam em tratamento de informação.

As etapas definidas para a resolução do meu projecto foram:

* Criação e implementação de um protocolo usado para coordenação de um sistema de computadores que faz a leitura de dados relativos às condições meteorológicas na cidade da Maia.
* Encontrar ASPs gratuitos para alojar os algoritmos desenvolvidos.
* Desenho e criação de uma base de dados descentralizada para guardar os dados das leituras.

## 3.2 Protocolo de coordenação de máquinas

No primeiro ciclo do estágio foi planeado e implementado um protocolo para a coordenação de uma rede de máquinas que fazem leitura de dados, que seja eficiente e evite a sobrecarga da API para a recolha de dados com acções redundantes.

Como a API atribuída para testes era invocada através de um URL, decidi implementar este protocolo em *node js* em *JavaScript*.

### 3.2.1 Versão 1

Inicialmente, escolhi desenhar um protocolo baseado no *Simple Network Management Protocol* (SNMP), o protocolo padrão de administração de redes, especializado no controle remoto de dispositivos diferentes de uma rede.

O SNMP funciona com um modelo “*Manager-Agent*”, definindo dois tipos de dispositivos:

* Dispositivos geridos, contém um *software* de agente SNMP e uma MIB (*Management Information Base*).
* Estação de gestão contém *software* de gestor para gerir os dispositivos geridos e uma interface para o gestor humano.

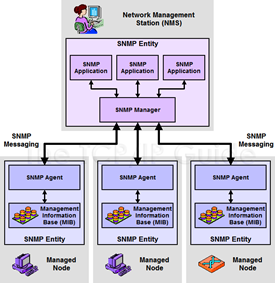
Para possibilitar gerir dispositivos muito diferentes, o SNMP implementa um modelo orientado à informação em vez de um orientado a comandos. A MIB de um dispositivo contém variáveis (objectos) que podem ser lidas e/ou escritas, possibilitando a execução de comandos através da escrita de variáveis.

Para além de um sistema de *polling* para o gestor alterar variáveis no dispositivo, o SNMP suporta também um sistema de notificações assíncronas chamado *trap* que permite os dispositivos geridos notificarem o gestor com actualizações sem a necessidade de um pedido por parte do gestor[2].

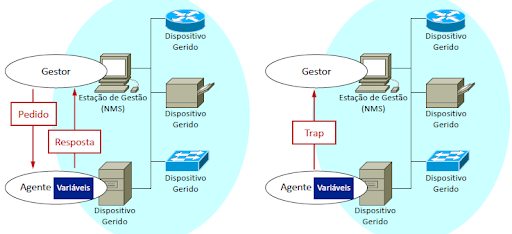
Propus para este projecto, a criação de *software* agente para os computadores responsáveis pela recolha de dados e uma estação de gestão para os coordenar.

A intercalação da actividade de recolha de dados das máquinas é obtida pelo uso de uma variável “intervalo de tempo”.*.*

Este protocolo foi chumbado por Pedro Pimenta pois considerou a estação de gestão, um ponto de falha no protocolo. Também determinou que a configuração de cada máquina com uma variável *timeInterval* obrigaria ao reajuste das variáveis *timeInterval* em todos os agentes da rede.



**FIG 3.1:** Esquema do protocolo de SNMP [3]



**FIG 3.2:** Transferência de informação: *Polling* (esquerda) vs. *Trap* (direita).

### 3.2.2 Versão 2

Este protocolo foi criado para tentar colmatar as falhas do protocolo anterior. Nomeadamente evitar qualquer falha que possa suspender o funcionamento de todo o sistema e facilitar a inclusão de novas máquinas à rede.

Este novo protocolo foi baseado num sistema “*producer-consumer*” para sistemas distribuidos, mas ambos os algoritmos *producer* e *consumer* executam na mesma máquina comunicando por HTTP.

Cada máquina contém um *Machine API* e um *Web API*.

A *Web API* integra o *consumer* que, na máquina que desperta primeiro, envia um sinal para o *producer* de todas as máquinas incluindo a própria para as despertar.

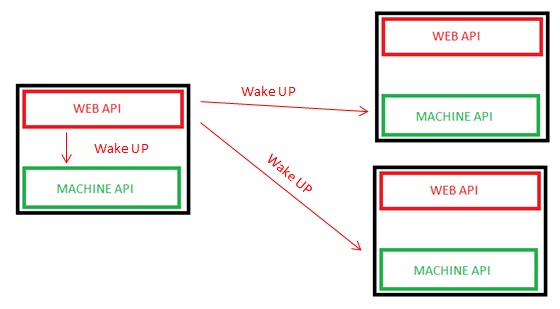
A *Machine API* contém um *fetcher* e um *producer*. O *producer* recebe o sinal para despertar e gera um número aleatório entre 1 e 10000 e devolve para a *Web API* um objecto JSON que inclui o número gerado aleatoriamente e o identificador da máquina que o gerou. O *consumer* em cada máquina recebe os objectos gerados pelos vários *producers* e compara os números gerados por todas as máquinas e retorna para o *fetcher*  na *Machine API* o objecto JSON com o número gerado mais alto.

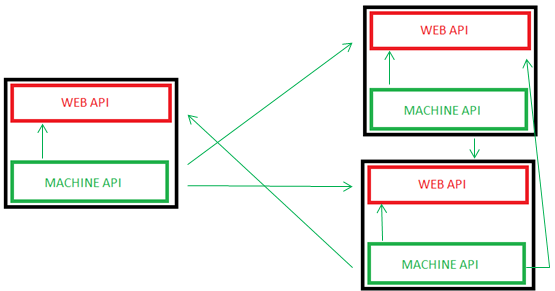
O objectivo do *fetcher* é invocar as API’s de leitura mas apenas depois de verificar que o *fetcher* está localizado na máquina que gerou o número mais alto.

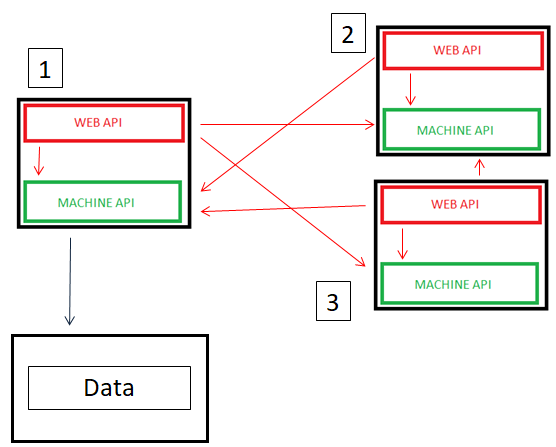
Deste modo, apenas a máquina que gerou o número mais alto faz a leitura de dados, evitando sobrecarregar as API’s. Mesmo que uma máquina falhe e não responda, não influencia o processo pois basta uma máquina estar activa para o protocolo funcionar. Finalmente, a inclusão de uma nova máquina na rede é fácil pois apenas é necessário configurar o endereço IP e *port* da nova máquina e a constante que identifica a máquina.

O facto da primeira versão ter sido rejeitada após a primeira semana e meia teve um grande efeito na escolha de usar um sistema de *broadcast* dada a proximidade do prazo limite, a maior simplicidade na apresentação e o objectivo inicial deste protocolo ser para uso prático na coordenação de apenas quatro ou cinco máquinas.

Durante a implementação deste protocolo encontrei várias dificuldades na implementação, principalmente na escalabilidade, pois sem um elo central para gerir a comunicação, a entrada de uma nova máquina no sistema seria pouco prática e significaria a reconfiguração da forma da rede em todos os computadores.

**Fig.3.3 -** Primeira máquina a acordar, desperta as restantes.

**Fig 3.4 -** Envio de números gerados aleatoriamente.



**Fig. 3.5 -** Partilha do número mais alto e máquina que o gerou faz a leitura de dados.

### 3.2.3 Versão 3

Após consideração e debate, devido às dificuldades encontradas na implementação do protocolo v2, particularmente a implementação de comunicação entre máquinas de forma descentralizada, resolvi implementar um protocolo baseado no primeiro protocolo apresentado mas com as modificações e melhorias necessárias para completar os objectivos definidos.

O sistema funciona num sistema “gestor-agentes” com máquinas que operam como agentes totalmente independentes e fazem, rotativamente, invocações à API de recolha de dados num intervalo de tempo escolhido pelo gestor. Cada máquina usa um *timer* para fazer uma recolha de dados através do cálculo das três variáveis: intervalo de tempo da rotação, número de máquinas e o número identificador da máquina.

Isto permite ao gestor do sistema, modificar o comportamento das máquinas enviando pedidos para os agentes por HTTP para alterar as variáveis quando se aumenta o número de agentes ou o tempo entre leituras.

A configuração das variáveis é guardado no ficheiro “conf.JSON”.

Para permitir que os agentes ajam independentemente uns dos outros, sem sobreposição de invocações da API, cada um vai calcular o intervalo de tempo entre as suas recolhas de dados através das variáveis “ID” do agente, “número de máquinas na rede” e o “intervalo de tempo” escolhido pelo gestor no qual a rede de agentes deve fazer uma leitura. Para manter o funcionamento e evitar sobreposições, é importante que os IDs dos agentes nunca sejam configurados com um número superior ao do total de agentes. Assim, se um agente for retirado da rede e os restantes agentes configurados para um número total de máquinas inferior, os IDs devem ser atualizados para que não haja nenhum acima do número total de máquinas.

Como exemplo considere-se uma rede de 5 agentes com IDs {1, 2, 3, 4, 5}. Se o agente 3 for retirado da rede e a configuração do número total de agentes, alterado para 4, então é sugerido que o ID do agente 5 seja alterado para 3 para evitar que o agente 1 e 5 invoquem a API simultaneamente. Alteração na variável “número de máquinas na rede” ou na variável “intervalo” causam os ciclos de leitura de dados a ajustar e recomeçar.

Como objectivo seguinte criei um algoritmo para o gestor, executável em qualquer computador, permitindo facilmente modificar as variáveis em cada agente ou fazer *broadcasts* de comandos, para todos os agentes, para iniciar ou parar os *timers.*

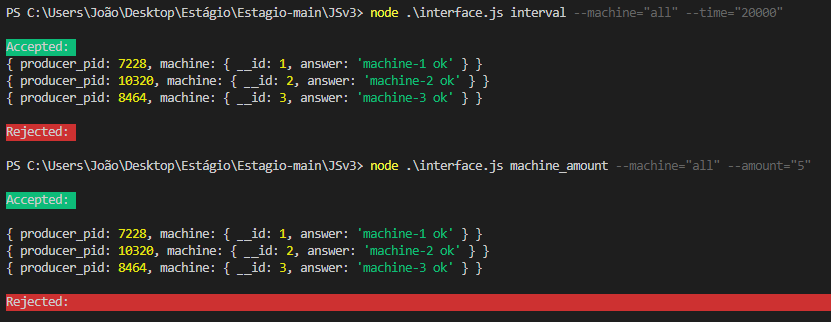
### 3.2.3.1 Gestor

O gestor tem como principal função o controle remoto dos vários agentes. O gestor consegue enviar ordens para iniciar as leituras de dados no intervalo de tempo configurado nos agentes assim como interromper o seu funcionamento. Tem também a capacidade de alterar as variáveis configuradas nos agentes como o intervalo de tempo entre leituras, o número total de agentes activos na rede e o ID de uma máquina.

Para estabelecer a comunicação de um gestor para vários agentes, decidi usar o método do *JavaScript, Promise.allSettled()* que permite fazer múltiplos *querys* e registar as respostas distinguindo os sucessos dos erros. Não foram encontrados na minha pesquisa, muitos exemplos de implementações deste método introduzido na versão 12.9.0 do *Node js*, que fossem compatíveis com o objectivo que pretendia, atrasando esta componente do projecto.

O endereço dos agentes está identificado numa lista no ficheiro “machines.JSON”. Neste ficheiro deve ser definida qualquer alteração na localização ou número de agentes.

Foi necessário a implementação de uma interface simples para invocar individualmente qualquer função existente no *manager.js.* Uma possibilidade foi a criação de uma página HTML mas não permitia a utilização de ficheiros *JSON* onde a está guardado a localização dos agentes por isso a ideia foi abandonada em prol de uma interface em *node js,* utilizando a linha de comandos através do módulo “yargs”.



**Fig. 3.6 -** Exemplo de uso de interface para alterar variáveis em 3 agentes activos.

Foram implementados vários comandos para o manager no formato “node ./interface.js cmd arg1 arg2…”, por exemplo, ‘**node ./interface.js check --machine=”all”**’.

Seguem os vários comandos introduzidos e descrição, omitindo a invocação da interface usando o comando *node:*

* **ls →** Imprime na consola as configurações gravadas (Intervalo dos agentes, número de máquinas total e versão da base de dados corrente),
* **interval --time=”60” →** Envia a todos os agentes o novo intervalo do ciclo de recolha de dados em segundos,
* **machine\_amount --amount=”5” →** Envia a todos os agentes a informação do número total de agentes na rede.
* **machine\_ID --machine=”1” --id=”2” →** Modifica o ID do agente com o ID recebido como primeiro argumento para o número recebido no segundo argumento.
* **check --machine=”1” →** Imprime na consola as configurações do agente. Este comando pode ser usado com o argumento **--machine=”all”** de forma a fazer o pedido a todos os agentes.
* **start --machine=”1” →** Ordena um agente para começar o “timer” para a leitura de dados. Este comando pode ser usado com o argumento **--machine=”all”** de forma a fazer o pedido a todos os agentes.
* **stop --machine=”1” →** Ordena um agente para parar as leituras. Este comando pode ser usado com o argumento **--machine=”all”** de forma a fazer o pedido a todos os agentes.

## 3.3 Pesquisa de ASPs gratuitos

O segundo ciclo deste projecto envolvia encontrar um serviço preferencialmente com um plano gratuito, que fornecesse uma plataforma e os recursos para hospedar o sistema de base de dados descentralizado que se pretende criar.

Foram estudadas várias hipóteses e encontrados dois serviços com as capacidades pretendidas: “Back4App” e “Heroku”. No entanto, foi planeada uma implementação da base de dados descentralizada usando um protocolo que não necessita de uma plataforma online para funcionar e pode até ser usado em redes internas. Como tal, durante o projecto, não foi necessário usar os serviços investigados.

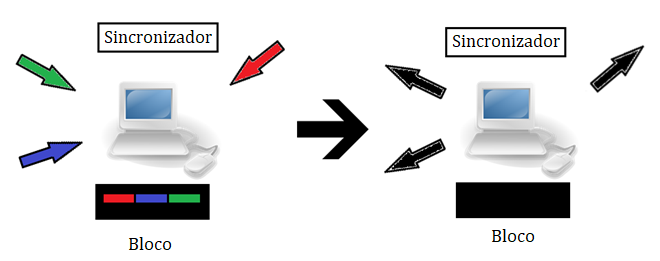
## 3.4 Base de dados descentralizada

O objectivo principal deste estágio foi a criação de uma base de dados descentralizada para os dados recolhidos das APIs. Para isso, foi implementado um protocolo inspirado em tecnologia *blockchain* para descentralizar os dados através de um novo agente que chamei de sincronizador.

### 3.4.1 Sincronizador

Cada agente vai conter mais três estruturas: as listas ‘*collection*’ e ‘*database*’ (base de dados persistente) e uma variável ‘versão’ referente à versão mais actual da base de dados. Sempre que um agente faz uma invocação da API de recolha de dados, o objecto *JSON* retornado é guardado na lista *collection*.

Existe um agente singular chamado sincronizador que também opera num intervalo de tempo e tem como função reunir as listas *collection* de todos os agentes num bloco com a lista de todas as leituras desde a última sincronização. Este bloco é devolvido, assim como o número da nova versão da base de dados. A lista de leituras devolvida a cada agente vai ser acrescentada à lista *database* que contém a versão da base de dados actualizada. A lista *collection* é reiniciada em cada agente. Desta forma cada agente contém a base de dados total no ficheiro “database.JSON”, obtendo assim a descentralização.



**Fig. 3.7 -** Sincronizador reúne 3 agentes, as listas *collection* num bloco (esquerda). Devolve aos agentes a nova versão e o bloco com as leituras a acrescenta à base de dados (direita)

O sincronizador não é um ponto de falha pois apesar de ser uma entidade única na rede, não é responsável por guardar informação. Assim, mesmo que a máquina que está a executar o sincronizador tenha uma falha crítica, não é um problema pois cada agente vai continuar a guardar os dados de cada recolha na lista *collection* até ser encontrado um novo local para correr o sincronizador. Se no pior caso possível, o sincronizador falha e posteriormente outras máquinas falham, então são perdidos os dados recolhidos pelas máquinas que falharam, mas apenas desde a última sincronização.

Por uma questão de eficiência, apenas as listas de dados desde a última sincronização são enviados para o sincronizador, logo, novos agentes inseridos na rede ou que falhassem a sincronização receberiam apenas a base de dados relativos aos dados recolhidos pelos agentes mais antigos desde a última sincronização. Neste caso, o sincronizador reconhece os agentes que têm uma versão da base de dados desactualizada e partilha com estes agentes a base de dados inteira.

Posteriormente pode ser adicionada à rede outros agente capazes de sincronizar bases de dados ou o gestor pode adquirir essa função, criando uma situação em que o sincronizador pode desactualizado. Neste caso sincronizador considera que a versão de número superior é a mais actualizada e reconhece o agente com a versão de base de dados mais elevada e faz a actualização.

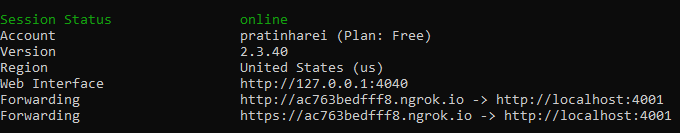
Na implementação do sincronizador, foi novamente necessário o uso da função *JavaScript Promise.allSettled()* para iniciar múltiplas comunicações com servidores, mas devido a dificuldades que encontrei no uso dessa função com o método *Post* do HTTP, decidi usar o módulo *Axios* que contém uma função de comportamento semelhante, *axios.all().*

### 3.4.2 *Tunneling*

Era importante que o sistema estivesse ligado à internet para permitir o funcionamento do protocolo através de máquinas localizadas em redes diferentes.

Depois de implementado o código, foi testado em parceria com um colega de estágio responsável pela recolha e tratamento de dados. O objectivo era usar o produto final para recriar a sincronização de base de dados em computadores em locais geográficos diferentes em sintonia com o trabalho do colega no tratamento de dados. Inicialmente foi encontrado um obstáculo na comunicação entre computadores em redes diferentes devido a *routers* NAT e à *Windows Firewall* que impede a comunicação directa entre redes domésticas. Estes obstáculos foram contornados com a utilização de *software* de terceiros.

Devido à fácil utilização do serviço, escolhi usar o sistema ngrok que cria um túnel entre o *localhost* e a internet gerando um URL público no momento em que é estabelecida a conexão. Este URL é fornecido ao gestor e sincronizador estabelecendo comunicação no sistema. Para isso basta executar o terminal ngrok fornecido e correr a instrução “**ngrok http 4001”**, em que o número 4001 representa uma possível porta do *localhost* onde o agente está “*listening”*.

**Fig. 3.8 -** Túnel criado pelo ngrok a partir do url “http://localhost:4001”

Este serviço foi suficiente em contexto de testes mas a versão paga permite o uso de subdomínios fixos evitando ter de configurar o ficheiro que contém todos os agentes e o seu endereço público. Também confere uma nova camada de protecção para a conexão através de palavra-passe. Também foram encontradas alternativas ao ngrok, como o “serveo” ou “Teleconsole” entre outros, com funcionalidades gratuitas e pagas e ainda uma alternativa completamente grátis em “LocalTunnel”. As vantagens e desvantagens entre estes serviços assim como uma hiperligação para cada serviço podem ser encontradas nas referências[4].

Capítulo 4

# Conclusão

No início do projecto fui atraído para este tema pela curiosidade na possibilidade de teorizar uma rede de computadores e depois implementá-la. Sempre tive interesse em protocolos de rede e senti-me sempre animado para fazer uso desses conhecimentos. Ao mesmo tempo, senti preocupação pelo facto de estar sozinho encarregado deste projecto dada a minha inexperiência e falta de treino para além do recebido durante a licenciatura. Penso que essa dificuldade foi aumentada pela situação actual da pandemia que tornou este estágio num trabalho remoto. Como consequência, quaisquer dúvidas durante o progresso tinham de ser esclarecidas num horário combinado e compatível entre o aluno e orientadores para reunir online. Este sistema levou a atrasos no progresso em todas as fases. Porém, ao ter desenvolvido este projecto sozinho, senti-me mais recompensado sempre que ultrapassei um obstáculo.

Penso que estas dificuldades foram mais sentidas durante o planeamento do protocolo de coordenação de máquinas a ser usado quando a versão 1 foi rejeitada após a apresentação. Acredito que dada a opção de debater os aspectos positivos e negativos da versão então seria capaz de melhorar a ideia e fortalecer a defesa dos seus méritos para a apresentação. Face à recusa do projecto por parte da autoridade do projecto, considerei que devia rejeitar a ideia o que me levou a começar a versão 2. Foi apenas quando agendei uma reunião online com o professor e orientador José Paulo Leal para discutir dificuldades na implementação da versão 2 que percebi como podia ultrapassar os defeitos da primeira ideia e criar uma nova versão que cumprisse os objectivos do projecto. Esta reunião teve consequência directa na criação da versão 3 (final) mas considero que se tivesse reconhecido e ultrapassado os defeitos da versão 1 antes da apresentação, então prevejo que o progresso do projecto pudesse ter sido encurtado em quase três semanas.

Quando me inscrevi no estágio, estava à procura de uma oportunidade para ganhar novos conhecimentos em programação. *JavaScript* era uma linguagem com a qual tinha pouca experiência e por essa razão, aproveitei para a escolher na implementação deste projecto, por ser apropriada para lidar com comunicação entre clientes e servidores e, de forma a ganhar experiência com a linguagem.

Houve uma “curva de aprendizagem” à medida que me habituava à natureza “*non blocking”* do *JavaScript* e estudava funções e módulos existentes que pudesse incluir no meu trabalho. Actualmente sinto me muito mais apto no uso eficaz da linguagem e confiante da minha capacidade em conceber projectos maiores ou aplicações em *JavaScript*.

Os objectivos definidos na proposta de estágio foram todos cumpridos e o produto final está pronto a ser usado. Penso que o resultado obtido foi satisfatório e com grande potencial para a fácil diversificação das capacidades dos agentes, a implementação de novas funcionalidades no gestor, e a criação de novos agentes singulares. No futuro, é sugerido que sejam implementadas novas barreiras de segurança para além das proporcionadas pelo protocolo HTTP e o serviço de *tunneling*, por exemplo encriptação de conteúdo das mensagens de forma as databases enviadas.

Foi uma experiência muito positiva que me permitiu ampliar as minhas técnicas de programação e ganhar experiência de trabalho nesta área da computação. Aprendi o valor de um bom planeamento inicial com a utilização de *milestones* em contexto de projecto, criando a pressão necessária para manter um progresso visível ao longo do seu desenvolvimento. Agradeço à Câmara da Maia pelo privilégio e oportunidade de participar no seu projecto.

Capítulo 5

# Apêndices

**Referências**

[1] Ngrok company website, How it works, <https://ngrok.com/product>

[2] Charles M. Kozierok, “The TCP/IP Guide” [Online] Available: <http://www.tcpipguide.com/free/index.htm>

[3]Charles M. Kozierok, “The TCP/IP Guide” [Online] Image available: <http://www.tcpipguide.com/free/t_TCPIPSNMPOperationalModelComponentsandTerminology-3.htm>

[4] Software testing help “Top 4 BEST NGROK alternatives in 2021” [Online] Available:

<https://www.softwaretestinghelp.com/ngrok-alternatives/>

**Anexos**

Anexo a: Funcionamento do protocolo na primeira fase:

<https://youtu.be/OyRZYJTLPps>

Nesta demonstração estão 4 terminais em uso: os primeiros 3 correspondem a agentes em ordem crescente de ID, a executar no *localhost* e o último corresponde ao gestor. O sistema está configurado para fazer uma leitura a cada 10 segundos com 3 agentes na rede.

Aos 10 segundos do vídeo, no gestor, é dada a ordem para os agentes começarem a leitura. Cada agente vai fazer as leituras intercaladamente, podemos olhar para a variável “time” em cada leitura:

Leituras Agente 1: ‘12:54:00’, ‘12:54:30’.

Leituras Agente 2: ‘12:54:10’, ‘12:54:40’.

Leituras Agente 3: ‘12:54:20’, ‘12:54:50’.

Na marca 1:45 min do vídeo, o gestor envia a ordem para a paragem dos agentes.

Anexo b: Funcionamento do protocolo na última fase: